UNIVERSIDADE ESTADUAL DO CEARÁ

CENTRO DE CIÊNCIAS E TECNOLOGIA

CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

JONNAS CHRISTIAN SOUSA DE PAIVA

BIBLIOTECA DE SELF-HEALING PARA TESTES AUTOMATIZADOS DE INTERFACES WEB

FORTALEZA – CEARÁ

2025

SUMÁRIO

1 INTRODUÇÃO

A prática de testes de software é fundamental para assegurar a qualidade e a confiabilidade de sistemas modernos. Garantir que uma aplicação funcione corretamente e atenda aos requisitos especificados é um objetivo central em qualquer projeto de software, tornando a qualidade uma característica inegociável. Para alcançar essa validação de forma eficiente, especialmente em contextos nos quais novas funcionalidades são frequentemente integradas, torna-se indispensável o uso de testes automatizados, que possibilitam a detecção sistemática de falhas, aumentam a confiança no sistema e contribuem para a robustez do produto final (MYERS; SANDLER; BADGETT, 2012).

Um dos principais desafios enfrentados pelas equipes que adotam testes automatizados é a fragilidade diante de mudanças frequentes nas interfaces web. Alterações aparentemente simples no *Document Object Model* (DOM)[1], como a troca de um `id`, uma `class` ou o reposicionamento de elementos, podem tornar seletores obsoletos, ocasionando falhas em testes que antes funcionavam corretamente. Segundo Khankhoje (2023), frameworks com suporte a *self-healing* surgem como uma solução promissora para os desafios de manutenção em testes automatizados, permitindo maior eficiência e qualidade mesmo diante de alterações frequentes nas interfaces web. Esses frameworks se baseiam em mecanismos capazes de identificar mudanças e adaptar automaticamente os testes, reduzindo o esforço manual necessário para mantê-los atualizados.

Diante disso, este trabalho propõe desenvolver uma biblioteca de *self-healing* voltada para testes automatizados de interfaces web, capaz de lidar com mudanças na estrutura do DOM e reduzir o esforço de manutenção. Enquanto soluções existentes geralmente impõem barreiras de uso por dependerem de configurações complexas ou por se limitarem a determinados frameworks, a proposta aqui apresentada busca oferecer uma alternativa mais simples e amplamente aplicável, com fácil integração a diferentes ferramentas.

1.1 MOTIVAÇÃO

A manutenção é frequentemente apontada como um dos principais desafios da automação de testes. À medida que a complexidade do software cresce, aumenta também a quantidade de scripts necessários para acompanhar essa evolução, o que sobrecarrega as equipes com tarefas de atualização e correção. Segundo Battina (2019), cerca de 40% do tempo dos

[1] DOM é uma representação em árvore da estrutura HTML de uma página web, permitindo que scripts acessem, naveguem e modifiquem seus elementos dinamicamente.

testadores é consumido apenas com manutenção, incluindo esforços para depurar e ajustar testes que falham devido a alterações no sistema.

Além dos desafios relacionados à manutenção constante, ferramentas existentes que propõem mecanismos de *self-healing* também apresentam limitações técnicas que comprometem sua adoção em cenários mais amplos. O Healenium[2], por exemplo, embora seja uma solução robusta e de código aberto, possui suporte restrito a determinadas linguagens e frameworks, não sendo compatível com outras ferramentas amplamente utilizadas, como o Cypress. Além disso, sua configuração pode ser complexa e pouco flexível (KHANKHOJE, 2023).

Algumas soluções modernas também dependem de técnicas de inteligência artificial (IA) para adaptação de seletores, o que pode tornar a implementação mais pesada, menos transparente e limitada a contextos específicos. Essas limitações evidenciam a importância de mecanismos de autoajuste que priorizem a simplicidade, a transparência e a flexibilidade, sendo de fácil integração e capazes de atender diferentes contextos e tecnologias de automação.

