Ordenação adaptativa de casos de teste manuais apoiada em estruturas de prefixos compartilhados (trie/DAG)

Marcelo Saraiva
juniormarcelo443@gmail.com
Universidade Federal do Acre
Rio Branco, Acre, Brasil

Jânio Dias
janio.dias@sou.ufac.br
Universidade Federal do Acre
Rio Branco, Acre, Brasil

Renilson Oliveira
renilson.oliveira@sou.ufac.br
Universidade Federal do Acre
Rio Branco, Acre, Brasil

Cleuton Almeida
cleuton.almeida@ufac.br
Universidade Federal do Acre
Rio Branco, Acre, Brasil

Catarina Costa
catarina.costa@ufac.br
Universidade Federal do Acre
Rio Branco, Acre, Brasil

Laura Costa
laura.sarkis@ufac.br
Universidade Federal do Acre
Rio Branco, Acre, Brasil

Nícolas Riccieri
nicolas@motorola.com
Universidade Federal do Acre
Rio Branco, Acre, Brasil

Olacir Castro
olacir.junior@ufac.br
Universidade Federal do Acre
Rio Branco, Acre, Brasil

Resumo

A execução manual de testes de software, embora essencial em cenários que exigem julgamento humano, apresenta elevado custo cognitivo devido à redundância de passos e à complexidade do gerenciamento de contexto. Abordagens tradicionais de priorização de testes, voltadas principalmente à automação, não consideram adequadamente fatores humanos e dependências de estado, evidenciando uma lacuna na literatura. Este trabalho propõe um sistema de recomendação para ordenação e consolidação adaptativa de casos de teste manuais, com o objetivo de reduzir a repetição de passos e apoiar a tomada de decisão do testador. A pesquisa adota a abordagem de Design Science Research (DSR), envolvendo o desenvolvimento de um artefato baseado em normalização semântica, classificação de impacto no contexto e uso de estruturas de prefixos compartilhados (trie/DAG). A avaliação foi conduzida por meio de experimentos com diferentes suítes de teste e análise de métricas quantitativas e qualitativas. Os resultados indicam redução estrutural de até 19,7% no número de passos na sequência sugerida, além de melhor organização da sequência de testes, evidenciando o potencial da abordagem como apoio à execução manual. A efetividade em detecção de falhas após consolidação não é medida neste estudo; as limitações quanto à validação com testadores em contexto organizacional são discutidas na Seção 6.

Palavras-Chave

Teste de Software; Casos de Teste; Sistemas de Recomendações; Trie/DAG.

1 Introdução

A atividade de teste de software constitui um dos principais mecanismos para garantia da qualidade de sistemas complexos [16] [12]. Embora avanços significativos tenham sido alcançados em automação de testes, testes manuais continuam desempenhando papel essencial, especialmente em cenários nos quais a automação é inviável ou insuficiente. Isso ocorre, por exemplo, em atividades que exigem julgamento humano, interpretação contextual e avaliação subjetiva [12] [8], como testes de usabilidade, percepção visual e comportamento dinâmico de interfaces [20].

Apesar de sua importância, a execução manual de suítes extensas de testes impõe elevado custo cognitivo e operacional aos testadores. Essa atividade demanda atenção contínua, memória de trabalho e tomada frequente de decisões [2] [7]. O problema se intensifica quando os casos de teste apresentam sobreposição de passos, dependências práticas de contexto entre fluxos ou exigem reinicializações ou repreparação frequente do ambiente. Nesses contextos, a execução tende a se tornar mais suscetível a retrabalho, interrupções e aumento do esforço mental, especialmente para testadores menos experientes.

Na prática, testadores experientes frequentemente reorganizam a ordem de execução dos testes para reduzir esse esforço. Essa reorganização busca reutilizar estados previamente estabelecidos e evitar ações que comprometam execuções subsequentes. Casos de teste associados a uma mesma funcionalidade, por exemplo, costumam compartilhar etapas iniciais ou estruturas semelhantes, o que permite otimizações durante a execução [4]. Esse comportamento sugere que a execução manual de testes é um processo adaptativo, fortemente influenciado pelo estado do sistema e pela experiência do testador.

Contudo, é necessário distinguir o problema tratado neste trabalho das formulações clássicas de *Test Suite Prioritization* (TSP) e *Test Suite Reduction* (TSR). A TSP ordena *casos de teste* inteiros, em geral visando detectar falhas cedo em regressão automatizada [3, 19, 22]. A TSR seleciona um subconjunto de casos redundantes [22]. Em ambas, a unidade central costuma ser o caso e o executor frequentemente é automatizado. Neste trabalho a unidade de análise é o *passo normalizado*; o objetivo é produzir uma *sequência linear consolidada* que reduza repetição estrutural de passos semanticamente equivalentes, respeitando impacto destrutivo sobre o contexto e preservando pontos de validação por caso. Trata-se, portanto, de consolidação e sequenciamento de passos em testes manuais — parente conceitual mais próximo da redução em nível de passo do que da priorização clássica de casos — operando como recomendação

human-in-the-loop (HITL): o sistema sugere; o testador executa e decide.

No entanto, técnicas tradicionais de priorização de casos de teste foram majoritariamente concebidas para ambientes automatizados, com foco em testes de regressão [19] [3] [22]. Essas abordagens geralmente assumem que os testes são independentes, executados a partir de um estado inicial consistente e com custos similares. Tais premissas, embora adequadas para automação, não refletem integralmente a dinâmica da execução manual.

Em testes manuais, diferentes ações exercem impactos distintos sobre o contexto disponível para os passos seguintes. Ações de caráter predominantemente verificador tendem a preservar o contexto atual para fins de sequenciamento, enquanto ações que alteram ou encerram fluxos — como criação, edição, remoção de dados ou encerramento de telas — podem exigir repreparação ou tornar mais custosas execuções posteriores que dependiam do contexto anterior. Ignorar essas diferenças pode levar a sequências de execução ineficientes, com quebras de contexto repetidas e aumento do esforço do testador.

Embora a literatura reconheça a importância de dependências de estado em contextos como teste baseado em modelos [1] [21], revisões recentes indicam que a maior parte das abordagens de priorização ainda negligencia fatores humanos e a execução manual [22]. Observa-se, portanto, uma lacuna na proposição de técnicas que considerem, de forma integrada, o papel ativo do testador, a redução de redundância entre casos e o impacto dos passos sobre a continuidade da sessão manual — sem depender exclusivamente de automação ou de modelos completos do sistema sob teste.

Para enfrentar essa lacuna, este trabalho propõe um sistema de recomendação que apoia a organização da execução de suítes de teste manuais, por meio da consolidação e ordenação de passos com base em normalização semântica, classificação de impacto no contexto e uma estrutura de prefixos compartilhados — uma árvore de prefixos (*trie*) que, após a fusão (*merge*) de trechos equivalentes, pode resultar em um grafo acíclico dirigido (DAG, do inglês *directed acyclic graph*) sobre passos normalizados. O sistema não substitui o testador na execução nem infere automaticamente o estado interno do produto; em vez disso, infere rótulos sobre os passos (por exemplo, verificação versus ação e destrutividade em relação ao contexto de execução) e sugere uma sequência linear que reduz repetição de trechos comuns, explicitando onde cada caso é validado e registrando indicadores associados à qualidade da consolidação.

