FACULDADE DE TECNOLOGIA DE SÃO PAULO

PATRÍCIA VIANA AMENT

Testes Automatizados: Performance, qualidade e boas práticas com o uso do Robot Framework

SÃO PAULO
Junho/2023

FACULDADE DE TECNOLOGIA DE SÃO PAULO

PATRÍCIA VIANA AMENT

Testes Automatizados: Performance, qualidade e boas práticas com o uso do Robot Framework

Trabalho submetido como exigência parcial para a obtenção do Grau de Tecnólogo em Análise e Desenvolvimento de Sistemas Orientador: Prof.ª Me. Edméa Pujol Cantón

SÃO PAULO
Junho/2023

FACULDADE DE TECNOLOGIA DE SÃO PAULO

PATRÍCIA VIANA AMENT

Testes Automatizados: Performance, qualidade e boas práticas com o uso do Robot Framework

Trabalho submetido como exigência parcial para a obtenção do Grau de Tecnólogo em Análise e Desenvolvimento de Sistemas.

Parecer do Professor Orientador

O Trabalho de Conclusão de Curso da aluna Patrícia Viana Ament atendeu a todos as exigências do Departamento de Tecnologia da Informação.

Conceito/Nota Final: 10/10 -> Dez inteiros

Orientador: Profª Me. Edméa Pujol Cantón

SÃO PAULO, 20 de Junho de 2023.

Edméa Pujol Cantón

Assinatura do Orientador

AGRADECIMENTOS

Agradeço primeiramente a meus familiares, por todo apoio incondicional. Agradeço aos meus colegas de faculdade Angélica Gabas, Alexandre Feitosa, Juan Carloza e Marcelo Oliveira, pelo companheirismo e perseverança diante dos temores da vida acadêmica. Agradeço aos meus amigos da vida, Joyce Haji, Alex Ribeiro, Bruna Félix, Franklin Jones e Leonardo Nunes, pela paciência e compreensão em tempos de ausência. Agradeço ao meu companheiro de vida, Carlos Henrique. Sem você, nada disso seria possível.

E por fim, agradeço a minha orientadora, Professora Edméa Pujol Cantón, pelos conselhos e confiança aplicados neste trabalho.

“*O sucesso não consiste em não errar, mas em não cometer os mesmos equívocos mais de uma vez.*”

(George Bernard Shaw)

AMENT, Patrícia Viana. **Testes Automatizados**: Performance, qualidade e boas práticas com o uso do Robot Framework. 2023. 55 f. Graduação (Tecnologia em Análise e Desenvolvimento de Sistemas) – Faculdade de Tecnologia e São Paulo – FATEC-SP. São Paulo, 2023.

RESUMO

Neste trabalho são apresentados os objetivos, desafios e abordagens de escrita de testes automatizados utilizando Robot Framework. Aspectos de confiabilidade, manutenção, boas práticas e capacidade de feedback são ponderados, analisados e exemplificados com trechos de código para melhor explanação do assunto. Não obstante, são apresentadas abordagens e atributos singulares do framework, abordando as adversidades inerentes à construção de projetos de testes automatizados de qualidade. O entendimento dessas dificuldades apresenta-se como objetivo central deste estudo, contando com a exploração e evolução dos testes automatizados como referencial histórico à essa reflexão.

Palavras-chave: Testes automatizados, automação em testes, qualidade de testes automatizados, Robot Framework

AMENT, Patrícia Viana. **Automated Tests:** Performance, quality and best practices using the Robot Framework. 2023. 55 s. Graduation (Technology in Systems Analysis and Development) – Faculdade de Tecnologia e São Paulo – FATEC-SP. São Paulo, 2023.

ABSTRACT

In this work, objectives, challenges and approaches to writing automated tests using Robot Framework are presented. Aspects of reliability, maintenance, good practices and feedback capacity are considered, analyzed and exemplified with code snippets for a better explanation of the subject. Nevertheless, unique approaches and attributes of the framework are presented, addressing the adversities inherent in the construction of quality automated test projects. Understanding these difficulties is the central objective of this study, relying on the exploration and evolution of automated tests as a historical reference for this reflection.