1.2 OBJETIVO GERAL

Desenvolver uma biblioteca em `Python` para *self-healing* de testes web automatizados, com configuração simples e integração flexível a múltiplos frameworks de automação.

1.3 OBJETIVOS ESPECÍFICOS

- Implementar a extração de elementos do DOM com suporte a diferentes tipos de seletores;
- Desenvolver o mecanismo de comparação entre versões do DOM;
- Integrar a biblioteca aos frameworks Selenium, Cypress e Robot Framework;
- Avaliar a eficácia da biblioteca por meio de validações técnicas e práticas.
- Disponibilizar a biblioteca para instalação via *Python Package Index* (PyPI)[3]

[2] <https://healenium.io/>

[3] PyPI é o repositório oficial de pacotes de software para a linguagem Python, permitindo que projetos sejam publicados e instalados facilmente pela comunidade. Disponível em: <https://pypi.org/>.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo aborda os conceitos de testes de software, automação de testes e mecanismos de *self-healing*, que fundamentam a proposta deste trabalho.

2.1 TESTES DE SOFTWARE

Testes de software são um conjunto de atividades realizadas com o propósito de avaliar a qualidade de um sistema e identificar possíveis defeitos em seus artefatos. Essas atividades envolvem tanto a verificação, que consiste em assegurar que o sistema atenda aos requisitos especificados, quanto a validação, que verifica se o produto final satisfaz as necessidades dos usuários e demais partes interessadas (ISTQB, 2023). Os testes também têm como objetivos detectar defeitos, avaliar produtos de trabalho, validar o funcionamento do sistema e fornecer informações que apoiem a tomada de decisões. Além disso, contribuem para aumentar a confiança na qualidade do produto e reduzir riscos relacionados à não conformidade com requisitos funcionais ou legais.

De acordo com a CTFL (ISTQB, 2023), os testes de software podem ser classificados em funcionais, que verificam se o sistema executa corretamente suas funcionalidades; não funcionais, que avaliam atributos que caracterizam qualidade; de caixa preta, que derivam testes a partir das especificações para verificar o comportamento do sistema; e de caixa branca, que se baseiam na estrutura interna do sistema, como código e fluxos de controle. Além dessas classificações, os testes também são organizados em diferentes níveis, de acordo com o escopo da verificação: teste de unidade, que verifica componentes isolados; teste de integração, que avalia a interação entre módulos ou sistemas; teste de sistema, que considera o sistema completo; e teste de aceitação, que valida se o produto atende aos critérios definidos pelo cliente.

2.2 AUTOMAÇÃO DE TESTES

A automação de testes consiste na utilização de ferramentas para executar, verificar e comparar os resultados de testes de software, com o objetivo de reduzir o esforço manual, aumentar a eficiência e garantir a repetibilidade no processo de verificação. Trata-se de uma prática estratégica que permite o reaproveitamento de testes, a detecção de regressões com maior agilidade e a padronização das execuções. Os artefatos utilizados e gerados nesse processo

compõem o chamado *testware*[1]. Embora traga benefícios em escala e consistência, a automação também impõe desafios quanto ao esforço inicial, à manutenção de scripts e à instabilidade provocada por alterações frequentes nas interfaces (FEWSTER; GRAHAM, 1999).

A norma ISO/IEC/IEEE (2022) ressalta que a automação deve ser aplicada com critério, considerando o contexto do projeto, os objetivos dos testes e os recursos disponíveis. Nem todos os tipos de teste são adequados para automação — especialmente aqueles que envolvem julgamento subjetivo ou são executados com pouca frequência. Fatores como a estabilidade da interface, a maturidade do sistema e a frequência de mudanças impactam diretamente na viabilidade da automação. Para que os benefícios sejam sustentáveis, é necessário considerar os custos de implementação e manutenção dos scripts, a capacitação da equipe e a escolha criteriosa das ferramentas. Como reforçado por Fewster e Graham (1999), uma abordagem mal planejada pode transformar o que seria uma otimização em um caos automatizado.

