Uso de Modelos de Linguagem de Grande Porte na Geração Automática de Artefatos de Teste End-to-End Baseados em Comportamento

Leonardo Serafin
Universidade do Vale do Itajaí
Itajaí, Brasil
serafin@edu.univali.br

Jhonatan Alves
Universidade do Vale do Itajaí
Itajaí, Brasil
jhonatan.alves@univali.com.br

Resumo

This paper presents a method for automatically generating behavior-based End-to-End testing artifacts using Large Language Models. The work targets the high cost and error-proneness of manually specifying and maintaining test artifacts in agile settings. We develop and evaluate a tool that, from target-system information provided by the user (e.g., URLs, UI elements, and execution prerequisites), incrementally derives user stories, acceptance criteria, BDD scenarios, and an executable Cypress test project. The approach leverages prompt engineering techniques—Few-Shot Learning and Least-to-Most Prompting—to improve structure and consistency across artifact levels. We validate the tool on a scheduling web application with authentication and persistent CRUD operations, representative of typical web systems. The evaluation measures utility, correction, and response time, showing high practical usefulness of the generated artifacts, low manual correction effort, and generation times compatible with interactive QA workflows. Overall, the results indicate that integrating LLM-driven artifact generation with web automation frameworks such as Cypress is a viable strategy to reduce specification effort while preserving traceability between behavior descriptions and automated tests.

Palavras-chave

Automação de Testes, Testes End-to-End, Behavior-Driven Development, LLMs, Engenharia de prompts, Cypress

1 Introdução

A busca pela qualidade de software tem evoluído de forma contínua desde os primeiros sistemas computacionais. Com o crescimento do mercado de tecnologia da informação e a adoção massiva de metodologias ágeis, a necessidade de entregar aplicações robustas e confiáveis em ciclos curtos de desenvolvimento tornou-se um dos principais desafios da engenharia de software [10]. Dentro desse contexto, práticas de automação de testes surgem como uma solução para mitigar falhas e garantir a estabilidade das aplicações [21]. Abordagens de teste como End-to-End (E2E) têm sido amplamente empregadas para validar o comportamento dos sistemas do ponto de vista do usuário final, abrangendo todos os componentes envolvidos em um fluxo funcional [9].

Avanços significativos na área de Inteligência Artificial, especialmente com o desenvolvimento de Modelos de Linguagem de Grande Porte (LLMs), têm proporcionado novas abordagens para a automação de tarefas repetitivas e a geração de artefatos textuais estruturados [18, 19]. Nesse cenário, as LLMs emergem como ferramentas essenciais para elevar a produtividade e a qualidade no desenvolvimento de sistemas [3, 20], potencial que se mostra particularmente relevante para ambientes que operam com equipes reduzidas e necessitam de entregas rápidas e frequentes [10].

O interesse no uso de LLMs para apoiar atividades de teste de software, como a geração de casos de teste e a análise de requisitos e documentação, tem crescido tanto na comunidade científica quanto no setor industrial [7]. Em paralelo, a automação da *execução* de testes E2E tem sido amplamente viabilizada por ferramentas como Cypress, Selenium e Playwright, que permitem interagir com a interface, simular ações do usuário e verificar comportamentos esperados de forma sistemática, aumentando a repetibilidade e favorecendo a detecção precoce de falhas [12].

Contudo, ainda são limitadas as soluções que automatizam, de forma integrada, as diferentes atividades do teste de software E2E, em especial as etapas que antecedem e sustentam a execução dos testes, como a *especificação* e a *manutenção* dos artefatos que descrevem o comportamento a ser validado. Diante desse cenário, as LLMs surgem como uma alternativa promissora para conectar as etapas dos testes reduzindo o esforço manual e padronizando resultados. Nesse contexto, esse trabalho propõe uma ferramenta para a geração automática de testes E2E integrando LLMs com metodologias consolidadas de engenharia de software e técnicas de engenharia de prompt com o objetivo de apoiar processos mais ágeis, eficientes e sustentáveis em ambientes de desenvolvimento modernos.

O restante desse trabalho está organizado da seguinte forma. Na seção 2, discutimos as limitações atuais do processo de concepção de testes End-to-End e caracterizamos o problema que motiva esta pesquisa. Na seção 3, apresentamos os conceitos essenciais que sustentam a proposta desenvolvida, abrangendo práticas de teste de software, automação, BDD e o uso de Modelos de Linguagem de Grande Porte na automação de teste. Em seguida, na seção 4, introduzimos a ferramenta proposta, detalhando sua arquitetura, funcionamento e fluxo de geração dos artefatos de teste. Na seção 5, discutimos os experimentos realizados e analisamos os resultados obtidos. Por sua vez, na seção 6, apresentamos os trabalhos correlatos e comparamos nossa solução às abordagens existentes. Por fim, na seção 7, apresentamos as conclusões e apontamos direções para trabalhos futuros.

2 Problema

Os testes End-to-End são fundamentais para garantir a qualidade de aplicações modernas sob a perspectiva do usuário final. Contudo, sua implementação enfrenta desafios significativos. A elaboração dos artefatos que antecedem a execução — como histórias de usuário, critérios de aceite, cenários BDD e scripts E2E — permanece

predominantemente manual e depende da interpretação detalhada das regras de negócio, análise de fluxos e compreensão de dependências entre telas e serviços. Trata-se de uma atividade intensiva, repetitiva, suscetível a erros e de difícil manutenção, especialmente em sistemas que evoluem rapidamente [3].

Embora existam diferentes ferramentas consolidadas para a automação da execução — como Cypress, Selenium e Playwright — a etapa de especificação e manutenção dos testes ainda carece de suporte adequado. Essa lacuna gera um descompasso: enquanto a execução pode ser automatizada e integrada a pipelines de CI/CD, a concepção e manutenção dos artefatos que alimentam a execução continuam dependentes de esforço humano. Como consequência, aumenta-se o tempo necessário para definir cenários de validação, ampliam-se os custos associados à manutenção dos testes e reduz-se a eficácia da detecção precoce de falhas.

Dessa forma, o problema central desse trabalho reside na falta de ferramentas capazes de automatizar, de maneira integrada, a especificação e a execução dos testes End-to-End reduzindo a dependência de esforço manual e promovendo maior padronização, consistência e reprodutibilidade do processo de teste.