A abordagem combina heurísticas de ordenação no grafo de passos (por exemplo, priorização de verificações e postergação de ramos que quebram o contexto, com re-setups quando necessário) com métricas que sintetizam aspectos da recomendação, entre eles: (i) impacto dos passos no contexto (passos classificados como destrutivos, re-setups); (ii) profundidade de *merge* na estrutura de prefixos compartilhados (associada à consolidação de ramos, e não apenas à contagem trivial de testes com o mesmo prefixo textual); (iii) indicadores ligados à necessidade de repreparação da sequência (re-setups); e (iv) trocas de contexto entre trechos da sequência otimizada. Com base nesses elementos, o sistema gera uma proposta de execução da suíte, visando reduzir redundância de passos e apoiar decisões do testador. A partir dessa motivação, este trabalho busca responder às seguintes questões de pesquisa:

- **QP1.** Em que medida um sistema de recomendação baseado em normalização semântica e estruturas de prefixos compartilhados (*trie*/DAG) sobre passos consegue reduzir a redundância estrutural na execução de suítes de testes manuais, considerando dependências de contexto entre passos?
- **QP2.** Como diferentes modos de inferência (heurístico local versus modelo de linguagem remoto) impactam o comportamento estrutural da recomendação e o custo computacional do pipeline de otimização?

As principais contribuições deste trabalho são:

(i) Um modelo de sequenciamento baseado em passos normalizados em estrutura *trie*/DAG, que representa o reuso de prefixos e marcações de validação por caso, apoiado em classificação binária de impacto no contexto (verificação *versus* ação; destrutividade e regras de quebra de contexto);
(ii) Um conjunto de heurísticas de priorização na geração da sequência, acompanhado de métricas agregadas que refletem impacto no contexto, consolidação por *merge*, *re-setups* e trocas de contexto, oferecendo leitura objetiva do *trade-off* entre continuidade e redundância;
(iii) Um protótipo funcional e uma avaliação experimental quantitativa em três suítes de teste de tamanhos crescentes (10, 20 e 50 casos), com três repetições independentes por configuração, comparando o provedor heurístico local com o modelo de linguagem de grande escala (LLM) remoto (gpt-4o-mini), evidenciando reduções estruturais de até 19,7% e fator de lentidão da inferência remota entre 270× e 329× em relação ao *baseline* local;
(iv) Um módulo de visualização (sequência otimizada e estrutura em árvore) que explicita os fatores considerados na recomendação e facilita a análise pelo testador.

Diferentemente de abordagens tradicionais centradas em execuções majoritariamente automatizadas e em ordenações que ignoram a dinâmica humana, esta proposta considera o contexto da execução manual, a reutilização de passos entre casos e o papel do testador como decisor final, funcionando como apoio à decisão na execução de suítes de teste. O caráter adaptativo entre sessões é reforçado pela reutilização de normalizações e classificações armazenadas (base de conhecimento), reduzindo retrabalho em execuções subsequentes.

O restante deste artigo está organizado da seguinte forma. A Seção 2 apresenta os conceitos e os trabalhos relacionados. A Seção 3 descreve a proposta do sistema. A Seção 4 detalha o método de pesquisa adotado. A Seção 5 apresenta os resultados experimentais e a respectiva discussão. A Seção 6 discute as ameaças à validade e as limitações do estudo. Por fim, a Seção 7 apresenta as conclusões e os trabalhos futuros.

2 Conceitos e Trabalhos Relacionados

2.1 Conceitos Principais

2.1.1 Testes Manuais e Sobrecarga Cognitiva. A atividade de teste de software constitui um dos principais mecanismos para a garantia da qualidade, confiabilidade e robustez de sistemas complexos, sendo amplamente reconhecida como etapa crítica do ciclo de desenvolvimento [17] [13]. Apesar de avanços substanciais em automação de testes, a execução manual permanece essencial em cenários que

exigem julgamento humano, interpretação contextual e avaliação subjetiva, como testes de usabilidade, inspeção visual de interfaces e validação de experiência do usuário [8] [20].

Diferentemente da execução automatizada, testes manuais dependem intensamente de processos cognitivos humanos, incluindo atenção sustentada, memória de trabalho e tomada contínua de decisões [2] [7]. Esse esforço cognitivo é intensificado em suítes extensas, nas quais casos de teste apresentam sobreposição de etapas, exigem frequentes reinicializações do sistema ou demandam constante alternância de contexto operacional.

Estudos recentes destacam que a execução manual não ocorre de forma estritamente sequencial ou rígida; testadores experientes frequentemente reorganizam dinamicamente a ordem de execução com o objetivo de reduzir retrabalho e otimizar o reaproveitamento de estados previamente estabelecidos [4] [6]. Esse comportamento evidencia que a execução manual de testes constitui um processo adaptativo, fortemente influenciado por fatores cognitivos e pela evolução do estado do sistema.

2.1.2 Dependências de Estado e Modelagem por Grafos. Técnicas tradicionais de priorização de casos de teste foram majoritariamente concebidas para ambientes automatizados, com foco predominante em testes de regressão [19] [3] [22]. Nessas abordagens, assume-se que os casos de teste são executados de forma independente e partem de estados iniciais consistentes, premissa adequada para automação, mas frequentemente inválida no contexto manual.

Em sistemas interativos, ações de teste provocam transições de estado que alteram configurações internas, estruturas de dados e modos operacionais do sistema. Consequentemente, diferentes casos de teste tornam-se interdependentes, e sua ordem de execução pode facilitar ou inviabilizar verificações subsequentes. Ações não destrutivas preservam o contexto atual, enquanto ações destrutivas ou transformadoras podem modificar o estado de forma irreversível, exigindo reinicializações custosas do ambiente de teste.

A importância de dependências de estado é amplamente reconhecida em áreas como teste baseado em modelos e verificação formal, onde o comportamento do sistema é representado por estados e transições [1] [21]. Nessas abordagens, grafos direcionados e máquinas de estados finitos são empregados para modelar explicitamente relações de dependência e identificar sequências válidas de execução.

No contexto deste trabalho, a dependência entre casos não é modelada por um autômato explícito do sistema sob teste, mas por prefixos compartilhados sobre passos normalizados: uma árvore de prefixos (*trie*) que, após fusão de trechos equivalentes, pode resultar em um grafo acíclico dirigido (DAG) sobre passos (Seção 3.2). Essa representação captura reuso estrutural e impacto de ações sobre o contexto de execução sem exigir modelo completo do produto.

2.1.3 Sistemas de Recomendação. Revisões sistemáticas indicam que a maioria das abordagens de priorização de testes negligencia fatores humanos e assume execução totalmente automatizada [22] [6]. No entanto, em cenários manuais, o testador exerce papel central na tomada de decisão, ajustando dinamicamente estratégias de execução com base na experiência e no contexto operacional.

Sistemas de recomendação HITL têm sido aplicados em domínios que exigem adaptação contextual e supervisão humana, como apoio à decisão médica, curadoria de conteúdo e sistemas educacionais inteligentes. No contexto deste trabalho, esse paradigma é aplicado à ordenação de casos de teste manuais por meio de um mecanismo de recomendação que combina normalização/classificação de passos e consolidação estrutural da sequência. Na implementação atual, a recomendação é produzida de forma predominantemente em lote a cada submissão de suíte, considerando impacto dos passos no contexto e métricas de execução; a adaptação ao longo do tempo ocorre principalmente entre sessões, pela reutilização de conhecimento armazenado (base de normalizações e classificações), e não por replanejamento contínuo durante cada passo executado pelo testador.

2.2 Trabalhos Relacionados

A priorização de casos de teste tem sido amplamente investigada na literatura de engenharia de software, tradicionalmente com foco em testes automatizados e cenários de regressão. Abordagens clássicas buscam maximizar a detecção precoce de falhas por meio de métricas como cobertura de código, análise de impacto e histórico de execuções. Mais recentemente, técnicas baseadas em aprendizado de máquina, linguagem natural e métodos híbridos têm sido exploradas para melhorar a eficácia dessas estratégias. Entretanto, grande parte dessas abordagens ainda assume independência entre casos de teste e execução automatizada, negligenciando dependências de estado e fatores humanos presentes na execução manual.