2.3 SELF-HEALING

O conceito de *self-healing* em automação de testes refere-se à capacidade de uma estrutura reagir de maneira autônoma a falhas decorrentes de mudanças inesperadas, geralmente causadas por alterações na estrutura da interface da aplicação (SAARATHY; BATHRACHALAM; RAJENDRAN, 2024). Essas falhas normalmente ocorrem devido a mudanças no DOM, como alterações em atributos, reorganização de elementos ou substituição de identificadores. Frameworks com suporte a *self-healing* monitoram a execução dos testes, detectam quando um seletor não localiza o elemento esperado e tentam, com base em regras ou comparações estruturais, encontrar alternativas viáveis para permitir a continuidade da execução com sucesso, evitando a falha prematura dos testes.

De acordo com Khankhoje (2023), soluções de *self-healing* geralmente operam a partir de uma arquitetura composta por módulos de monitoramento, detecção de falhas, reidentificação de elementos e atualização de scripts. Essas soluções podem empregar desde estratégias baseadas em IA até mecanismos mais simples, como análise de atributos do DOM, comparação de versões e aplicação de heurísticas estruturais. Independentemente da técnica utilizada, o objetivo é manter os testes mesmo com mudanças no DOM, reduzindo a necessidade de manutenção e aumentando a confiabilidade da automação.

[1] *Testware* é o conjunto de artefatos utilizados e produzidos durante os testes, como scripts, dados de entrada, saídas esperadas, ferramentas de apoio e relatórios gerados.

3 TRABALHOS RELACIONADOS

Este capítulo apresenta trabalhos relacionados ao aumento da resiliência de testes automatizados, selecionados pela diversidade de abordagens, entre inteligência artificial, heurísticas estruturais e estratégias de *fallback*. A ordem de apresentação reflete essa diversidade, destacando ao final a abordagem mais alinhada à proposta deste projeto.

3.1 FRAMEWORK DE TESTES SELF-HEALING COM INTELIGÊNCIA ARTIFICIAL E APRENDIZADO DE MÁQUINA

O trabalho de Saarathy, Bathrachalam e Rajendran (2024) propõe um framework de automação de testes com capacidade de auto-recuperação, baseado em IA e aprendizado de máquina (ML). O objetivo central é reduzir o esforço de manutenção causado por falhas em testes automatizados devido a mudanças frequentes na interface das aplicações web. A proposta combina técnicas como identificação dinâmica de localizadores, espera inteligente e detecção de anomalias, aplicadas a partir de modelos treinados com dados históricos. Os autores validaram sua abordagem em uma aplicação real, observando aumento na confiabilidade da suíte de testes (de 70% para 90%) e redução de 70% no tempo de manutenção, além de economia estimada de 60% nos custos relacionados.

Embora o trabalho apresente resultados relevantes, sua implementação exige um ecossistema complexo de ferramentas e uma infraestrutura robusta para o treinamento e uso de modelos de ML, o que pode limitar sua aplicabilidade em cenários com restrições técnicas. A proposta deste projeto, por sua vez, aborda a mesma problemática sob uma perspectiva mais acessível, priorizando soluções genéricas baseadas em heurísticas e estrutura do DOM, independentes de grandes volumes de dados ou configurações avançadas, e com maior facilidade de integração a ferramentas de mercado.

3.2 ROBULA+: GERAÇÃO DE LOCALIZADORES XPATH ROBUSTOS PARA TESTES WEB

O trabalho de Leotta *et al.* (2016) apresenta o algoritmo ROBULA+, voltado à geração automática de localizadores `XPath` mais robustos para testes de aplicações web. A proposta parte do reconhecimento de que os testes automatizados, embora amplamente utilizados, são altamente sensíveis a mudanças estruturais no DOM, sendo os localizadores um dos principais

pontos de fragilidade nos processos de automação.