3 Fundamentação

3.1 Teste de Software

Teste de software pode ser definido como o processo de verificação e validação de um programa com o objetivo de encontrar falhas e determinar se os requisitos são atendidos. Com o aumento da complexidade dos sistemas, os testes evoluíram para processos sistemáticos integrando-se às estratégias de garantia de qualidade (QA), processos ágeis e DevOps [2], tornando-se essenciais para reduzir riscos e aumentar a confiabilidade das aplicações [10].

Os testes são classificados sob diferentes perspectivas. Do ponto de vista de níveis, incluem-se os testes unitários, que validam componentes isolados; os testes de integração, que verificam a interação entre componentes; os testes de sistema, que avaliam o comportamento do sistema completo; e os testes de aceitação, que confirmam a aderência às expectativas do usuário final. Por outro lado, do ponto de vista de tipos, incluem-se os testes funcionais, destinados a validar comportamentos especificados, e os testes não funcionais, que avaliam atributos de qualidade como desempenho, segurança e usabilidade [6].

Nesse contexto, os testes End-to-End (E2E) configuram-se como testes funcionais executados no nível de sistema que simulam o comportamento do usuário final em um ambiente próximo à produção. Seu objetivo principal é garantir a integridade e a comunicação entre todas as camadas da aplicação (frontend, backend, banco de dados e APIs), detectando falhas de integração que testes isolados não costumam capturar [8].

Os testes E2E são estruturados em fluxos completos (como autenticação, cadastro, compra ou emissão de relatórios), validando a coerência entre transições de telas e regras distribuídas. Para documentar e operacionalizar esses fluxos, utilizam-se casos de teste e artefatos como histórias de usuário e critérios de aceite. Esses documentos conectam os requisitos à validação, servindo de base para a automação e garantindo a rastreabilidade do processo.

3.2 Automação de Testes

Os testes de software podem ser executados de forma manual ou automatizada. A execução manual, embora adequada para explorações iniciais e verificações não repetitivas, apresenta limitações significativas: pode ser demorada, sujeita a variabilidade humana e difícil de escalar em sistemas complexos. À medida que aplicações incorporam múltiplos fluxos e integrações, estratégias exclusivamente manuais tornam-se insuficientes para garantir confiabilidade, velocidade e reprodutibilidade.

Nesse cenário, a automação de testes se consolida como uma alternativa essencial, utilizando ferramentas e códigos para executar casos de teste de forma automática, sem a necessidade de intervenção humana direta. Os principais benefícios da automação incluem o aumento da eficiência em tarefas repetitivas, a ampliação da cobertura de testes e a liberação das equipes técnicas para atividades de maior valor analítico [4].

No contexto ágil, a automação torna-se estratégica para viabilizar a integração e entrega contínua (CI/CD), na qual os testes End-to-End desempenham papel essencial na validação rápida e repetitiva dos fluxos de negócio críticos. Sua complexidade, porém, demanda boas práticas como padrões de projeto e priorização de casos de teste, focando em caminhos principais para equilibrar cobertura e custo de manutenção [13].

Uma prática amplamente recomendada para mitigar o custo de manutenção dos testes E2E é a adoção do padrão de projeto Page Object. Esse padrão organiza a automação ao encapsular elementos e comportamentos de cada página ou componente da interface em módulos reutilizáveis, separando a lógica dos testes das estruturas específicas da interface do usuário. Dessa forma, alterações na camada visual (como ajustes em seletores, componentes ou fluxos de navegação) impactam apenas o objeto correspondente, evitando modificações dispersas em múltiplos scripts de teste. Essa abordagem aumenta a robustez da automação, reduz a incidência de testes frágeis, melhora a legibilidade e favorece a padronização do código de teste, contribuindo para a manutenção dos testes E2E [8].

3.3 Cypress

Considerando os desafios apresentados anteriormente, a escolha da ferramenta de automação torna-se um fator determinante para a qualidade, estabilidade e manutenibilidade dos testes End-to-End. Entre as soluções amplamente utilizadas no mercado, destaca-se o Cypress [17], cuja arquitetura e funcionalidades foram projetadas para a automação de testes em aplicações Web, destacando-se por simplificar a escrita, execução e depuração de testes E2E. Sua principal característica é executar os testes diretamente no navegador, o que permite acesso transparente ao DOM, controle de rede, sincronização automática e maior estabilidade na execução. Além disso, o Cypress fornece uma experiência de desenvolvimento orientada ao feedback rápido, permitindo depuração visual, *time travel* e relatórios detalhados. Essa abordagem integrada reduz a complexidade na configuração inicial e possibilita que equipes de QA, desenvolvimento e automação adotem a ferramenta de maneira colaborativa [17].

O Cypress possibilita simular, de forma realista, os fluxos percorridos pelo usuário na interface da aplicação. Além disso, por integrar

funcionalidades como interceptação de requisições, espera inteligente e isolamento de cenários, o Cypress contribui para reduzir a fragilidade típica dos testes E2E, que costumam ser sensíveis a mudanças na interface ou no comportamento assíncrono da aplicação. Assim, o Cypress consolida-se como uma ferramenta estratégica para a execução eficaz e confiável de testes End-to-End.

3.4 Behavior Driven Development

Embora ferramentas como o Cypress sejam fundamentais para a execução automatizada de testes End-to-End, a eficácia dessa automação depende diretamente da qualidade da especificação que orienta o que deve ser testado. Nesse sentido, o Behavior-Driven Development (BDD) surge como uma abordagem capaz de estruturar o comportamento esperado do sistema de maneira clara, verificável e alinhada ao usuário, servindo como base conceitual para a construção de testes E2E consistentes.

Diferentemente de técnicas tradicionais de especificação, o BDD enfatiza a descrição clara do comportamento esperado do sistema a partir da perspectiva do usuário final, utilizando cenários estruturados no formato *Given–When–Then*, geralmente escritos em Gherkin. Essa padronização favorece a redução de ambiguidades, aumenta a rastreabilidade entre requisitos e funcionalidades implementadas e possibilita que stakeholders não técnicos participem ativamente do processo de definição dos critérios de aceite [23].