Khan et al. [9] propõem uma técnica de priorização baseada em aprendizado de máquina com otimização de hiperparâmetros, na qual modelos preditivos utilizam métricas históricas de execução para estimar a probabilidade de detecção de falhas. Os resultados indicam ganhos de eficácia em comparação com heurísticas tradicionais. Contudo, a abordagem é voltada exclusivamente para testes automatizados de regressão e não considera interdependências de estado entre testes.

Li et al. [11] apresentam o método VRank para priorização de entradas de teste em sistemas baseados em *deep learning*, utilizando técnicas de ranqueamento para identificar entradas com maior probabilidade de revelar falhas de classificação. Embora demonstre o potencial de técnicas inteligentes para ordenação de testes, o trabalho é restrito a entradas de dados para modelos de aprendizado profundo e não aborda execução manual.

Rao e Nandal [18] propõem uma abordagem dinâmica de priorização de testes de regressão baseada em aprendizado supervisionado, na qual a ordenação é continuamente ajustada conforme métricas de execução são coletadas. Apesar de explorar adaptação dinâmica, o método permanece centrado em ambientes automatizados e não considera dependências de estado nem de contexto de ações durante a execução.

Garg e Shekhar [5] utilizam um algoritmo evolucionário multiobjetivo (NSGA-II) para priorização de casos de teste com base em sensibilidade a falhas e custo de execução. A técnica busca gerar ordenações equilibradas por meio de otimização heurística eficiente. Contudo, a proposta não modela explicitamente dependências entre testes nem contempla execução manual.

Martes et al. [14] apresentam uma análise detalhada de modelos baseados em transformadores para a tarefa de detecção de paráfrases, utilizando o Microsoft Research Paraphrase Corpus (MRPC)

como base de avaliação. Foram comparados três modelos de linguagem pré-treinados: BERT, RoBERTa e MPNet, em combinação com quatro métricas de similaridade: similaridade cosseno, produto escalar, distância Euclidiana e distância Manhattan. Os resultados destacam a importância da escolha adequada de métricas de similaridade e *thresholds* para otimizar o desempenho dos modelos, além de evidenciar a superioridade do MPNet em relação aos outros modelos testados.

Latina et al. [10] investigam o uso de técnicas avançadas de Processamento de Linguagem Natural (PLN) para o aprimoramento de sistemas de detecção de plágio. O estudo propõe um modelo híbrido que combina as capacidades do Word2Vec, para capturar relações semânticas, e do BERT, para representações contextuais profundas. Utilizando uma arquitetura de aprendizado de máquina baseada em *stacking ensemble* (combinando Regressão Logística e XGBoost), os autores alcançaram uma acurácia de 93% na detecção de plágios fortemente parafraseados, superando as limitações de ferramentas tradicionais que dependem apenas da correspondência exata de palavras. Apesar da alta eficácia na identificação de similaridades em textos estáticos para fins de integridade acadêmica, o trabalho deixa lacunas quanto à aplicação dessas métricas de similaridade em fluxos de trabalho dinâmicos ou operacionais. A técnica foca na detecção passiva de conteúdo, sem considerar o impacto que a similaridade semântica pode ter na otimização de esforços humanos ou na gestão de contextos em atividades sequenciais.

Em síntese, embora avanços recentes explorem aprendizado de máquina, heurísticas híbridas e algoritmos evolucionários para priorização de testes, os trabalhos existentes concentram-se predominantemente em ambientes automatizados e desconsideram aspectos como dependências de estado, contexto de ações e fatores humanos. Esse cenário evidencia a necessidade de abordagens que integrem modelagem explícita de estados, reutilizações de contexto e aprendizado incremental, como proposto neste trabalho.

Tabela 1: Comparativo entre os trabalhos relacionados de priorização e a proposta deste trabalho.

Trabalho	Man.	Est.	HITL	Reu.	Adapt.
Khan et al. [9]	–	–	–	–	–
Li et al. [11]	–	–	–	–	–
Rao & Nandal [18]	–	–	–	–	✓
Garg & Shekhar [5]	–	–	–	–	–
Este trabalho	✓	✓	✓	✓	✓

Legenda. Man.=aplicabilidade a testes manuais; Est.=dependências de estado/contexto; HITL=*human-in-the-loop*; Reu.=reuso estrutural de prefixos; Adapt.=adaptação entre execuções.
Nota. Martes et al. [14] e Latina et al. [10] são citados como apoio à normalização semântica e não compõem este comparativo direto.

3 Proposta

A proposta é um sistema de recomendação voltado à ordenação e consolidação adaptativa de casos de teste manuais, com o objetivo de reduzir o esforço do testador humano e diminuir a execução redundante de passos quando vários casos compartilham trechos semelhantes ou equivalentes.

O sistema foi concebido para apoiar a decisão do testador: não executa os testes de forma automática no produto sob teste, mas sugere uma sequência de execução em que passos repetidos são, quando possível, unificados, mantendo o rastreio de quais casos são cobertos em cada ponto da sequência.

A recomendação leva em conta o impacto dos passos sobre o contexto de execução. Cada passo é analisado quanto ao tipo (verificação ou ação) e quanto à alteração ou destruição do contexto atual. Com base nisso, o motor de otimização ordena ramos compatíveis e, quando necessário, prevê reexecução de prefixos (re-setups) após passos ou ramos que encerram ou invalidam o contexto anteriormente estabelecido.

O caráter adaptativo manifesta-se sobretudo pela persistência e reutilização de conhecimento: normalizações e classificações de passos são armazenadas (base de conhecimento em banco de dados), de modo que execuções posteriores se beneficiem de resultados já obtidos, reduzindo chamadas redundantes ao módulo de inferência (por exemplo, modelos de linguagem) e refinando o comportamento do sistema ao longo do uso.

3.1 Visão Geral do Sistema

O sistema apoia testadores na execução manual de suítes: em vez de apenas reordenar casos, produz uma sequência linear de passos em que prefixos comuns podem ser percorridos uma única vez, indicando onde cada caso atinge seu ponto de validação. A recomendação combina ordenação (quando favorece sobreposição) e fusão de passos sob uma *trie* de prefixos que, após consolidações, forma um DAG sobre passos normalizados — e não sobre casos inteiros ligados por dependências de estado explícitas.

A arquitetura pode ser descrita em três componentes principais:

(1) Interface de entrada, na qual o testador informa a suíte por: criação manual de casos e passos; texto estruturado ou livre (incluindo colagem direta); ou importação de arquivos nos formatos CSV, JSON, Excel (XLS/XLSX) e texto (por exemplo TXT tratado como conteúdo textual para o mesmo parser de casos e passos).
(2) Módulo de recomendação, responsável pelo pipeline de processamento: normalização semântica dos passos (com apoio de modelo de linguagem ou de heurísticas locais), classificação dos passos, reordenação dos casos para favorecer compartilhamento de prefixos, construção da *trie*/DAG, geração da sequência (por exemplo, via percurso em profundidade com critérios de prioridade) e cálculo de métricas. O resultado é uma proposta de ordem de execução materializada como lista de passos com metadados (tipo, destrutividade, validações, re-setups).
(3) Visualização e análise, em que o testador inspeciona a recomendação em lista (sequência otimizada) e, quando disponível, em árvore (estrutura derivada da *trie*), além de métricas como: redução de passos e percentual economizado; grupos e identificadores normalizados; passos destrutivos; re-setups; trocas de contexto; e profundidade de *merge* na estrutura de prefixos compartilhados (indicador relacionado à estrutura de fusão, e não a um índice isolado de "similaridade" entre casos).

A Figura 1 sintetiza o fluxo arquitetural completo, conectando os módulos de entrada, processamento e geração de métricas para contextualizar essa visualização no pipeline de ponta a ponta.