O algoritmo ROBULA+ aplica um conjunto de heurísticas para construir expressões `XPath` que mantenham sua validade mesmo após atualizações na interface da aplicação. Entre as estratégias utilizadas estão a priorização de atributos considerados mais estáveis (como `id` e `name`), o uso combinado de múltiplos atributos, a exclusão de atributos frágeis (como `style`, `onclick`, entre outros) e a especialização progressiva de seletores a partir de um `XPath` genérico. Os testes realizados em oito aplicações reais mostraram que ROBULA+ reduz em até 90% a fragilidade dos localizadores em comparação com ferramentas tradicionais como Selenium IDE e FirePath, melhorando significativamente a robustez dos testes automatizados.

Embora o artigo não proponha um mecanismo de *self-healing* propriamente dito, ele contribui significativamente para o aumento da resiliência dos testes frente a mudanças estruturais. A proposta deste projeto se diferencia ao dar um passo além: além de buscar robustez na geração dos seletores, também trata da sua adaptação automática em tempo de execução, oferecendo uma abordagem reativa e autônoma à quebra de testes causada por alterações no DOM, garantindo a continuidade da execução.

3.3 SELF-HEALING COM INTELIGÊNCIA ARTIFICIAL EM PROCESSOS DE CONFORMIDADE REGULATÓRIA

O trabalho de Tamraparani e Dalal (2023) propõe uma abordagem de automação de testes voltada para instituições financeiras, com foco em atividades relacionadas à conformidade regulatória. A solução utiliza IA e ML para gerar e corrigir automaticamente scripts de teste, reduzindo o esforço manual em processos como detecção de fraudes e análise documental.

Apesar de promissor, o estudo deixa claro que essa abordagem depende de grandes volumes de dados, infraestrutura robusta e mão de obra especializada. Além disso, por atuar em domínios altamente controlados e com regras bem definidas, a solução se mostra limitada diante de cenários em que os elementos da interface mudam com frequência e sem padrão previsível. Nesta proposta, o foco está justamente em enfrentar esse tipo de desafio: adaptar os testes a mudanças estruturais no DOM em ambientes onde não há garantia de consistência ou padronização. Com isso, busca-se oferecer uma solução que seja capaz de operar mesmo em contextos incertos e em constante evolução, como é comum em aplicações web modernas.

3.4 ESTRATÉGIAS HEURÍSTICAS E SELETORES ALTERNATIVOS BASEADAS EM FALLBACK

Enquanto muitos estudos recentes têm focado no uso de IA para tornar testes automatizados mais resilientes, algumas abordagens alternativas se destacam por utilizar estratégias heurísticas e mecanismos de *fallback*, sem recorrer a IA. Um exemplo prático é apresentado pela empresa Wopee.io (2024), no qual são aplicadas tentativas sequenciais de múltiplos seletores — como os seletores `id`, `name`, `XPath` e `CSS` — para localizar elementos na interface. Caso o seletor principal falhe, o sistema tenta alternativas previamente definidas, utilizando também heurísticas simples como a correspondência textual flexível (*fuzzy matching*)[1].

Embora não seja um estudo acadêmico formal, esse tipo de solução prática reforça a relevância de estratégias baseadas em regras para aumentar a robustez dos testes automatizados. Logo, este trabalho serve como referência ao demonstrar que é possível implementar mecanismos de auto-recuperação eficazes sem depender de técnicas complexas de IA, alinhando-se à proposta aqui desenvolvida de explorar seletores alternativos e heurísticas como caminho viável para testes mais resilientes no contexto da automação de interfaces web.

[1] *Fuzzy matching* é uma técnica utilizada para comparar strings de forma aproximada, identificando similaridades mesmo quando há pequenas diferenças ou erros de escrita.

4 METODOLOGIA

Este capítulo apresenta a metodologia adotada para o desenvolvimento da biblioteca de *self-healing*, abrangendo seu funcionamento, a validação em diferentes frentes e a estratégia de disponibilização para uso pela comunidade.