Além disso, o BDD exerce impacto direto na estratégia de testes ao estabelecer uma ligação explícita entre comportamentos de negócio e verificações automatizadas. Os cenários descritos em linguagem natural fornecem não apenas documentação executável, mas também uma estrutura formal que pode ser transformada em testes funcionais automatizados. Dessa forma, o BDD orienta a implementação dos testes E2E, garantindo que eles validem com precisão os comportamentos especificados.

3.5 Modelos de Linguagem de Grande Porte no Teste de Software

Os Modelos de Linguagem de Grande Porte (LLMs) são sistemas de *deep learning* treinados em grandes volumes de dados textuais, capazes de compreender e gerar linguagem natural com coerência e contexto [16]. Por aprenderem padrões linguísticos a partir de extensos corpora, esses modelos executam diversas tarefas com alta proficiência, incluindo sumarização, tradução e geração de conteúdo técnico. Na engenharia de software, LLMs têm sido aplicados para reduzir o esforço manual em atividades como documentação, geração de código e elaboração de artefatos de teste. Embora contribuam para aumentar a produtividade e padronizar a produção desses artefatos, seu uso eficaz ainda enfrenta desafios, como a dependência de instruções bem formuladas, possíveis respostas imprecisas e limitada transparência sobre seus processos internos de raciocínio [1, 14].

Nesse contexto, a engenharia de prompt surge como uma estratégia essencial para orientar o comportamento dos modelos e mitigar suas limitações. Diversas técnicas têm sido propostas baseando-se no princípio comum de que a eficácia das respostas depende da formulação de instruções claras, estruturadas e contextualmente ricas. Assim, o papel da engenharia de prompt é fornecer ao LLM o enquadramento necessário para executar tarefas específicas de maneira consistente, precisa e alinhada aos objetivos estabelecidos. Portanto, na geração de artefatos de teste, os prompts assumem papel fundamental ao direcionar o modelo para produzir saídas coerentes com o domínio de negócio e com os requisitos funcionais envolvidos.

Entre as técnicas mais utilizadas, destaca-se o *Few-Shot Learning*, no qual o modelo recebe alguns exemplos de como a tarefa deve ser realizada antes de produzir novas respostas. Pesquisas mostram que exemplos bem formulados permitem ao modelo internalizar rapidamente o padrão desejado e gerar resultados alinhados aos requisitos, sem necessidade de treinamento adicional [11]. Outra técnica relevante é o *Least-to-Most Prompting*, que organiza o raciocínio do modelo em uma sequência de passos de complexidade crescente. A abordagem fundamenta-se na decomposição do problema em subtarefas menores, cujas respostas intermediárias auxiliam o modelo a construir soluções estruturadas e coerentes progressivamente [25].

Quando combinadas, as estratégias de *Few-Shot Learning* e *Least-to-Most Prompting* potencializam a capacidade dos LLMs de lidar com tarefas de maior complexidade, como a geração incremental de cenários de teste, critérios de aceite e descrições comportamentais em engenharia de software, oferecendo maior precisão e controle sobre o processo de geração. Contudo, além das técnicas de estruturação de prompts, a qualidade das respostas também é influenciada pelos parâmetros de inferência do modelo, em especial a *temperatura*, que determina o grau de aleatoriedade das saídas. Valores elevados tornam as respostas mais variadas e criativas, enquanto valores reduzidos favorecem maior estabilidade, previsibilidade e consistência técnica. No contexto da geração de artefatos de teste — que demanda precisão terminológica, aderência ao comportamento especificado e reprodutibilidade entre execuções — temperaturas baixas mostram-se mais adequadas, pois minimizam variações indesejadas e tornam o processo mais confiável. Dessa forma, a escolha da temperatura integra a própria estratégia de uso de LLMs na engenharia de software, influenciando diretamente a qualidade e a utilidade dos artefatos produzidos.

4 Desenvolvimento

Com o objetivo de apoiar equipes de QA e desenvolvimento na produção automatizada de artefatos de teste orientados por comportamento, propomos uma ferramenta baseada em LLMs que deriva, de forma incremental, histórias de usuário, critérios de aceite, cenários BDD e scripts E2E. Em vez de exigir a especificação manual desses artefatos, a solução consolida informações do sistema sob teste e as transforma em saídas alinhadas à lógica de negócio e ao comportamento esperado.

4.1 Arquitetura da Solução

A solução foi estruturada em uma arquitetura de três camadas — front-end, back-end e modelo de linguagem — conforme ilustrado na Figura 1. O front-end concentra a interação com o usuário, coleta as entradas do fluxo comportamental a ser testado e organiza instruções estruturadas para o modelo; o back-end atua como camada intermediária de orquestração, encaminhando as requisições ao serviço de LLM, normalizando as respostas para exibição no

front-end, além de isolar variáveis sensíveis e parâmetros de execução; e o modelo de linguagem, por sua vez, realiza a transformação incremental do contexto fornecido em artefatos de teste.

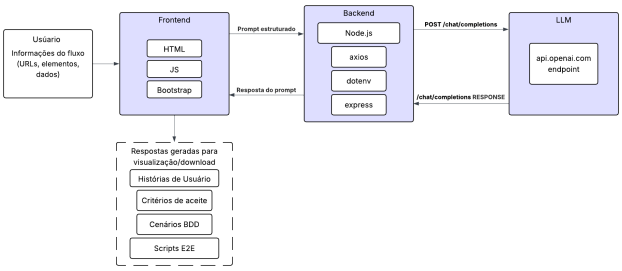

Figura 1: Arquitetura da ferramenta proposta para geração incremental de artefatos de teste.

O front-end foi construído como uma aplicação Web desenvolvida com HTML, Bootstrap e JavaScript. A interface dispara requisições sequenciais ao back-end e apresenta os resultados de forma gradativa em seções expansíveis (*accordions*), exibindo estado de carregamento durante o processamento e disponibilizando a exportação dos artefatos textuais em formato *Markdown*. O back-end, por sua vez, foi implementado em Node.js com *Express*, expondo uma API REST simples para receber as instruções do front-end e repassá-las ao provedor de LLM, retornando ao cliente um *payload* padronizado com o conteúdo gerado. A comunicação com a API é realizada por meio da biblioteca *axios*. Variáveis sensíveis (como chaves de acesso) são mantidas em arquivos `.env`, enquanto parâmetros de execução do modelo (como modelo selecionado e temperatura) são configurados no servidor para assegurar estabilidade e consistência entre execuções.