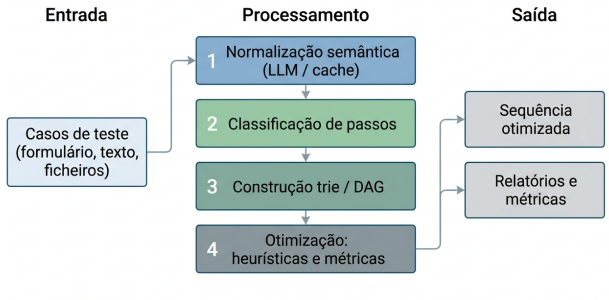

Figura 1: Visão geral: fluxo entre entrada da suíte, processamento do pipeline e saída da recomendação com métricas.

3.2 Modelagem de Dependências Entre Casos de Teste

Para explorar a reutilização de passos e os efeitos das ações sobre o estado durante a execução manual, modelamos a suíte como uma estrutura de prefixos compartilhados sobre os passos normalizados dos casos de teste (Figura 2): em primeiro lugar uma árvore de prefixos (trie), que, após o *merge* de sufixos equivalentes, pode ser vista como um grafo direcionado acíclico (DAG) cujos vértices correspondem a passos (identificados após normalização), e não a casos de teste inteiros. Cada caso de teste é representado por um caminho nessa estrutura, de modo que prefixos comuns entre casos aparecem como nós compartilhados, reduzindo redundância sem precisar declarar explicitamente arestas do tipo "teste i habilita teste j". Os passos são ainda classificados (por exemplo, verificação versus ação e destrutividade em relação ao contexto), o que informa a ordenação dos ramos e a necessidade de reexecução de prefixos (re-setup) quando um ramo quebra o contexto estabelecido pelos ancestrais. Antes da construção dessa estrutura, os casos podem ser reordenados de modo a maximizar o compartilhamento de prefixos, favorecendo sequências com menos repetição de passos. A partir dessa *trie*/DAG, o sistema gera uma ordem linear de execução (por exemplo, via busca em profundidade com critérios de prioridade), marca em quais pontos cada caso é validado e calcula métricas associadas à redução de passos, a trocas de contexto e aos re-setups, aproximando o objetivo de menos reinicializações do ambiente e menos redundância.

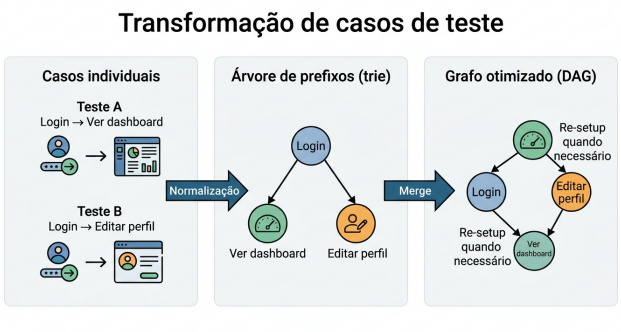

Figura 2: Modelagem de dependências por prefixos compartilhados: de casos individuais para *trie*/DAG sobre passos normalizados. O ramo destacado mostra como prefixos comuns são consolidados em um único caminho na *trie*/DAG.

3.3 Modelagem de Estado

A análise do impacto dos passos sobre o contexto de execução constitui um dos eixos do sistema. Não se mantém, nesta versão, um modelo explícito do estado interno do sistema sob teste (por exemplo, variáveis ou autômato do produto); em vez disso, cada passo descrito nos casos de teste é classificado quanto ao papel na sequência e quanto ao efeito sobre o contexto que se pretende preservar entre passos.

Os passos são tratados, em linhas gerais, em dois eixos complementares (Figura 3):

(1) Tipo de passo: distingue-se verificação — checagens que, na classificação adotada, não são tratadas como destrutivas ao contexto — de ação, que pode alterar fluxo, dados ou configuração percebida pelo testador.
(2) Destrutividade em relação ao contexto: para ações (e na ordenação dos ramos), identifica-se se o passo altera ou encerra de forma relevante o contexto acumulado (por exemplo, encerramento de fluxo, remoção, reset), o que influencia a ordem relativa dos ramos e a necessidade de reexecução de prefixos (re-setup) após certos trechos. Assim, o "estado" é abstraído pelos rótulos associados aos passos e pelas regras do otimizador, e não por um simulador do SUT.

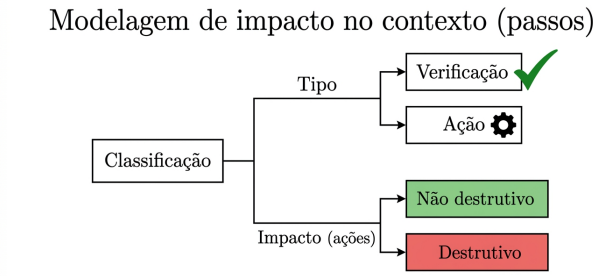

Figura 3: Modelagem de estado por classificação de passos (tipo e impacto no contexto).

Essa abordagem aproxima-se da intuição de ações não destrutivas (verificações e ações que não invalidam o contexto para os efeitos do modelo) versus ações destrutivas ou fortemente transformadoras (que exigem postergação ou recuperação de contexto na sequência sugerida).

3.4 Mecanismo de Recomendação

O módulo de recomendação integra normalização de passos, classificação, reordenação dos casos para favorecer prefixos comuns, construção de uma *trie* (com *merge* que pode produzir um DAG sobre passos normalizados) e geração de uma sequência linear sugerida — tipicamente por percurso em profundidade com critérios de prioridade entre ramos irmãos.

A recomendação prioriza, quando a estrutura permite, verificações e ações que não são tratadas como destrutivas ao contexto antes de ramos destrutivos ou que quebram o contexto estabelecido pelos ancestrais, postergando alterações que forçam re-setups ou encerram fluxos compartilhados. O objetivo é reduzir redundância de passos (por fusão e reuso de prefixos) e organizar a ordem de forma coerente com esses rótulos de impacto, e não com um estado em tempo real do sistema sob teste informado a cada passo.

No protótipo implementado, a ordenação é calculada de forma batch a cada submissão da suíte: não há, nesta versão, replanejamento contínuo ligado à execução manual passo a passo (atualização dinâmica conforme o testador avança na bancada). O caráter não totalmente estático entre sessões advém da reutilização de normalizações e classificações armazenadas (base de conhecimento), que enxugam trabalho repetido em execuções posteriores.

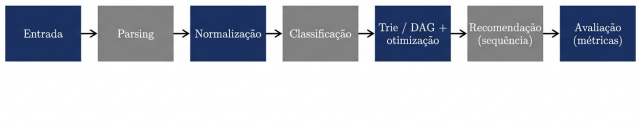

Figura 4: Pipeline do mecanismo de recomendação: normalização, classificação, construção trie/DAG, otimização e cálculo de métricas.

A complexidade computacional do otimizador é dominada pela construção da *trie*/DAG, que opera em $O(N)$ no número total de passos da suíte, e pela busca em profundidade com critérios de prioridade, em $O(V + E)$ sobre os vértices e arestas da estrutura consolidada. Em termos práticos, o pipeline mantém tempo de execução na ordem de centenas a poucas unidades de milhares de milissegundos para suítes com até 50 casos e 456 passos no provedor local (ver Seção 5).

Na classificação, verificações são sempre não destrutivas; ações que alteram ou encerram fluxos (criar, remover, encerrar telas) são marcadas como destrutivas. Na DFS, ramos irmãos são ordenados por: (i) verificações; (ii) ações não destrutivas; (iii) ações destrutivas; (iv) ramos que exigem re-setup. A fusão ocorre apenas entre passos com o mesmo `normalized_id`. Como ilustrado na Figura 4, o pipeline integra normalização, classificação, construção da *trie*/DAG e geração da sequência linear com pontos de validação por caso.