Keywords: Automated tests, test automation, quality of automated tests, Robot Framework

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

QA	Quality Assurance
RF	Robot Framework
POM	Page Object Model

SUMÁRIO

INTRODUÇÃO

Diante de um mercado altamente competitivo, organizações e equipes são desafiadas a entregar produtos de software com qualidade em intervalos de tempo cada vez menores. Segundo Bernardo e Kon (2008, p.1), “os sistemas de software devem não só fazer corretamente o que o cliente precisa, mas também fazê-lo de forma segura, eficiente e escalável e serem flexíveis, de fácil manutenção e evolução”. A pressão de lidar com prazos apertados somada às expectativas de clientes e usuários não exime dos desenvolvedores e equipes de Quality Assurance (QA) a responsabilidade de realizar testes de software ao longo do seu processo de desenvolvimento.

> A reprodutibilidade dos testes permite simular identicamente e inúmeras vezes situações específicas, garantindo que passos importantes não serão ignorados por falha humana e facilitando a identificação de um possível comportamento não desejado. (BERNARDO, 2011, p.10).

Sob essa perspectiva fomenta o segmento de automação de testes, em amplo crescimento e determinante ao ciclo de desenvolvimento de softwares, considerando que “o esforço associado a testar um software perfaz cerca de 50% das despesas totais de desenvolvimento, podendo ser um valor mais elevado no caso de aplicações críticas” (FONSECA, 2021, p.2). Nos últimos anos temos o surgimento de diversas ferramentas que sustentam o propósito de validar, quantificar e qualificar aspectos gerais de softwares, utilizadas segundo suas particularidades e vantagens de implementação.

Para atender tal demanda, o ramo de automação se apresenta como resposta a atender diversas frentes, oferecendo opções capazes de validar desde a qualidade do código fonte até o nível mais alto de integração, como a modalidade de testes de interface. E é diante de tais soluções que se destaca o Robot Framework (RF), framework de automação open source genérico, aberto e extensível. “Pode ser integrado a praticamente qualquer outra ferramenta para criar soluções de automação poderosas e flexíveis” (ROBOT FRAMEWORK FOUNDATION, 2022), apresentando uma interessante alternativa para testes de aceitação e regressivos. Como descreve Chinnaswamy et al. (2018, p.11), “o Robot Framework executa o processo de teste chamando palavras-chave, que podem ser escritas usando as linguagens Python ou Java”. Sua flexibilidade, fácil aprendizado e sem custos fazem do Robot Framework

uma oportuna ferramenta de validação de software, sendo o seu uso, vantagens e boas práticas o objeto central do trabalho.

Apresentação do Problema

A crescente demanda por produtos e serviços de qualidade impulsionam empresas e organizações a buscarem soluções de mercado que atendam esse cenário. Como consequência crescem as alternativas e recursos disponíveis, ferramentas que prezam pela verificação, validação e qualidade de softwares.

Em um mundo diversificado, competitivo e global, aspectos como qualidade, performance e confiabilidade podem ser o diferencial competitivo para marcas e negócios. Em meio a tantas opções, escolher o melhor software ou framework como instrumento de testes requer das equipes envolvidas muita pesquisa, capacitação e estudos de caso. Não obstante, existem os fatores de recursos financeiros e humanos, conforme cita Myers (2012), “um caso de teste que consegue encontrar um novo erro prova ser um valioso investimento”.

Ante essa problemática, como os testes automatizados se tornam essenciais para os ramos de Engenharia de Software e Quality Assurance, destacando o Robot Framework como elemento central a essa reflexão?

Objetivo Geral

Este trabalho tem como objetivo geral estudar aspectos de performance, qualidade, manutenibilidade e boas práticas adotadas no uso da ferramenta de testes automatizados Robot Framework.