A partir dessa arquitetura, o funcionamento da ferramenta ocorre em três macroetapas: (i) coleta de contexto; (ii) geração incremental dos artefatos; e (iii) apresentação e exportação das saídas.

4.2 Coleta de contexto

O contexto necessário para a geração dos artefatos é fornecido pelo usuário, que descreve o fluxo comportamental a ser testado (como login, logout, agendamento, deleção ou edição). Para isso, informa-se: (i) as URLs que compõem o fluxo; (ii) os elementos de interface envolvidos na interação (como campos ou botões), juntamente com seus respectivos seletores CSS; e (iii) dados adicionais indispensáveis para uma execução consistente, como chamadas de API, cookies de autenticação, preparação de dados em banco e demais particularidades do domínio. Essas informações permitem que o modelo opere com contexto suficiente para produzir artefatos alinhados ao comportamento esperado.

4.3 Geração Incremental de Artefatos

Para gerar os artefatos de teste, a ferramenta emprega uma estratégia de *multi-prompting* baseada em *Least-to-Most Prompting*, em que os artefatos são gerados do nível mais abstrato ao mais prescritivo, e cada saída intermediária é incorporada como contexto na etapa seguinte. Essa condução progressiva favorece consistência entre os artefatos, permite reduzir ambiguidades e limita variações indesejadas entre execuções.

Em cada etapa do processo, aplica-se *Few-Shot Learning* por meio da inclusão de um conjunto fixo de exemplos do artefato-alvo, com o objetivo de ancorar o modelo em um padrão de escrita e reduzir variações indesejadas, estabelecendo formato, estilo e granularidade esperados. No primeiro prompt, as entradas obtidas da coleta de contexto são combinadas com exemplos de histórias de usuário; na etapa seguinte, o prompt incorpora as histórias geradas e exemplos de critérios de aceite; em seguida, as histórias e critérios de aceite passam a compor o contexto para a geração dos cenários BDD.

Por fim, a ferramenta reúne todo o contexto acumulado e cria um quarto prompt que orienta a LLM a produzir um conjunto de testes E2E para Cypress estruturado conforme o padrão Page Object. A resposta é retornada em formato JSON, contendo a estrutura de diretórios e arquivos do projeto Cypress, como `e2e` (diretório padrão onde ficam os arquivos de especificação dos testes End-to-End), `pages` (módulos no padrão Page Object que encapsulam seletores e ações reutilizáveis da UI), `fixtures` (dados estáticos, como usuários, *payloads* e estruturas JSON utilizadas nos testes) e `support` (arquivos de suporte e configuração, incluindo comandos customizados e inicializações globais), além do arquivo `cypress.config.js` (que contém parâmetros de execução, *baseUrl*, padrões de *specs* e demais configurações do framework).

4.4 Apresentação e Exportação de Artefatos

Na interface, as saídas são organizadas em seções expansíveis (*accordions*), uma para cada categoria de artefato gerado. Cada seção exibe o estado de processamento durante a requisição e, ao final, apresenta o conteúdo formatado, que pode ser exportado individualmente em *Markdown* para facilitar reutilização em documentação e repositórios.

No caso específico dos testes Cypress, a resposta estruturada em JSON é convertida pelo front-end em um arquivo `.zip` via biblioteca *JSZip*, contendo toda a estrutura necessária para a execução direta dos testes automatizados.

5 Resultados

Para avaliar a ferramenta proposta nesse trabalho, utilizamos uma aplicação Web de agendamentos contendo autenticação e operações CRUD persistentes, desenvolvida com *Express* para gerenciamento das rotas de API e *MongoDB* como banco de dados. A aplicação de teste reproduz características comuns de aplicações corporativas modernas, contemplando fluxos dependentes de estado de sessão, regras distribuídas e encadeamentos funcionais envolvendo múltiplas telas.

O processo de avaliação buscou determinar se os artefatos produzidos pela ferramenta seriam úteis e aplicáveis para equipes de QA e desenvolvimento, bem como quantos ajustes seriam necessários para torná-los plenamente executáveis. Foram selecionados cinco fluxos funcionais da aplicação: (i) Login; (ii) Logout; (iii) Agendamento de serviço; (iv) Deleção de agendamentos; e (v) Edição de agendamento. Para cada fluxo, a ferramenta gerou histórias de usuário, critérios de aceite, cenários BDD e scripts E2E para Cypress.

A avaliação dos resultados considerou três métricas por fluxo: Utilidade (U), Correção (C) e Tempo (T). A primeira mensura a

proporção de artefatos úteis, a segunda a proporção de artefatos que necessitam de ajustes e a terceira avalia o tempo necessário para gerar os artefatos. As métricas foram definidas da seguinte forma:

$$U = \left(\frac{N_{\text{úteis}}}{N_{\text{total}}} \right) \times 100 \tag{1}$$

$$C = \left(\frac{N_{\text{corrigidos}}}{N_{\text{total}}} \right) \times 100 \tag{2}$$

$$T = \sum_{k \in \{\text{HU, CA, BDD, E2E}\}} t_k \tag{3}$$

onde N_{total} representa o número total de artefatos gerados para um fluxo; $N_{\text{úteis}}$ representa o número de artefatos considerados úteis; $N_{\text{corrigidos}}$ corresponde ao número de artefatos que necessitaram de algum ajuste manual para tornarem-se plenamente utilizáveis. Para a métrica de tempo, considerada em segundos, o tempo para a geração dos artefatos é definido como a soma dos tempos das etapas `HU` (história de usuário), `CA` (critérios de aceite), `BDD` (cenários BDD) e `E2E` (scripts de testes E2E para Cypress), onde t_k representa o tempo de resposta da etapa k (i.e., o intervalo entre o envio da requisição dessa etapa e o recebimento completo da resposta).

Para a contabilização de $N_{\text{úteis}}$ e $N_{\text{corrigidos}}$, foi realizada uma inspeção manual dos artefatos gerados em cada execução. Considerou-se *útil* o conjunto de artefatos cuja estrutura e conteúdo estavam suficientemente alinhados ao fluxo descrito, permitindo o aproveitamento direto. Por sua vez, os artefatos contabilizados em $N_{\text{corrigidos}}$ necessitaram de edição manual para se tornarem úteis, isto é, executáveis, compreensíveis e semanticamente adequados ao contexto da aplicação.