3.5 Limitações e Trade-off com Detecção de Falhas

A consolidação assume equivalência operacional entre passos normalizados: passos fundidos devem ser intercambiáveis do ponto de vista do contexto relevante à sessão. Quando essa premissa falha — por exemplo, se dois textos paráfrasticos correspondem a ações distintas no sistema sob teste — a sequência sugerida pode omitir transições relevantes à detecção de falhas. Da mesma forma, re-setups estruturais reduzem repetição na sequência linear, mas não garantem equivalência comportamental completa com a ordem documentada de cada caso.

Por isso, o sistema opera como recomendação HITL: o testador executa, interpreta e pode divergir da ordem sugerida. A avaliação reporta redução estrutural e custo computacional do pipeline, não eficácia em detecção de falhas. Trabalhos futuros incluem mutantes, falhas semeadas e revisão humana de fusões propostas pelo normalizador.

3.6 Exemplo Ilustrativo

Para tornar concreta a abordagem, consideramos três casos de uma suíte fictícia de gestão de alunos (`TEST-L`, `TEST-R`, `TEST-C`), com 17 passos no total na ordem original L→R→C. A Figura 5 ilustra o efeito qualitativo; este exemplo detalha o processamento.

Entrada. Os casos compartilham o prefixo "Abrir sistema → Login → Navegar até Alunos". `TEST-L` verifica a listagem; `TEST-R` remove um aluno; `TEST-C` cadastra um aluno e verifica a inclusão. Executados separadamente na ordem L→R→C, os três primeiros passos repetem-se integralmente em cada caso (17 passos no total). Além da redundância, ordens inadequadas (por exemplo, *Remover* antes de *Listar*) podem invalidar verificações posteriores.

Processamento. Após normalização, os passos 1–3 mapeiam para `N01_abrir`, `N02_login` e `N03_nav_alunos`. Passos subsequentes recebem identificadores distintos. Na classificação, verificações são não destrutivas; *Remover* e *Confirmar* são destrutivos. A reordenação *greedy* favorece `TEST-C` (caso mais longo) e prefixos comuns, resultando na ordem C→R→L. A trie ramifica após `N03`; a DFS posterga o ramo destrutivo e insere re-setup antes de validar `TEST-L`.

Sequência sugerida (14 passos).

(1) Abrir o sistema
(2) Fazer login com usuário de teste
(3) Navegar até a tela Alunos
(4) Clicar em Novo aluno
(5) Preencher o nome "Maria Teste"
(6) Salvar o cadastro
(7) Verificar que o aluno aparece na lista [`TEST-C`]
(8) Selecionar o aluno "João Teste"
(9) Clicar em Remover
(10) Confirmar a remoção [`TEST-R`]
(11) Abrir o sistema [re-setup]
(12) Fazer login com usuário de teste [re-setup]
(13) Navegar até a tela Alunos [re-setup]
(14) Verificar que a lista de alunos é exibida [`TEST-L`]

O prefixo comum executa-se uma única vez no início (elimina-se duas repetições). Como a remoção altera o contexto da lista, o pipeline insere re-setup (passos 11–13) antes da verificação de `TEST-L`. Métricas estruturais: 17 passos originais, 14 na sequência sugerida (redução de 17,6%), `resetup_count=1`, `merge_depth_max=3`. Ressalva: métricas do pipeline; tempo humano não é medido neste estudo (Seção 6).

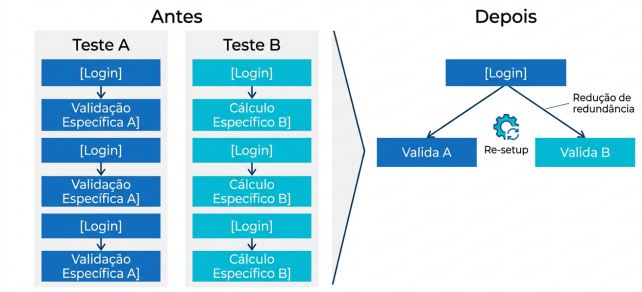

Figura 5: Ilustração didática do efeito de otimização na sequência de execução (antes/depois), com consolidação de passos comuns e indicação de validações/re-setup.

4 Materiais e Métodos

Esta pesquisa adota uma abordagem de Pesquisa em Ciência do Projeto (Design Science Research — DSR), na qual o conhecimento é produzido por meio da construção e da avaliação de um artefato destinado a resolver uma classe de problemas identificada na literatura e na prática de teste manual [15].

O ciclo metodológico organiza-se em seis momentos articulados: (i) identificação do problema; (ii) definição de objetivos de solução; (iii) concepção e desenvolvimento do artefato; (iv) demonstração; (v) avaliação; (vi) comunicação dos resultados.

Essa estrutura é compatível com a indisponibilidade de validação organizacional prolongada por parte do cliente final: a avaliação privilegia demonstração funcional, métricas objetivas derivadas do próprio artefato e análise de cenários com suítes reutilizáveis, explicitando-se as limitações de generalização para outros contextos industriais.

4.1 Identificação do Problema

O problema de pesquisa decorre da tensão entre a relevância dos testes manuais (julgamento humano, contexto, custo cognitivo) e a ineficiência frequentemente observada na execução de suítes extensas: sobreposição de passos, alternância de contexto, repreparações e ordens de execução pouco favoráveis quando comparadas às estratégias adotadas por testadores experientes. A literatura de priorização e regressão automatizada cobre parcialmente esse espaço, porém com premissas frequentemente distantes da dinâmica manual. Dessa forma, delimita-se a necessidade de um sistema de apoio à decisão (human-in-the-loop) que sugira uma sequência consolidada de passos, com transparência quanto aos fatores considerados, sem substituir o testador na execução física nem exigir, nesta fase, um modelo completo de estados do sistema sob teste no estilo de teste baseado em modelos.

4.2 Objetivos de Solução

Definiram-se objetivos de solução que orientam o desenho do artefato:

(1) Entrada flexível da suíte (formulário, texto livre, arquivos JSON, CSV ou planilhas), compatível com práticas reais de registro de casos.
(2) Normalização semântica de passos com cache persistente (base de conhecimento), reduzindo custo repetido de inferência e permitindo evolução incremental entre execuções.
(3) Classificação de passos quanto ao tipo (verificação versus ação) e ao impacto no contexto (destrutividade binária, complementada por heurísticas de quebra de contexto na ordenação).
(4) Estruturação da suíte como árvore de prefixos sobre passos normalizados, com *merge* de sufixos que pode resultar em um DAG, explorando reuso de prefixos entre casos.
(5) Geração de uma sequência linear sugerida com marcação de validação por caso e métricas explicativas (redução de passos, re-setups, trocas de contexto, profundidade de *merge*, entre outras).
(6) Transparência e reprodutibilidade acadêmica: exportação de resultados, histórico de execuções em banco de dados e possibilidade de comparar provedores de inferência *(benchmark)*.

4.3 Concepção e Desenvolvimento

A concepção traduziu os objetivos em uma arquitetura em camadas, implementada como aplicação web em Python com Flask, persistência em SQLite e interface em templates e recursos estáticos. O núcleo algorítmico concentra-se no módulo de engenharia, que executa, em sequência: normalização; classificação; reordenação dos casos para maximizar prefixos comuns; construção da *trie* e *merge* de sufixos; otimização da ordem de visitação (DFS com critérios de prioridade); construção do resultado com pontos de validação; serialização da árvore para visualização; cálculo de métricas.

A inferência sobre o texto dos passos é abstraída por uma fábrica de provedores, composta por um modo local (heurístico, sem dependência de API externa) e pelo provedor da OpenAI, ambos selecionáveis por configuração. O cache de normalização e classificação utiliza tabelas dedicadas em SQLite, constituindo uma base de conhecimento reutilizável entre sessões.

O desenvolvimento seguiu iterações incrementais sobre esse pipeline, com scripts auxiliares para execução fora da interface e inspeção de saídas em console, e suítes de exemplo para regressão informal do comportamento.