A biblioteca será desenvolvida com base na linguagem `Python`, utilizando o *Selenium* como ferramenta para extração da estrutura do DOM. O funcionamento geral envolverá a captura do estado da página em dois momentos distintos: o tempo t_0 (DOM validado durante uma execução bem-sucedida dos testes) e o tempo t_1 (DOM capturado no momento de uma falha de execução). Esses dados serão armazenados no formato *JavaScript Object Notation* (JSON)[1], permitindo a detecção precisa de alterações na estrutura da página ao longo do tempo.

Durante a execução da suíte de testes automatizados, a biblioteca capturará *snapshots*[2] provisórios do DOM de cada tela visitada, de forma a registrar o contexto estrutural de cada etapa da execução. Esses registros permanecerão armazenados em memória temporária até a conclusão do teste. Caso o teste seja concluído com sucesso, os *snapshots* capturados serão promovidos a estados t_0 oficiais, atualizando a referência para futuras comparações. Em caso de falha, o DOM atual será capturado como t_1, permitindo a comparação direta com o último t_0 registrado para a respectiva tela. Detectadas diferenças significativas, a biblioteca corrige os seletores: se o framework de teste já utiliza um arquivo JSON para centralizar seus elementos, esse mesmo arquivo é atualizado automaticamente; caso contrário, é gerado um arquivo JSON contendo as sugestões de correção dos seletores. Dessa forma, busca-se minimizar a necessidade de intervenção manual e aumentar a resiliência das suítes de testes diante da evolução contínua da interface do sistema.

A validação da eficácia do mecanismo será realizada em quatro frentes complementares. Inicialmente, serão desenvolvidos testes unitários para assegurar o correto funcionamento dos componentes internos da biblioteca. Em seguida, será feita a aplicação prática da solução em ambientes simulados de automação, monitorando a capacidade de detecção de alterações no DOM, a atualização automática de seletores e a estabilidade dos testes reexecutados após o processo de *self-healing*. Além disso, serão conduzidas duas formas adicionais de avaliação: a submissão da solução ao *Technology Acceptance Model* (TAM)[3] e a realização de testes práticos

[1] JSON é um formato de dados leve e estruturado, usado para representar e trocar informações entre sistemas, especialmente em aplicações web.

[2] *Snapshot* é uma cópia instantânea do estado atual de uma estrutura em um determinado momento.

[3] TAM é um modelo teórico que avalia a aceitação de tecnologias com base na utilidade percebida e na facilidade de uso.

com um grupo de cinco usuários, visando avaliar a efetividade e a usabilidade da biblioteca no contexto de automação de testes.

Após a validação, a biblioteca será preparada para publicação no PyPI, permitindo sua instalação e utilização por meio do gerenciador de pacotes `pip`. A estrutura do projeto será organizada conforme as boas práticas de distribuição de pacotes Python, garantindo a documentação básica e a definição de metadados necessários. O processo de empacotamento incluirá a configuração do projeto para publicação e a adoção de um esquema de versionamento, assegurando o controle de atualizações futuras. Essa disponibilização permitirá que a biblioteca seja facilmente acessada e utilizada por profissionais da área de Qualidade de Software (QA), ampliando o alcance e a aplicabilidade da solução proposta em diferentes cenários de automação de testes, tanto em ambientes corporativos quanto acadêmicos.

5 CRONOGRAMA

O desenvolvimento do projeto será dividido em etapas, envolvendo a construção da biblioteca, sua publicação no PyPI e a conclusão do Trabalho de Conclusão de Curso (TCC)[1].