Com relação à LLM utilizada, adotou-se o modelo *gpt-4o* com temperatura igual a 0,2, visando a estabilidade e previsibilidade na geração dos artefatos dado que valores baixos reduzem a aleatoriedade das respostas, resultando em saídas mais consistentes, técnicas e menos suscetíveis a variações indesejadas entre execuções.

Para cada execução de um fluxo, foram calculados os percentuais de Utilidade (U) e Correção (C) com base no total de artefatos gerados naquela execução, enquanto o tempo total de geração dos artefatos foi registrado em segundos. Dessa forma, cada execução produziu um valor para cada métrica avaliada. A Tabela 1 apresenta, para cada fluxo, a média (μ) e o desvio padrão (σ) dessas métricas ao longo de 10 execuções independentes.

Fluxo	Utilidade (%)	Correção (%)	Tempo (s)
Login	88,7±5,9	9,9±2,5	27,2±5,7
Logout	99,5±1,7	4,3±5,0	32,8±5,3
Agendamento de serviço	100,0±0,0	17,8±9,5	34,4±5,1
Deleção de agendamentos	100,0±0,0	4,5±6,7	40,6±4,2
Edição de agendamento	100,0±0,0	4,3±7,2	40,6±12,8

Tabela 1: Resultados para as métricas de Utilidade, Correção e Tempo.

Os resultados de Utilidade indicam uma tendência positiva conforme a complexidade do fluxo aumenta. Fluxos mais ricos em contexto — contendo dependências funcionais, dados de autenticação e uma estrutura mais detalhada de componentes — permitem à LLM compreender melhor o funcionamento da aplicação, resultando em artefatos mais completos e aproveitáveis. Esse comportamento é evidenciado nos fluxos de Edição, Deleção e Agendamento, que alcançaram as maiores médias de Utilidade. Em contraste, o fluxo de Login apresentou a menor média de Utilidade, uma vez que oferece um volume reduzido de informações ao modelo. Contudo, observou-se uma média geral de Utilidade de 97,6%, demonstrando que a LLM se beneficia de descrições com maior nível de detalhamento para produzir artefatos consistentes.

Observando os desvios padrão (σ) reportados na Tabela 1, nota-se a variabilidade entre execuções para cada fluxo. Em particular, a métrica de Utilidade apresentou σ entre 0,0 e 5,9, enquanto a métrica de Correção apresentou σ entre 2,5 e 9,5. Para o Tempo total, os desvios padrão ficaram entre 4,2 e 12,8, indicando que a variabilidade é mais pronunciada no tempo de resposta (possivelmente influenciado por latência/infraestrutura), enquanto a Utilidade mostrou-se mais estável nos fluxos mais ricos em contexto.

Por sua vez, a métrica de Correção apresenta comportamento inverso ao da Utilidade: quanto maior a complexidade do fluxo, menor o volume de ajustes necessários. Isso ocorre porque cenários ricos em contexto podem fornecer à LLM informações suficientes para estruturar corretamente as etapas e dependências entre as ações, fazendo com que necessitem de menos ajustes. Por outro lado, fluxos simples podem exigir mais intervenções manuais devido à falta de informações para a construção dos artefatos. Embora parte das execuções tenha exigido ajustes, a média geral de Correção foi de 8,2%, indicando baixo esforço de retrabalho na maioria dos casos. Ainda assim, o fluxo de Agendamento apresentou maior variabilidade e maior média de Correção (17,8±9,5), sugerindo maior sensibilidade a nuances de contexto e dependências funcionais específicas desse fluxo.

Entre as necessidades de ajustes identificadas, observam-se dois tipos principais: (i) ordenação inadequada de requisições, como acessar uma tela de deleção antes da criação do agendamento correspondente; e (ii) interpretações incorretas de componentes da interface, exigindo pequenos ajustes em seletores. Apesar disso, os artefatos que demandaram ajustes exigiram apenas correções pontuais, confirmando a viabilidade da solução e a facilidade de manutenção.

No que diz respeito ao tempo de geração dos artefatos, observou-se aumento conforme a complexidade do fluxo cresce, coerente com o maior volume de contexto e o nível de detalhamento exigido nas respostas. O tempo médio geral foi de 35,1±8,7 s, indicando viabilidade operacional para uso interativo por analistas de teste. Observou-se maior variabilidade de tempo no fluxo de Edição (40,6±12,8 s), e eventuais outliers podem estar associados a latência de infraestrutura e inicialização do serviço em modo *standby* após períodos de inatividade. Embora esse comportamento gere atrasos ocasionais, não comprometeu a aplicabilidade da solução nem a interpretação dos resultados.

De forma geral, os resultados demonstram que a ferramenta é capaz de gerar artefatos úteis, aplicáveis e com baixo nível de correção necessária, especialmente em fluxos que fornecem maior riqueza

de contexto para a LLM. A tendência de melhora nos fluxos mais complexos reforça a importância do detalhamento das entradas e da integração comportamental entre prompts, consolidando a abordagem como uma alternativa promissora para acelerar a automação de testes E2E baseados em comportamento.

5.1 Análise Crítica e Limitações

Os resultados apresentados devem ser interpretados à luz de algumas limitações. Primeiro, os experimentos foram conduzidos com o modelo *gpt-4o* e temperatura fixa igual a 0,2, escolhida para privilegiar estabilidade e reprodutibilidade. Embora temperaturas mais altas possam aumentar diversidade e potencialmente enriquecer descrições, elas tendem a elevar a variabilidade e a chance de inconsistências entre execuções. Assim, ainda é necessário explorar sistematicamente o impacto de diferentes temperaturas e estratégias de amostragem sobre as métricas reportadas.

Segundo, não foi realizado um estudo de custo por requisição. Em cenários reais, o custo depende do modelo utilizado, do volume de tokens de entrada/saída por etapa do pipeline e do preço vigente do provedor. Uma análise mais completa deve instrumentar a ferramenta para registrar tokens por requisição e, a partir disso, estimar custo médio por fluxo (e por artefato), permitindo discutir viabilidade econômica em diferentes escalas de uso.