Objetivos Específicos

Enquanto objetivos específicos, este trabalho pretende:

- Pesquisar a evolução histórica dos testes automatizados, em especial as técnicas desenvolvidas e aprimoradas ao longo do tempo;
- Reconhecer a importância dos testes automatizados, enfatizando o uso do RF como alternativa estratégica para organizações, evidenciando seu uso flexível, escalável e pertinente enquanto instrumento validador de softwares;

- Apresentar as dificuldades inerentes às tarefas de desenvolvimento e manutenção dos scripts de automação do framework, no tocante em manter o nível de confiabilidade e robustez dos testes;
- Expor as principais bibliotecas extensíveis ao RF, sua implementação e funcionalidade, considerando boas práticas e vantagens.

Metodologia

O trabalho foi desenvolvido adotando revisão bibliográfica como metodologia de pesquisa, abordando técnicas, performance, capacidade de feedback e os principais aspectos do Robot Framework. Em paralelo, realizado o levantamento de boas práticas e sua aplicação, exemplificando o bom uso do framework de automação. Não obstante, conceitos, metodologias e evolução histórica dos testes automatizados são explanados como embasamento teórico.

Justificativa

Produtos de software fazem parte da rotina da humanidade, intensificada nas últimas décadas devido ao crescimento exponencial da tecnologia. O surgimento de smartphones, Internet das Coisas, Inteligência Artificial e demais ramos da Tecnologia que podem ser associados a esse movimento representam hoje o cerne da vida em sociedade, presente em todos os países e povos, mesclando hábitos e costumes em diversos níveis**.** O uso da tecnologia da informação chegou a um ponto no qual pequenas empresas, grandes corporações e até mesmo o governo simplesmente não podem mais sobreviver sem o uso de computadores e sistemas de informação (LOVELAND, 2005).

Em paralelo à sua construção, a preocupação com a qualidade do software é uma constante presente no processo de desenvolvimento, o que requer de seus envolvidos tempo e técnica para criar testes capazes de mapear eventuais falhas do projeto. Entretanto, ocasião e recursos necessários para executar referidos testes se apresentam um desafio. Sob essa ótica, testes automatizados comparecem como alternativa, ofertando como solução ferramentas capazes de validar regras de negócios, confiabilidade e usabilidade de softwares. Como boa prática, o processo de testes deve ocorrer em simultâneo ao processo de desenvolvimento do software,

partindo do levantamento de requisitos e modelagem, pois erros encontrados já nas fases iniciais do projeto podem gerar economia de recursos e correção.(BASTOS ET AL., 2007).

Diante do exposto, o estudo e compreensão dos testes automatizados se torna essencial para os ramos de Engenharia de Software e QA, destacando-se o Robot Framework como objeto central a essa reflexão. Enquanto ferramenta de testes, são muitas as vantagens decorrentes do RF que se estendem à retenção de custos, recursos e tempo, que serão amplamente discutidos no trabalho.

Estrutura do Trabalho

O trabalho segue a estrutura:

Em introdução, apresentada a contextualização histórica do objeto da pesquisa, abordando em sequência a questão central do trabalho. Em seguida, apresentados os objetivos geral e específicos da pesquisa, finalizando com a exposição da metodologia adotada para composição do trabalho e justificativa do seu desenvolvimento.

O desenvolvimento foi dividido em quatro capítulos, compondo o dois primeiros capítulos os aspectos introdutórios e gerais da automação de testes seguidos pelos dois capítulos finais dedicados a explorar os assuntos pertinentes ao Robot Framework.

No primeiro capítulo há a apresentação do tema geral do trabalho, abordando os fundamentos do teste de software com a definição das principais categorias, abordagens e técnicas inerentes à compreensão da área de Quality Assurance.

Em sequência, o segundo capítulo visa apresentar algumas ferramentas de interesse às equipes quanto ao gerenciamentos das atividades que envolvem o processo de teste, visando abordar como o bom uso dessas ferramentas impactam a qualidade das aplicações testadas. Aborda também a automação de testes, descrevendo sua definição e modalidades de atuação e expondo seu uso em aplicações Mobile, Backend e Frontend, finalizando com a apresentação de Frameworks de Automação e algumas de suas opções de mercado, tópico que servirá de introdução ao Robot Framework.