Atividades	**Mar**	**Abr**	**Mai**	**Jun**	**Jul**
Extração e armazenamento do DOM	X				
Comparação entre *snapshots* e detecção de alterações		X			
Adaptação de seletores			X		
Integração com frameworks de automação			X		
Avaliação da eficácia da biblioteca			X	X	
Empacotamento e publicação da biblioteca				X	
Escrita e revisão do TCC				X	X
Defesa do TCC					X

Tabela 1 – Cronograma de desenvolvimento do projeto

[1] TCC refere-se ao Trabalho de Conclusão de Curso, etapa final obrigatória de cursos de graduação.

6 CONCLUSÃO

O presente trabalho propôs o desenvolvimento de uma biblioteca para automação de testes web com capacidade de *self-healing*, visando aumentar a resiliência frente a alterações no DOM e reduzir o esforço de manutenção de scripts. A solução foi estruturada para oferecer configuração simples, integração flexível com diferentes frameworks e disponibilização pública através do PyPI, ampliando seu potencial de adoção em diversos ambientes de testes.

Ao longo da metodologia apresentada, foram definidos os mecanismos de detecção de alterações, adaptação de seletores e estratégias de validação técnica e prática, incluindo testes unitários, simulações controladas e avaliações com usuários. Com isso, espera-se que a biblioteca contribua para práticas de automação mais robustas, acessíveis e alinhadas às necessidades de equipes de QA em cenários ágeis e dinâmicos.

A conclusão deste projeto representará não apenas a entrega de uma ferramenta funcional, mas também um avanço prático na abordagem de problemas recorrentes em automação de testes, abrindo caminho para futuras evoluções e melhorias nesse campo.

REFERÊNCIAS

BATTINA, D. S. Artificial intelligence in software test automation: A systematic literature review. **Journal of Emerging Technologies and Innovative Research (JETIR)**, v. 6, n. 12, p. 1329–1336, 2019. Disponível em: <https://www.jetir.org/papers/JETIR1912176.pdf>.

FEWSTER, M.; GRAHAM, D. **Software Test Automation: Effective Use of Test Execution Tools**. New York: Addison-Wesley, 1999. ISBN 0201331403.

ISO/IEC/IEEE. **ISO/IEC/IEEE 29119-1:2022 – Software and systems engineering — Software testing — Part 1: Concepts and definitions**. 2022. <https://www.iso.org/standard/72206.html>. Norma internacional sobre conceitos fundamentais de teste de software.

ISTQB. **Syllabus CTFL v4.0 – Nível Foundation: Versão Brasileira**. 2023. <https://bstqb.online/ctfl>. Tradução oficial publicada pela BSTQB (Brazilian Software Testing Qualifications Board).

KHANKHOJE, R. Effortless test maintenance: A critical review of self-healing frameworks. **International Journal for Research in Applied Science and Engineering Technology**, v. 11, p. 1–12, 2023.

LEOTTA, M.; STOCCO, A.; RICCA, F.; TONELLA, P. Robula+: An algorithm for generating robust xpath locators for web testing. **Journal of Software: Evolution and Process**, John Wiley & Sons, v. 28, n. 3, p. 177–204, 2016.

MYERS, G. J.; SANDLER, C.; BADGETT, T. **The Art of Software Testing**. 3. ed. Hoboken, NJ: John Wiley & Sons, 2012.

SAARATHY, S. C. P.; BATHRACHALAM, S.; RAJENDRAN, B. K. Self-healing test automation framework using ai and ml. **International Journal of Strategic Management**, v. 3, n. 3, p. 45–77, 2024.

TAMRAPARANI, V.; DALAL, A. Self-generating & self-healing test automation scripts using AI for automating regulatory & compliance functions in financial institutions. **Revista de Inteligência Artificial em Medicina**, v. 14, n. 1, p. 784–796, 2023. Available online. Disponível em: <https://redcrevistas.com/index.php/Revista>.

Wopee.io. **Self-healing in SW Test Automation Explained**. 2024. Acesso em: 15 abr. 2025. Disponível em: <https://wopee.io/blog/self-healing-in-sw-test-automation/>.