Terceiro, a avaliação não incluiu uma comparação controlada com esforço humano. Embora a geração automática reduza trabalho manual, a comparação direta requer medir, com participantes (QAs e desenvolvedores), o tempo e a qualidade de artefatos produzidos manualmente versus a abordagem proposta, incluindo o esforço de revisão/ajuste. Essa análise é importante para quantificar ganhos de produtividade e entender a curva de adoção na prática.

Por fim, há ameaças à validade: (i) *validade de construto* — a classificação de "útil" e "necessita correção" foi baseada em inspeção manual, podendo introduzir subjetividade; (ii) *validade interna* — fatores como latência de rede, variações do serviço e não-determinismo residual do modelo podem afetar tempos e consistência, apesar do uso de temperatura baixa e múltiplas execuções; (iii) *validade externa* — os experimentos foram conduzidos em uma única aplicação e um conjunto limitado de fluxos, o que restringe generalização para outros domínios e níveis de complexidade; e (iv) *validade de conclusão* — o tamanho amostral por fluxo (10 execuções) limita inferências estatísticas mais robustas. Em trabalhos futuros devemos ampliar o conjunto de aplicações/fluxos avaliados, incorporar múltiplos avaliadores e aplicar análises estatísticas para fortalecer a evidência empírica.

6 Trabalhos Correlatos

A automação de testes End-to-End para aplicações Web tem sido investigada sob múltiplas abordagens, incluindo geração automática de casos de teste, técnicas de exploração/crawling da interface e estratégias para reduzir a fragilidade típica de testes de UI em cenários de regressão. Em paralelo, observa-se um movimento recente de incorporação de Inteligência Artificial — especialmente Processamento de Linguagem Natural, Modelos de Linguagem de Grande Porte (LLMs) e modelos visão-linguagem — para derivar artefatos de teste a partir de descrições em linguagem natural e/ou do estado observável da interface, ampliando o escopo da automação para além da simples execução.

Nessa seção, analisamos trabalhos correlatos que materializam esse potencial em ferramentas, técnicas e frameworks voltados à automação E2E. A comparação é conduzida segundo dimensões comuns: (i) fonte de entrada (texto, DOM/HTML, captura visual); (ii) escopo de saída (apenas script executável vs. artefatos intermediários como BDD/aceite); (iii) estratégias de confiabilidade e manutenção (mapeamento semântico de elementos, refino de seletores, agentes/exploração); e (iv) grau de integração do fluxo (geração, execução e validação/feedback). Com base nessas dimensões, discutimos como as abordagens existentes se aproximam e se diferenciam da solução proposta neste trabalho.

Sarkar et al. [22] propuseram um framework para converter histórias de usuário em linguagem natural em artefatos de teste executáveis em Java. A ferramenta realiza um pré-processamento do texto inicial a fim de extrair informações relevantes via Reconhecimento de Entidades Nomeadas, com atenção a termos de domínio e potenciais ambiguidades. Em seguida, utiliza modelos do tipo transformer para decompor a história em elementos semânticos como ator, ações, condições e resultados esperados, que são mapeados para casos de teste estruturados no formato BDD. A partir desses casos estruturados, a ferramenta sintetiza scripts de automação em Java que são compatíveis com Selenium/WebDriver e execução com JUnit/Cucumber.

Júnior et al. [15] apresentam uma ferramenta que, por meio de uma estratégia de *multi-level prompting*, transforma casos de teste descritos em linguagem natural em scripts E2E executáveis no Robot Framework. No nível 1, um prompt *zero-shot* converte o cenário em um JSON modular, no qual cada módulo representa uma página da aplicação (identificada pela URL) e contém uma sequência ordenada de passos de execução. No nível 2, os módulos são processados isoladamente: o sistema obtém o HTML/DOM via Crawl4AI e aplica um primeiro prompt para extrair, para cada passo, os elementos de interface e seus seletores; em seguida, um segundo prompt refina e valida esses seletores, melhorando a correspondência semântica entre elemento e ação. Por fim, no nível 3, um novo prompt *zero-shot* utiliza o JSON validado para gerar um teste completo e executável no Robot Framework, com comandos baseados na SeleniumLibrary.

Alian et al. [3] propuseram um framework para automação de testes E2E em aplicações web baseado em agentes e LLMs, organizado em um fluxo contínuo de descoberta, estruturação e validação de funcionalidades. Inicialmente, agentes exploram a interface e, com apoio da LLM, inferem funcionalidades a partir de elementos interativos; em seguida, constroem cooperativamente uma árvore hierárquica em que cada ramo representa um possível fluxo de uso relevante para os objetivos de teste. À medida que a árvore evolui, a LLM orienta quais ramos devem ser aprofundados com base no estado atual do sistema e na cobertura funcional desejada. Por fim, os caminhos dessa árvore são traduzidos em casos de teste E2E compostos por sequências de operações executáveis na interface, que são validadas em tempo real e registradas como cenários reprodutíveis.

Ayli et al. [5] propõem uma ferramenta para tornar testes automatizados de aplicações Web mais resilientes a mudanças na interface, na qual os casos de teste são inicialmente especificados em linguagem natural e representados em formato estruturado. A

partir dessa especificação, a ferramenta mapeia cada passo de teste para elementos da interface com base em similaridade semântica. Para isso, emprega técnicas de Processamento de Linguagem Natural e LLMs para identificar, entre os elementos disponíveis no DOM, aquele cujo rótulo ou contexto mais se aproxima da descrição textual fornecida, permitindo a execução automatizada e a manutenção robusta dos testes frente a alterações na UI.

Wang et al. [24] propõem uma técnica de teste automático de interfaces Web que utiliza Large Vision-Language Models (LVLMs) para integrar geração de entradas textuais e exploração guiada da UI em um ciclo de teste de ponta a ponta. O método endereça dois desafios principais: produzir valores de entrada válidos e contextualmente adequados para campos de texto e selecionar, a cada passo, o próximo elemento interativo a ser acionado em páginas complexas. Para isso, a ferramenta captura a screenshot da página, anota visualmente regiões de interesse e solicita ao LVLM a geração de valores apropriados, formulando em seguida a escolha do próximo elemento como um problema de visual question answering sobre a mesma imagem. A exploração do site é conduzida por um módulo de multi-armed bandit com estratégia de curiosidade, que decide quando explorar novos caminhos ou repetir ações promissoras. Os autores ainda apresentam variantes que combinam LVLM, LLM e heurísticas clássicas, compondo um loop de teste flexível e orientado por percepção visual e semântica da interface.