Iniciando o tema central do trabalho, o terceiro capítulo aborda o Robot Framework a nível introdutório, apresentando sua arquitetura, ferramentas de edição disponíveis para desenvolver seus testes e quais os métodos de escrita de scripts possíveis com o framework.

O quarto e último capítulo apresenta os principais aspectos do Robot Framework, com ênfase em abordar suas características enquanto estrutura de código e aplicação de boas práticas, apresentando conceitos e métodos voltados para manutenção, acompanhamento e controle de performance e sua capacidade de feedback.

Em conclusão são expostas as reflexões e resultados obtidos, apresentando também sugestões para pesquisas futuras.

1. ABORDAGENS, ROTINAS E TÉCNICAS DE TESTES DE SOFTWARE

A crescente demanda tecnológica em quase todas as áreas da vida em sociedade ascendeu também os riscos e falhas em utilizar softwares. Prejuízos de baixa qualidade de navegação, perda de memória e questões mais críticas como vazamento de dados são alguns dos exemplos que preocupam organizações e equipes de engenharia de software. Neste contexto, surge uma área de engenharia de software preocupada com a qualidade dos processos, produtos e serviços de software, denominada Quality Assurance (PRESSMAN, 2016).

Com foco na qualidade e robustez do software, os processos de QA envolvem a elaboração de protocolos, padrões e técnicas de qualidade que visam garantir que o software seja testado de maneira consistente e eficaz (SOMMERVILLE, 2011). Neste contexto incluem-se validações que partem da camada mais interna do software, seu código-fonte, integrando demais níveis de teste ao longo do desenvolvimento do projeto. Cabe ainda à área de QA identificar quaisquer falhas e erros do software, de maneira que sejam corrigidos antes de comercializados ao usuário final. Portanto, as equipes de QA devem adotar uma abordagem sistemática e proativa com a definição de padrões e constante revisão aos testes aplicados, desenvolver comunicação e documentações adequadas, compreendendo todos os envolvidos (SNYDER, 2002).

A área de QA abrange nomenclaturas, técnicas e abordagens próprias para alcançar seu propósito de examinar softwares. Nos tópicos seguintes deste capítulo são descritos alguns dos seus principais conceitos.

1.1 Técnicas de Teste

Dividida em duas principais áreas de atuação, as técnicas de teste agrupam determinadas abordagens que auxiliam as equipes que as utilizam a identificar falhas do software sob seu escopo de atuação. Como característica, cada área compreende testes que examinam o software do ponto de vista macro ou micro, conforme abordado nos tópicos a seguir.

Técnica Estrutural

A técnica estrutural compreende o conjunto de testes baseados em validar a estrutura interna do software a ser testado. Preocupado em validar as engrenagens que compõem a aplicação, essa abordagem é também reconhecida como técnica de caixa-branca ou caixa-transparente. Nesse cenário, o testador de software, ou mesmo o desenvolver da aplicação, tem como objetivo validar o código fonte do projeto, examinando classes, métodos e funções desenvolvidas.

> Usando métodos de teste caixa-branca, o engenheiro de software pode criar casos de teste que (1) garantam que todos os caminhos independentes de um módulo foram exercitados pelo menos uma vez, (2) exercitem todas as decisões lógicas nos seus estados verdadeiro e falso, (3) executem todos os ciclos em seus limites e dentro de suas fronteiras operacionais e (4) exercitem estruturas de dados internas para assegurar a sua validade. (MAXIM e PRESSMAN, 2016, p.500)

Esta abordagem também é constituída por um conjunto de classes de técnicas, conforme lembram Maxim e Pressman (2016, p.500-508): teste do caminho básico, teste de estrutura de controle, teste de fluxo de dados e teste de ciclo.