4.3.1 Ambiente e Protocolo Experimental. Os ensaios foram executados em computador com processador AMD Ryzen 5 PRO, 16 GB de RAM e Windows 11, com Python 3.13.3 e commit `94ab775` do repositório: `marcelosaraiva00/IARTES-PROJETO`. Foram avaliados os modos `local` (heurístico) e `openai` (`gpt-4o-mini`). Na interface web, o pipeline usou `use_cache=true`; no *benchmark* comparativo, `use_cache=false` para isolar o custo de inferência. Para cada par (suíte, provedor), registraram-se `elapsed_ms`, o JSON completo da resposta e, no *benchmark*, três repetições por configuração (média±desvio padrão). As suítes utilizadas estão resumidas na Tabela 2.

A demonstração ocorre pela interface web e por scripts reprodutíveis sobre as suítes documentadas (Seção 5).

4.4 Avaliação

Dada a restrição de tempo para estudos de campo formais com a organização usuária, a avaliação combina critérios objetivos produzidos pelo próprio sistema com análise qualitativa reflexiva dos cenários, explicitando ameaças à validade.

Dimensão quantitativa (interna ao artefato):

- Eficácia de consolidação: redução do número de passos na sequência sugerida versus soma dos passos originais; percentual de redução.
- Estrutura e contexto: contagem de passos destrutivos, re-setups, trocas de contexto e indicadores de profundidade de *merge*, permitindo comparar configurações e suítes.
- Eficiência operacional do pipeline: tempo de execução (milissegundos) registrado nas execuções e, quando aplicável, comparação entre provedores no *benchmark*.

Dimensão comparativa:

- *Benchmark* entre provedores disponíveis (local e remotos com chave configurada), sobre as mesmas entradas, com tabela-resumo de métricas.

Dimensão qualitativa:

- Inspeção da sequência e da árvore por parte dos autores (e, se viável, de um número reduzido de especialistas sem compromisso formal de cliente), focando em coerência da ordem, plausibilidade das fusões e interpretabilidade das métricas.

Ameaças à validade (a declarar na discussão): dependência de qualidade da normalização/classificação (especialmente com LLM); representatividade das suítes usadas; ausência de medição de tempo humano real na execução física; generalização limitada sem estudo longitudinal na indústria.

O desenho de avaliação articula QP1 com `reduction_percent`, `resetup_count` e `merge_depth_max`; QP2 com as mesmas métricas estruturais e `elapsed_ms` na comparação entre provedores.

5 Resultados e Discussão

Este capítulo apresenta os resultados obtidos com o protótipo, organizados para permitir avaliação técnica e interpretação do comportamento do sistema. A configuração experimental e as suítes de entrada são resumidas na Seção 5.1, com detalhes do protocolo na Seção 4.3.1. Em seguida, reportam-se os indicadores quantitativos do pipeline — redução relativa de passos, re-setups, trocas de contexto, profundidade de merge e tempos de execução — para o provedor local (Seção 5.2) e no *benchmark* comparativo entre provedores (Seção 5.3). Fecha-se com síntese qualitativa e discussão dos achados, explicitando limitações e ameaças à validade.

5.1 Configuração da Avaliação

A configuração experimental (hardware, versões, provedores, cache e protocolo de repetições) está detalhada na Seção 4.3.1. A Tabela 2 resume as suítes S1–S3. As métricas reportadas seguem o glossário da Tabela 3.

Tabela 2: Suítes de teste utilizadas na avaliação.

ID	Arquivo	Casos	Passos	Formato
S1	`suite_1_portugues.txt`	10	77	Texto livre
S2	`suite_2_portugues.txt`	20	136	Texto livre
S3	`suite_3_portugues.txt`	50	456	Texto livre

Suítes localizadas no diretório `test_suites/` do repositório do projeto.

Tabela 3: Glossário das métricas coletadas no benchmark.

Métrica	Descrição
`test_count`	Número de casos de teste na suíte.
`total_original_steps`	Total de passos antes da otimização.
`total_optimized_steps`	Total de passos na sequência consolidada.
`steps_eliminated`	Diferença entre passos originais e otimizados.
`reduction_percent`	Percentual de redução estrutural de passos.
`verifications_detected`	Passos classificados como verificação.
`actions_detected`	Passos classificados como ação.
`destructive_steps`	Passos que alteram o contexto de execução.
`resetup_count`	Reexecuções de prefixo após quebra de contexto.
`context_switches`	Passos destrutivos na sequência (definição operacional).
`merge_depth_max`	Profundidade máxima de consolidação na trie/DAG.
`merge_depth_avg`	Profundidade média de consolidação na trie/DAG.

5.2 Resultado Quantitativo por Suíte (Provedor Local)

A Tabela 4 sintetiza os resultados obtidos com o provedor principal adotado neste estudo (local), permitindo comparar o efeito da otimização em diferentes suítes e observar, em conjunto, indicadores de redução, reconfiguração de contexto e profundidade de consolidação estrutural.

Tabela 4: Resultados principais por suíte com provedor `local`.

Suíte	Casos	Orig.	Otim.	Red.%	Elim.
S1	10	77	75	2,6	2
S2	20	136	124	8,8	12
S3	50	456	366	19,7	90

Legenda: Orig.=`total_original_steps`; Otim.=`total_optimized_steps`; Red.%=`reduction_percent`; Elim.=`steps_eliminated`.

Como mostrado na Tabela 4, a redução de passos cresce com o aumento da suíte, saindo de 2,6% em S1 para 19,7% em S3, com eliminação absoluta de 90 passos no maior cenário. Os tempos do provedor `local` permanecem na ordem de milissegundos (Tabela 7), o que indica viabilidade prática para uso iterativo durante a preparação da execução manual.

Para complementar a análise de eficiência, a Tabela 5 apresenta métricas estruturais e de contexto associadas ao comportamento do otimizador.

Tabela 5: Métricas estruturais e de contexto por suíte (provedor `local`).

Suíte	Re-setup	CtxSw	Destr.	MMax	MAvg
S1	2	22	22	1	0,16
S2	5	39	39	2	0,72
S3	11	50	50	7	0,78

Legenda: Re-setup=`resetup_count`; CtxSw=`context_switches`; Destr.=`destructive_steps`; MMax=`merge_depth_max`; MAvg=`merge_depth_avg`.

Observa-se que o crescimento de resetup_count e merge_depth_max em S3 é coerente com a maior complexidade estrutural da suíte e com a presença de mais ramos compartilhados na *trie*/DAG. As métricas context_switches e destructive_steps evoluem em conjunto nos três cenários, reforçando que o custo de transição na sequência otimizada está fortemente associado à quantidade de passos classificados como destrutivos. Esses resultados sustentam que o mecanismo reduz redundância sem ignorar o custo de manutenção de contexto entre ramos.

Com base nos resultados do provedor `local` nas três suítes (Tabelas 4 e 5), a síntese abaixo responde à QP1.

Tabela 6: Resposta sintética à QP1 (redução estrutural, provedor local).

QP1: O sistema reduz redundância estrutural entre 2,6% (S1) e 19,7% (S3), com `resetup_count`> 0 em todas as suítes e crescimento coerente de `merge_depth_max`, indicando que a consolidação escala com a sobreposição de prefixos sem ignorar o custo de reconfiguração de contexto.

5.3 Resultados Comparativos Entre Provedor Local x Openai (Benchmark)

Para comparar o comportamento semântico e o custo computacional entre modos de inferência, executou-se o *benchmark* com os provedores `local` e `openai` nas suítes S1, S2 e S3 (suite_1_portugues.txt, suite_2_portugues.txt e suite_3_portugues.txt), com três repetições independentes por par (suíte, provedor). A Tabela 7 sintetiza as métricas principais e o tempo de execução (média ± desvio padrão).

Tabela 7: Benchmark com 3 repetições por suíte e provedor.