A Tabela 2 apresenta uma comparação entre os trabalhos correlatos e a proposta desse artigo.

Trabalho	Entrada	Saídas	Execução/validação	Integração do fluxo
Sarkar et al. [22]	Texto (histórias de usuário)	BDD + scripts em Java (Selenium)	Execução via JUnit/Cucumber	Geração da especificação ao script
Júnior et al. [15]	Texto + HTML/DOM (crawling)	Script E2E (Robot Framework)	Script pronto para execução	Geração (prompting multi-nível)
Alian et al. [3]	Exploração da UI (agentes)	Casos/fluxos E2E	Validação em tempo real	Descoberta + geração + validação
Ayli et al. [5]	Texto + DOM	Passos executáveis com mapeamento semântico	Resiliência a mudanças de UI	Foco em manutenção/robustez
Wang et al. [24]	Captura visual (LVLM)	Exploração guiada + entradas válidas	Ciclo E2E com exploração	Exploração/descoberta orientada por percepção
Proposta deste trabalho	Texto (descrição do fluxo)	HU + CA + BDD + projeto Cypress	Execução E2E no Cypress	Pipeline completo e incremental

Tabela 2: Comparação entre trabalhos correlatos e a abordagem proposta.

Em síntese, os trabalhos analisados convergem ao demonstrar que IA (NLP/LLMs/LVLMs) é capaz de reduzir esforço na automação de testes E2E, seja convertendo descrições textuais em passos executáveis, seja explorando a interface para descobrir fluxos. No entanto, observa-se que a nossa proposta se diferencia por articular, de forma integrada, quatro aspectos: **(i) uma metodologia de geração incremental**, que organiza a transformação do comportamento esperado em diferentes níveis de abstração; **(ii) maior cobertura de tipos de artefatos**, ao derivar desde histórias de usuário e critérios de aceite até cenários BDD e testes executáveis; **(iii) foco em reprodutibilidade e rastreabilidade**, ao manter encadeamento explícito entre os artefatos gerados; e **(iv) orientação à adoção prática**, ao empacotar a saída como um projeto de automação pronto para integração ao ambiente de testes.

Do ponto de vista metodológico, adotamos uma estratégia de geração incremental fundamentada em *Least-to-Most Prompting* e *Few-Shot Learning*, na qual o modelo é conduzido do nível mais abstrato ao mais prescritivo, incorporando cada saída como contexto da etapa subsequente para reduzir ambiguidades e variações entre artefatos. Em termos de cobertura, a solução não se limita ao script executável: ela deriva, de forma encadeada, *histórias de usuário*, *critérios de aceite* e *cenários BDD*, culminando na síntese de um *projeto* Cypress estruturado com Page Object. Essa cadeia, por sua vez, amplia a rastreabilidade entre especificação e automação, uma vez que cada artefato é conectado ao anterior e pode ser inspecionado como evidência do comportamento esperado. Por fim, a reprodutibilidade e adoção prática são favorecidas tanto pela padronização do processo (sequência fixa de etapas e exemplos de referência por tipo de artefato) quanto pela entrega do resultado em um projeto Cypress completo (diretórios `e2e`, `pages`, `fixtures` e `support`), pronto para importação e execução no ambiente de testes com mínima adaptação.

7 Conclusões

Esse trabalho investigou o uso de Modelos de Linguagem de Grande Porte (LLMs) para apoiar a geração automatizada de artefatos de teste orientados por comportamento. A ferramenta proposta, fundamentada em *Few-Shot Learning* e *Least-to-Most Prompting*, demonstrou viabilidade ao derivar, de forma incremental, histórias de usuário, critérios de aceite, cenários BDD e um projeto de testes E2E em Cypress estruturado em Page Object, reduzindo o esforço de especificação manual e aproximando a geração automática de um fluxo operacional para equipes de QA e desenvolvimento.

Os resultados indicaram alta utilidade dos artefatos gerados e baixa necessidade de correções, com tendência de melhora em fluxos mais complexos e ricos em contexto. Esse comportamento sugere que a qualidade e o detalhamento das entradas, aliados à estratégia incremental de *prompting*, contribuem para aumentar a consistência entre os níveis de abstração e a rastreabilidade entre especificação e automação. Além disso, o tempo médio de geração mostrou-se compatível com o uso prático da ferramenta em atividades de apoio ao teste.

Como trabalhos futuros, pretende-se: (i) investigar o impacto de diferentes configurações de geração do modelo, como variações de temperatura e estratégias de amostragem, de modo a compreender seus efeitos sobre estabilidade, consistência e qualidade dos artefatos gerados; (ii) incorporar mecanismos de instrumentação para registrar o consumo de tokens por etapa do pipeline, permitindo estimativas de custo por fluxo e análises de viabilidade em cenários de uso real; (iii) reduzir a dependência de entradas manuais por meio de mecanismos de identificação automática de elementos de interface e enriquecimento do contexto a partir do DOM/HTML; (iv) investigar estratégias de pós-processamento e validação automática dos artefatos gerados — incluindo mecanismos de auto-revisão e correção assistida — para diminuir retrabalho e aumentar a taxa de executabilidade; e (v) ampliar a avaliação empírica com diferentes aplicações, conjuntos de fluxos e participação de avaliadores humanos, possibilitando comparações controladas com esforço manual e fortalecendo a validade dos resultados obtidos.