Técnica Funcional

A técnica funcional está voltada, como o nome sugere, a inspecionar as funcionalidades do sistema. Sua abordagem parte do estudo dos requisitos funcionais e não funcionais documentados na especificação do projeto testado, propondo assim validar os aspectos externos ao seu código-fonte. Em síntese, tem como procedimento de teste validar dados de entradas e saídas esperadas baseado nas funcionalidades desenvolvidas no software. Devido a essa visão alheia ao cerne da aplicação também é reconhecida como técnica de caixa-preta. Importante ressaltar que a seleção de testes e técnicas escolhidas pelo testador não sobrepõem a utilidade de outras abordagens, conforme lembram Maxim e Pressman (2016, p.509) “o teste caixa-preta não é uma alternativa às técnicas caixa-branca. Em vez disso, é uma abordagem complementar, com possibilidade de descobrir uma classe de erros diferente daquela obtida com métodos caixa-branca.”

Considerando tempo e recursos finitos torna-se impraticável aplicar testes ponderando todos os cenários possíveis, exigindo, portanto, criteriosidade no momento de definir o escopo de validações. Não obstante, tem como composição o

conjunto de classes de técnicas compreendendo outras abordagens, conforme citam Maxim e Pressman (2016, p.509-519): baseado em grafos; particionamento de equivalência; análise de valores limites; teste de matriz ortogonal; baseado em modelos; teste da documentação e dos recursos de ajuda; teste para sistemas em tempo real e padrões para teste de software.

1.2 Categorias de Teste

As categorias de testes agrupam conjunto de técnicas que auxiliam os testadores a validar aspectos específicos do software. As categorias de testes se preocupam a testar questões como performance, segurança e demais áreas substanciais à qualidade do software. A seguir, algumas das principais categorias utilizadas e suas respectivas responsabilidades.

Teste de Desempenho

Os testes de desempenho são desenhados para emular situações de estresse e carga comuns do mundo real. Seu objetivo é identificar se a aplicação está preparada para lidar com cenários de sobrecarga, como acesso simultâneo de múltiplos usuários a uma aplicação Web, avaliando aspectos como degradação do tempo de resposta do servidor, especificando em qual momento considera-se notável e inaceitável seu desempenho. Cabe a esse tipo de teste identificar eventuais falhas e configurações que comprometam a performance da aplicação, pois “uma vez que o propósito do teste é demonstrar que o programa não atende aos seus objetivos, os casos de teste devem ser projetados para mostrar que o programa não satisfaz seus objetivos de desempenho” (MYERS, 2012, p.126).

Teste de Segurança

Devido à crescente apreensão dos consumidores com sua privacidade e dados, produtos e serviços digitais passam a considerar aspectos de segurança como objetivo. Testes de segurança surgem nesse contexto para mensurar a validade dessas especificações (MAXIM e PRESSMAN, 2016). Cabe a esse tipo de teste cobrir possíveis cenários de falha em diversos níveis da aplicação, como submeter os

mecanismos de proteção de memória do sistema, sua base de dados e demais aspectos, considerando que os níveis dessas validações são aplicados de acordo com o tipo de programa a ser testado.

Teste de Usabilidade

O teste de usabilidade tem a pretensão de validar o nível mais alto da aplicação, emulando comportamentos do usuário final, aproximando o caso de teste de situações eventualmente vivenciadas pelo usuário no mundo real, examinando, conforme citam Mili e Tchier (2015, p.18) ,"[...] até que ponto o produto de software é fácil de usar e de personalizar de acordo com as necessidades e circunstâncias do usuário final". Segundo os autores, existem cinco atributos que podem ser considerados para mensurar os aspectos de usabilidade, citados abaixo (MILI e TCHIER 2015, p.18-19):

- Fácil de Usar: ponto relevante do teste, considerando que deve validar aspectos como simplicidade e padrões de interações, acesso a menus de ajuda, vocabulário simples dentre outras questões, ponderando que tais características devem ser claras e acessíveis aos diversos grupos de usuários.
- Fácil de Aprender: valida os atributos do software enquanto seu sistema intuitivo de interações, seus padrões e protocolos.
- Customização: mensura o nível de customização do software, considerando suas capacidades de adaptação de requisitos funcionais do usuário final, alterado pelo usuário final. Busca validar o nível de controle do usuário em termos de personalização do software.
- Calibrabilidade: este aspecto avalia o nível de controle do usuário final em ajustar requisitos operacionais do software.
- Interoperabilidade: por fim, este aspecto mede a capacidade de um software em operar com outras aplicações, levando em consideração métricas quantitativas e qualitativas para medir sua interoperabilidade.