Suíte	Prov.	Red.%	Otim.	Re-set.	CtxSw	Tempo (ms)
S1	local	2,6	75	2	22,0	470 ± 129
S1	openai	2,6	75	2	25,0	127.953 ± 7.884
S2	local	8,8	124	5	39,0	578 ± 36
S2	openai	8,1	125	5	42,7	186.935 ± 8.282
S3	local	19,7	366	11	50,0	1.626 ± 224
S3	openai	19,7	366	11	103,7	534.711 ± 40.853

Tempo reportado como média ± desvio padrão em 3 repetições.

Como mostrado na Tabela 7, o provedor `local` apresentou tempos consistentemente inferiores em todas as suítes, enquanto o `openai` apresentou maior custo computacional e variabilidade temporal. Em termos estruturais, S1 e S3 mostraram convergência entre provedores em `reduction_percent` e `total_optimized_steps`, enquanto em S2 houve pequena diferença (8,8% vs 8,1%; 124 vs 125 passos). As maiores diferenças semânticas aparecem em context_switches, especialmente em S3, sugerindo sensibilidade da classificação de destrutividade ao provedor de inferência.

Tabela 8: Diferenças entre provedores por suíte (Δ = openai − local).

Suíte	Δ Red.%	Δ Otim.	Δ CtxSw	Δ Tempo (ms)
S1	0,0	0	+3,0	+127.482,6
S2	-0,7	+1	+3,7	+186.357,0
S3	0,0	0	+53,7	+533.084,9

Para o protocolo adotado (`use_cache=false`, três repetições por par suíte–provedor), o modo heurístico `local` é a escolha recomendada: entrega sequências estruturalmente equivalentes ou muito próximas às do `openai` nas métricas principais (`reduction_percent`, `total_optimized_steps`), com custo computacional entre 272× e 329× inferior (Tabela 7), sem ganho estrutural consistente do LLM remoto neste cenário. Essa conclusão refere-se a métricas estruturais do pipeline; estudo com testadores em contexto real permanece como principal trabalho futuro (Seção 7). Em conjunto, os resultados indicam que a abordagem com OpenAI mantém desempenho estrutural semelhante ao *baseline* local em vários cenários, porém com custo computacional substancialmente maior e maior sensibilidade semântica em métricas de contexto. Os resultados do *benchmark* com três repetições por configuração (Tabelas 7 e 8) permitem responder à QP2: em S1 e S3, os provedores convergem em `reduction_percent` e `total_optimized_steps`; em S2 a divergência é modesta (8,8% vs. 8,1%; 124 vs. 125 passos). As maiores diferenças aparecem em `context_switches` (S3: Δ = +53,7) e em `elapsed_ms` (fator 272×–329×).

Tabela 9: Resposta sintética à QP2 (comparação local × OpenAI).

QP2: Em S1 e S3, os provedores convergem em `reduction_percent` e `total_optimized_steps`; em S2 a divergência é modesta (8,8% vs. 8,1%; 124 vs. 125 passos). As maiores diferenças aparecem em `context_switches` (S3: Δ = +53,7) e em `elapsed_ms` (fator 272×–329×).

5.4 Síntese Qualitativa Complementar

Selecionou-se manualmente um excerto da sequência otimizada da suíte S2 (`suite_2_portugues.txt`), totalizando 125 passos após a otimização (a partir de 136 originais), com base nos critérios: (i) suíte de tamanho intermediário; (ii) fusão visível de prefixos no início da sequência; (iii) presença de passos limítrofes na classificação destrutiva (por exemplo, conectar cabo USB). Analisa-se aqui um trecho inicial de 14 passos; o trecho completo permanece no material suplementar. Observou-se uma fusão plausível de ações de preparação e navegação antes da validação de casos específicos (por exemplo, um bloco contínuo no navegador culminando na validação de TEST-006), o que é compatível com o objetivo de reduzir repetição de prefixos. Como ilustrado na Figura 5, a consolidação preserva pontos de validação por caso e pode inserir re-setups após ramos destrutivos.

5.5 Discussão

Os resultados indicam que o artefato cumpre parcialmente o objetivo de reduzir redundância na execução manual, com reduções entre 2,6% e 19,7% nas suítes testadas (suite_1_portugues.txt, suite_2_portugues.txt, suite_3_portugues.txt). A presença de resetup_count > 0 em todas as suítes (S1=2, S2=5, S3=11) evidencia que a otimização não elimina completamente a necessidade de reexecução de prefixos quando há quebra de contexto. Esse comportamento é coerente com o desenho do otimizador e com a natureza conservadora da sequência sugerida, que prioriza preservação de contexto sem assumir equivalência semântica perfeita entre todos os passos.

Quanto à coerência com o mecanismo de recomendação, a inspeção dos resultados mostra padrão compatível com a política de priorização da DFS: manutenção de estrutura consolidada com postergação relativa de ramos mais custosos ao contexto, sem comprometer a cardinalidade final da sequência em cenários comparáveis. No *benchmark* da suíte S2, por exemplo, os provedores convergiram em total_optimized_steps e resetup_count, com divergência modesta em reduction_percent (8,8% vs. 8,1%), e também em métricas derivadas da classificação (destructive_steps e context_switches), indicando sensibilidade do sistema à etapa semântica de normalização/classificação. Essa sensibilidade tende a aumentar em entradas com passos curtos, ambíguos ou formulações heterogêneas.

Em relação à literatura, o trabalho aqui apresentado posiciona-se de forma distinta da priorização clássica para regressão automatizada por privilegiar consolidação em nível de passo e apoio human-in-the-loop à decisão de execução. Como contrapartida, a abordagem depende da qualidade de heurísticas e/ou LLM para normalização e classificação, e não implementa um modelo explícito de estado interno do SUT equivalente a autômatos de teste baseado em modelos. Assim, os resultados devem ser interpretados como evidência de ganho estrutural na sequência de execução, e não como prova formal de cobertura de transições de estado do sistema.

O trabalho distingue-se da priorização clássica de casos (TSP) e da redução de suítes (TSR) por operar em *nível de passo* e por manter o testador como executor e decisor final (HITL), em vez de otimizar detecção precoce de falhas em regressão automatizada. A avaliação reporta redução estrutural e custo computacional do pipeline, não eficácia em detecção de falhas.

Por fim, não foi conduzido estudo longitudinal com o cliente final no ambiente organizacional alvo. Portanto, as conclusões sobre esforço cognitivo permanecem indiretas, fundamentadas em métricas estruturais do protótipo e em inspeção técnica dos resultados, sem validação observacional contínua com usuários em operação real. Essa limitação não invalida os ganhos reportados, mas delimita a generalização dos achados e reforça a necessidade de estudos futuros com testadores em contexto de uso real.

6 Ameaças à Validade

Esta seção apresenta as principais ameaças à validade do estudo, organizadas segundo a classificação clássica: validade interna, validade externa, validade de construção e validade de conclusão.

6.1 Validade Interna

A validade interna pode ser afetada por fatores de implementação e execução experimental. O comportamento do sistema depende da configuração do ambiente (versão de runtime, bibliotecas e parâmetros de timeout), além da disponibilidade dos provedores remotos durante os ensaios. Em especial, variações de latência e indisponibilidade temporária de API podem impactar tempos de execução e, em alguns cenários, impedir o retorno completo do *benchmark*.

Outra ameaça interna decorre da etapa semântica de normalização/classificação: pequenas variações na resposta dos modelos podem alterar métricas como destructive_steps e context_switches, mesmo quando a redução estrutural de passos permanece estável. Para mitigar esse risco, os experimentos foram executados com protocolo fixo por suíte e provedor, mantendo rastreabilidade dos resultados obtidos.

6.2 Validade Externa

A generalização dos resultados é limitada pelo conjunto de dados utilizado e pela ausência de validação longitudinal em ambiente organizacional real. As suítes avaliadas representam cenários manuais plausíveis, mas não cobrem toda a diversidade de domínios, ferramentas e políticas de teste encontradas na indústria.