Referências

[1] Toufique Ahmed, Prem Devanbu, Christoph Treude, and Michael Pradel. 2024. Can LLMs Replace Manual Annotation of Software Engineering Artifacts? *2025 IEEE/ACM 22nd International Conference on Mining Software Repositories (MSR)* (2024), 526–538. https://api.semanticscholar.org/CorpusID:271855028

[2] Samia Akhtar. 2025. Software Testing Evolution: Comparative Insights into Traditional and Emerging Practices. *ICCK Journal of Software Engineering* 1, 1

(2025), 46–62. https://doi.org/10.62762/JSE.2025.246843
[3] Parsa Alian, Noor Nashid, Mobina Shahbandeh, Taha Shabani, and Ali Mesbah. 2025. A Feature-Based Approach to Generating Comprehensive End-to-End Tests. (01 2025). https://doi.org/10.48550/arXiv.2408.01894 arXiv:2408.01894 [cs.SE]
[4] Mauricio Aniche. 2022. *Effective Software Testing: A Developer's Guide*. Manning Publications, Shelter Island, NY. 376 pages.
[5] Maroun Ayli, Youssef Bakouny, Nader Jalloul, and Rima Kilany. 2024. Enhancing the Resiliency of Automated Web Tests with Natural Language. In *Proceedings of the 2024 4th International Conference on Artificial Intelligence, Automation and Algorithms (AI2A '24)*. Association for Computing Machinery, New York, NY, USA, 63–69. https://doi.org/10.1145/3700523.3700536
[6] Rex Black, Erik van Veenendaal, and Dorothy Graham. 2024. *Foundations of Software Testing: ISTQB Certification* (5 ed.). Cengage Learning EMEA, London. 288 pages. ISBN-10: 1473795885.
[7] George Daniel Marques Borges and Sósthenes Oliveira Lima. 2024. Inteligência Artificial Aplicada à Qualidade de Software. *Sistemas de Informação* 28, 138 (Sept. 2024). https://doi.org/10.69849/revistaft/th10249181506 Publicado em: 18 set. 2024.
[8] Beata Bylina and Agnieszka Antończak. 2024. Analysis of end-to-end test automation tools based on the examples of Selenium WebDriver and Playwright. In *Conference on Computer Science and Information Systems*. https://api.semanticscholar.org/CorpusID:274333402
[9] Letícia Santos Camargo. 2024. A utilização de testes automatizados no processo de garantia de qualidade de interfaces de aplicações web: um mapeamento sistemático. https://repositorio.ifgoiano.edu.br/bitstream/prefix/4677/1/TCC_Let%C3%ADcia%20Santos%20Camargo.pdf Acesso em: 25 maio 2025.
[10] Nivanderson Coutinho and Emerson Nascimento. 2025. Desafios e benefícios da implementação de testes automatizados em empresas de software. *Cuadernos de Educación y Desarrollo* 17 (05 2025), e8221. https://doi.org/10.55905/cuadv17n4-176
[11] Andrea Gajic and Sudip Vhaduri. 2025. A Comprehensive Survey of Challenges and Opportunities of Few-Shot Learning Across Multiple Domains. arXiv:2504.04017 [cs.LG] https://arxiv.org/abs/2504.04017
[12] Boni García, José M. del Álamo, Maurizio Leotta, and Filippo Ricca. 2024. Exploring Browser Automation: A Comparative Study of Selenium, Cypress, Puppeteer, and Playwright. In *Quality of Information and Communications Technology*. https://api.semanticscholar.org/CorpusID:273649400
[13] Jez Humble and David Farley. 2010. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley Professional.
[14] Haolin Jin, Linghan Huang, Haipeng Cai, Jun Yan, Bo Li, and Huaming Chen. 2024. From LLMs to LLM-based Agents for Software Engineering: A Survey of Current, Challenges and Future. *ArXiv* abs/2408.02479 (2024). https://api.semanticscholar.org/CorpusID:271709396
[15] Elvis Júnior, Alan Valejo, Jorge Valverde-Rebaza, and Vânia de Oliveira Neves. 2025. GenIA-E2ETest: A Generative AI-Based Approach for End-to-End Test Automation. In *Anais do XXXIX Simpósio Brasileiro de Engenharia de Software (SBES 2025) (SBES 2025)*. Sociedade Brasileira de Computação, 282–292. https://doi.org/10.5753/sbes.2025.9927
[16] Pranjal Kumar. 2024. Large language models (LLMs): survey, technical frameworks, and future challenges. *Artificial Intelligence Review* 57 (2024). https://api.semanticscholar.org/CorpusID:271961846
[17] Waweru Mwaura. 2021. *End-to-End Web Testing with Cypress: Explore techniques for automated frontend web testing with Cypress and JavaScript* (1 ed.). Packt Publishing.
[18] Caio Monteiro de Oliveira. 2024. Utilização de chat bots baseados em LLMs para automação de testes de software. https://www.monografias.ufop.br/bitstream/35400000/6440/3/MONOGRAFIA_Utiliza%C3%A7%C3%A3oChatsBots.pdf Acesso em: 25 maio 2025.
[19] Giusepe Tessari Lopes de Oliveira. 2024. Aplicação de LLMs na modelagem de requisitos: melhorando a criação de diagramas UML e estimulando a criatividade em novos casos de uso. http://hdl.handle.net/10183/284220 Acesso em: 25 maio 2025.
[20] Ahmed Ramadan, Husam Yasin, and Burhan Pektas. 2024. The Role of Artificial Intelligence and Machine Learning in Software Testing. arXiv:2409.02693 [cs.SE] https://arxiv.org/abs/2409.02693
[21] Karina de Jesus Rodrigues and Renata Mirella Farina. 2019. Qualidade de software, utilizando teste automatizado. In *Anais do IV Encontro de Iniciação Científica e Tecnológica (EnICT)*. https://arq.ifsp.edu.br/eventos/enict/4EnICT/paper/view/353 Acesso em: 25 maio 2025.
[22] Ankur Sarkar, S A Mohaiminul Islam, and MD Shadikul Bari. 2024. Transforming User Stories into Java Scripts: Advancing Qa Automation in The Us Market With Natural Language Processing. *Journal of Artificial Intelligence General science (JAIGS) ISSN:3006-4023* 7, 01 (None 2024), 9–37.
[23] John Ferguson Smart and Jan Molak. 2023. *BDD in Action, Second Edition: Behavior-Driven Development for the Whole Software Lifecycle* (2 ed.). Manning Publications.
[24] Siyi Wang, Sinan Wang, Yujia Fan, Xiaolei Li, and Yepang Liu. 2024. Leveraging Large Vision Language Model For Better Automatic Web GUI Testing. arXiv:2410.12157 [cs.SE] https://arxiv.org/abs/2410.12157
[25] Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, and Ed Chi. 2023. Least-to-Most Prompting Enables Complex Reasoning in Large Language Models. arXiv:2205.10625 [cs.AI] https://arxiv.org/abs/2205.10625