Uma das vantagens do teste de usabilidade é sua abordagem de alto nível, capaz de encontrar falhas no sistema que não seriam detectados por testes automatizados e regressivos. A FIGURA 1 ilustra alguns dos pontos apresentados neste tópico, exemplificando seu uso em um teste de aplicação Web.

FIGURA 1- Aspectos de Usabilidade em Aplicações Web

Fonte: Maxim e Pressman (2016).

Teste de Compatibilidade

Com a disponibilidade de diferentes sistemas operacionais e navegadores, os softwares devem ser capazes de ajustes e compatibilidades conforme as circunstâncias de configurações e usos disponíveis. Nesse contexto, cabe aos testes de compatibilidade validar os objetivos de compatibilidade e conversão projetados (MYERS, 2012).

1.3 Níveis de Teste

Os diferentes níveis de aplicação de testes auxiliam as equipes testadores a definir escopo de validação, abordando determinadas categorias e técnicas a depender do nível de desenvolvimento do software. Suas nomenclaturas e responsabilidades são descriminadas nos tópicos que seguem.

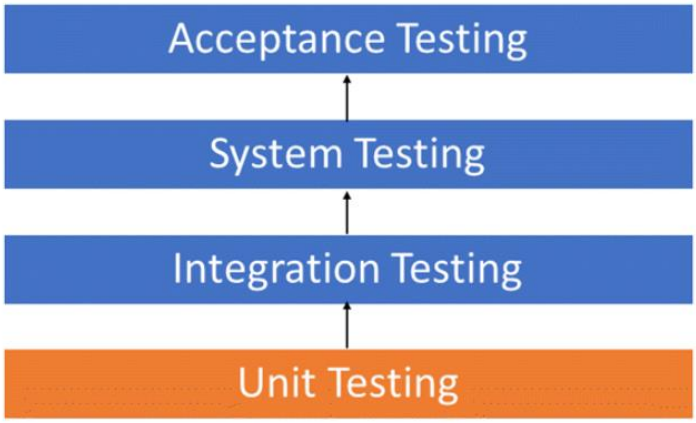
Teste de Unidade

Responsável por validar as unidades e componentes individuais de um software, o teste de unidade corresponde ao primeiro nível de teste, comprometido em validar se cada unidade do código do software atende conforme o esperado, isolando o código-fonte da aplicação em sessões. Assim, cada unidade testada pode compreender uma função, método ou objeto do código.

Devido sua particularidade, o teste de unidade é considerado como uma técnica de teste de caixa-branca, desenvolvido durante a fase de codificação do software e predominantemente aplicado pelos desenvolvedores. Entretanto, pode ser aplicado por profissionais de QA devido à falta de tempo e recursos do projeto (HAMILTON, 2023).

O uso de teste de unidade durante o processo de construção de software se apresenta como estratégico para as equipes de desenvolvimento, pois é um instrumento validador que auxilia na redução de tempo, prevendo falhas no início do ciclo de desenvolvimento, além de agregar na documentação do projeto. Entretanto, cabe aos desenvolvedores técnica para desenvolver testes correspondentes, considerando que testes inadequados podem apresentar alto custo de correção nos próximos níveis de teste, como os testes de sistema e integração (HAMILTON, 2023). Na FIGURA 2 é ilustrado os níveis de testes esperados no ciclo de desenvolvimento de software, evidenciando como camada primária o teste de unidade.

FIGURA 2 - Níveis de Teste



Fonte: HAMILTON, Thomas (2023).