Assim, os achados não devem ser extrapolados automaticamente para contextos com forte regulação, requisitos de segurança crítica ou fluxos de teste altamente especializados sem estudos adicionais de transferência.

6.3 Validade de Construção

As métricas adotadas representam construtos operacionais do protótipo e não medidas diretas de desempenho humano. Em particular, reduction_percent quantifica redução de passos na sequência sugerida, mas não mede diretamente tempo humano de execução, taxa de erro do testador ou carga cognitiva percebida.

De forma semelhante, context_switches segue uma definição operacional do algoritmo (associada a passos destrutivos no fluxo otimizado), não equivalendo, por si só, a um indicador psicométrico de esforço mental. Portanto, a interpretação dos resultados deve respeitar o nível de abstração dessas métricas.

6.4 Validade de Conclusão

A validade de conclusão é influenciada pelo número de suítes e pelo desenho de comparação adotado. Embora os resultados mostrem tendências consistentes de redução de redundância e diferenças semânticas entre provedores, a base empírica ainda é limitada para inferências estatísticas fortes sobre superioridade global de um provedor em todos os cenários.

7 Conclusão e Trabalhos Futuros

Este trabalho apresentou um artefato para apoio à execução manual de suítes de teste por meio de normalização semântica de passos, classificação de impacto no contexto e geração de sequência otimizada com base em estruturas de prefixos compartilhados (*trie*/DAG). Os resultados indicam viabilidade técnica da abordagem, com redução de redundância entre suítes e capacidade de produzir métricas explicativas para suporte à decisão do testador.

No cenário avaliado, observou-se redução estrutural consistente na sequência sugerida, ainda que com variação entre suítes conforme o grau de sobreposição estrutural. Os resultados referem-se a passos na sequência consolidada, não a tempo humano de execução nem a taxa de detecção de falhas.

As limitações discutidas na seção anterior orientam diretamente a agenda de evolução do trabalho. Como trabalhos futuros, destacam-se:

(1) **Validação com usuários reais em contexto organizacional:** medir tempo de execução manual, retrabalho e utilidade percebida da recomendação;
(2) **Ampliação de suítes e domínios, com métricas de esforço humano:** avaliar o artefato em cenários críticos, multilíngues e com maior diversidade de estilos de escrita;
(3) **Robustez da inferência:** fallback entre provedores, controle de timeout e ablação por etapa do pipeline;
(4) **Evolução algorítmica:** replanejamento incremental durante a execução manual e estudos de ablação (por exemplo, sem reordenação ou sem *merge*).

Em síntese, este projeto apresenta evidências iniciais de utilidade como ferramenta de apoio à execução manual de testes, com resultados promissores em redução de redundância e explicabilidade da recomendação. A consolidação dessa contribuição depende, como próximo passo, de validação empírica ampliada com usuários e ambientes reais de teste.

Disponibilidade de Artefatos

Os artefatos estão disponíveis em https://github.com/marcelosaraiva00/IARTES-PROJETO (commit `94ab775`) e em https://zenodo.org/record/19699079.

Agradecimentos

Foi utilizada a ferramenta de IA generativa Composer apenas como apoio à edição de todas as figuras ilustrativas. Os textos, análises e resultados são de autoria exclusiva dos autores.

Referências

[1] Paul Ammann and Jeff Offutt. 2017. *Introduction to software testing.* Cambridge University Press.

[2] Stuart K Card. 2018. *The psychology of human-computer interaction.* Crc Press.

[3] Sebastian Elbaum, Alexey G Malishevsky, and Gregg Rothermel. 2002. Test case prioritization: A family of empirical studies. *IEEE transactions on software engineering* 28, 2 (2002), 159–182.

[4] Daniel Flemström, Pasqualina Potena, Daniel Sundmark, Wasif Afzal, and Markus Bohlin. 2018. Similarity-based prioritization of test case automation. *Software quality journal* 26, 4 (2018), 1421–1449.

[5] Kamal Garg and Shashi Shekhar. 2024. Test case prioritization based on fault sensitivity analysis using ranked NSGA-2. *International Journal of Information Technology* 16, 5 (2024), 2875–2881.

[6] Vahid Garousi, Michael Felderer, and Mika V. Mäntylä. 2019. Guidelines for Including Grey Literature and Conducting Multivocal Literature Reviews in Software Engineering. *Information and Software Technology* 106 (2019), 101–121.

[7] Jeff Johnson. 2020. *Designing with the mind in mind: simple guide to understanding user interface design guidelines.* Morgan Kaufmann.

[8] Cem Kaner, James Bach, and Bret Pettichord. 2011. *Lessons learned in software testing: a context-driven approach.* John Wiley & Sons.

[9] Md Asif Khan, Akramul Azim, Ramiro Liscano, Kevin Smith, Yee-Kang Chang, Qasim Tauseef, and Gkerta Seferi. 2024. Machine learning-based test case prioritization using hyperparameter optimization. In *Proceedings of the 5th ACM/IEEE International Conference on Automation of Software Test (AST 2024).* 125–135.

[10] Jeffrey V Latina, Glaidelyn M Cabalsi, Joervy R Sanchez, Elnard Don M Vallejo, Criselle J Centeno, and Eufemia A Garcia. 2024. Utilization of NLP Techniques in Plagiarism Detection System through Semantic Analysis using Word2Vec and BERT. In *2024 International Conference on Expert Clouds and Applications (ICOECA).* IEEE, 347–352.

[11] Yinghua Li, Xueqi Dang, Lei Ma, Jacques Klein, and Tegawendé F. Bissyandé. 2024. Prioritizing test cases for deep learning-based video classifiers. *Empirical Software Engineering* 29, 5 (2024), 111. doi:10.1007/s10664-024-10520-1

[12] Glenford J Myers, Tom Badgett, Todd M Thomas, and Corey Sandler. 2004. *The art of software testing.* Vol. 2. Wiley Online Library.

[13] Glenford J. Myers, Corey Sandler, and Tom Badgett. 2011. *The Art of Software Testing* (3 ed.). John Wiley & Sons, Hoboken.

[14] Dianeliz Ortiz Martes, Evan Gunderson, Caitlin Neuman, and Nezamoddin N. Kachouie. 2025. Transformer Models for Paraphrase Detection: A Comprehensive Semantic Similarity Study. *Computers* 14, 9 (2025), 385. doi:10.3390/computers14090385

[15] Ken Peffers, Tuure Tuunanen, Marcus A Rothenberger, and Samir Chatterjee. 2007. A design science research methodology for information systems research. *Journal of management information systems* 24, 3 (2007), 45–77.

[16] Roger S Pressman. 2005. *Software engineering: a practitioner's approach.* Palgrave macmillan.

[17] Roger S. Pressman and Bruce R. Maxim. 2014. *Software Engineering: A Practitioner's Approach* (8 ed.). McGraw-Hill Education, New York.

[18] Gajender Rao and Deepak Nandal. 2026. Dynamic prioritization of test cases for regression testing using machine learning. *International Journal of System Assurance Engineering and Management* (2026), 1–19.

[19] Gregg Rothermel, Roland H. Untch, Chengyun Chu, and Mary Jean Harrold. 2001. Prioritizing test cases for regression testing. *IEEE Transactions on software engineering* 27, 10 (2001), 929–948.

[20] Jeffrey Rubin and Dana Chisnell. 2008. *Handbook of usability testing: How to plan, design, and conduct effective tests.* John Wiley & Sons.

[21] Mark Utting and Bruno Legeard. 2010. *Practical model-based testing: a tools approach.* Elsevier.

[22] Shin Yoo and Mark Harman. 2012. Regression testing minimization, selection and prioritization: a survey. *Software testing, verification and reliability* 22, 2 (2012), 67–120.