Teste de Integração

O teste de integração pode ser definido como o tipo de teste em que as unidades ou componentes de um software são testados em grupo com o objetivo de identificar e expor possíveis falhas quando integrados. Mesmo que testados minuciosamente no nível de teste de unidade, a integração com demais componentes do código pode inferir em erros. Sendo um software desenvolvido em partes, módulos

e objetos e de autoria, muitas vezes, de diversos desenvolvedores, são evidentes as chances de falha, conforme exemplificam Maxim e Pressman (2016, p. 475-476):

> Dados podem ser perdidos através de uma interface; um componente pode ter um efeito inesperado ou adverso sobre outro; subfunções, quando combinadas, podem não produzir a função principal desejada; imprecisão aceitável individualmente pode ser amplificada em níveis não aceitáveis; estruturas de dados globais podem apresentar problemas.

O teste de integração tem como foco validar, portanto, a qualidade de comunicação de dados entre os diversos módulos da aplicação. Devido a essa característica pode também ser reconhecido como ‘I&T’ (Integração e Teste), ‘Teste de String’ e ‘Teste de Thread’ (HAMILTON, 2023).

Teste de Sistema

Considerado como o nível mais complexo e incompreendido, o teste de sistema tem como objetivo testar o software como um todo, sendo este aplicável quando boa parte ou todos os componentes do sistemas se encontram integrados. Não obstante, é neste nível que o sistema é testado segundo os requisitos do usuário final.

Tendo como objetivo encontrar discrepâncias entre o sistema e seu propósito, este teste é vital ao ciclo de desenvolvimento de software, se consideradas as expectativas de qualidade esperadas neste nível da aplicação, momento em que a aplicação é vista como produto, portanto, o grau de qualidade e rigidez dos testes infere em maior ocorrência de falhas encontradas (MYERS, 2012).

Teste de Aceitação

Sendo o último nível de teste, o teste de aceitação antecede a implementação do software. Antes de ser disponibilizado ao usuário final, o teste de aceitação é aplicado para validar se o software está preparado para ser comercializado, se atende às funcionalidades e tarefas desenvolvidas na aplicação. Em síntese, o teste de aceitação verifica as falhas do sistema como um todo, validando sua estabilidade. Ademais, cabe ao teste confirmar se o sistema atende às regras de negócio estabelecidas nos requisitos do projeto (GAIDARGI, 2021).

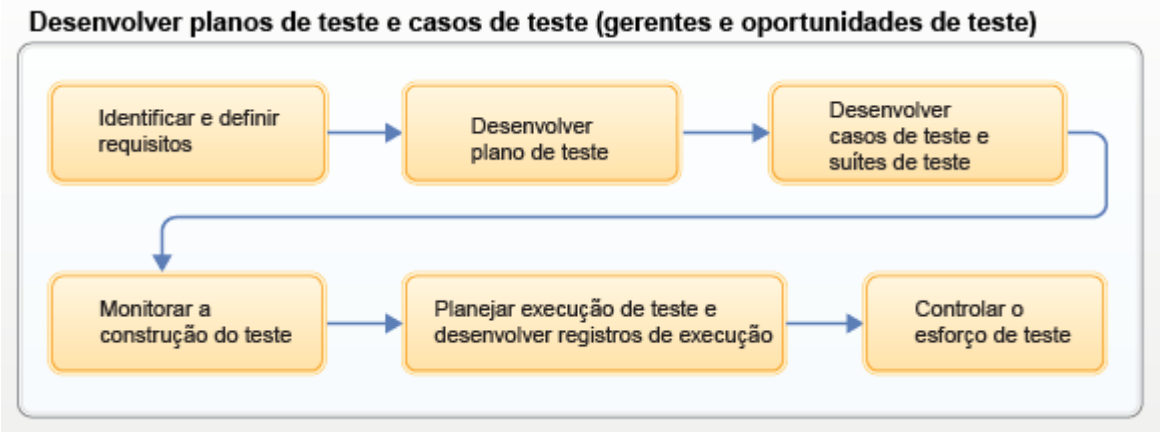
2. GERENCIAMENTO E AUTOMAÇÃO DE TESTES

O Gerenciamento de Teste é um processo adotado pelos *Test Managers* de QA que consiste em gerenciar as atividades voltadas ao teste, com o objetivo de garantir seu alto desempenho e qualidade quando aplicado.

> Para qualquer esforço de engenharia de software considerável, o gerenciamento de teste é um assunto importante. Conforme faz qualquer processo de gerenciamento, o gerenciamento de teste trata com diferentes aspectos de organização, planejamento, preparação de equipe, monitoramento e relatório do processo de teste a ser seguido. (IBM, 2022)

A prática de Gerenciamento de Teste conta com fases que acompanham do levantamento de requisitos até a monitoração de desempenho dos testes, conforme ilustrado na FIGURA 3:

FIGURA 3 – Fases do Gerenciamento de Teste



Fonte: IBM, 2022.

São de responsabilidade de cada fase:

- Identificar e definir requisitos: Etapa inicial do planejamento de teste e responsável em identificar os esforços envolvidos em sua elaboração, orientado pelos requisitos usados no desenvolvimento do projeto testado.
- Desenvolver plano de teste: Compete ao plano de teste definir o escopo da validação. Nesta etapa são definidos os ciclos de teste e demais atividades pertinentes ao escopo de validação conforme os requisitos levantados na etapa anterior. Segundo IBM (2021) “normalmente, um plano de teste identifica requisitos, riscos, casos de teste, os ambientes de teste a serem testados, objetivos de qualidade e negócios, planejamentos de teste e outros itens”. Com

o escopo definido, demais aspectos de negócio como estimativas e prazos já são pontuados nesta etapa.

- Desenvolver casos de teste e suíte de teste: Nesta fase são desenvolvidos os testes ou suítes de testes seguindo requisitos, escopo e cronograma definidos. É possível nesta etapa desenvolver apenas um caso de teste com o objetivo definir o que será testado. A partir dele, testes são desenvolvidos para garantir que o software atenda conforme os requisitos. Outra abordagem seria a criação de suítes de testes, conjuntos de casos de testes que são agrupados com o mesmo intuito de validar a qualidade do software.
- Monitorar construção do teste: Etapa designada a acompanhar a construção dos scripts, com a preocupação de validar se os testes são desenvolvidos conforme o caso de teste ou suítes de testes definido.
- Planejar execução de teste e desenvolver registro de execução: Nesta fase é definida a estratégia a ser usada para manter o registros das execuções dos testes, se serão necessárias ações manuais ou automatizadas para manter esses registros, além de alinhamentos como organização, nomenclatura e políticas de controle e execução dos testes.
- Controlar o esforço de teste: Realizado o acompanhamento das execuções dos testes do ponto de vista qualitativo, com o objetivo de mensurar se as expectativas reais correspondem às expectativas definidas na fase inicial do planejamento.

A adoção do Gerenciamento de Testes se apresenta como instrumento necessário às equipes de QA, haja visto a necessidade de acompanhar, controlar e organizar os testes desenvolvidos. Para seu uso, existem ferramentas disponíveis no mercado que auxiliam no planejamento, desenvolvimento, execução e gerenciamento de atividades de testes. Como exemplo, cita-se TestLink, qTest e Zephir. A adoção da ferramenta deve ser acompanhada de estudo prévio, de modo que seja aderente às necessidades do *Test Manager* e sua equipe.

Automação de Testes

As atividades complexas e inerentes à Engenharia de Software não excluem das equipes de desenvolvimento a responsabilidade de testar continuamente seus projetos. Neste contexto, a automação de testes apresenta-se como interessante

instrumento de validação, agilizando o processo de verificação constante da aplicação somado ao benefício de reduzir falhas no teste por não necessitar de ação manual. A automação de testes é definida por Maxim e Pressman (2016, p. 568) como “processo que envolve o uso de ferramentas de software para escrever e executar testes de software". Não obstante, o autor defende a relevância da automação de testes observando sua capacidade de garantir maior qualidade do teste e redução de custos de desenvolvimento.

Percebe-se o ganho da automação quando analisados, por exemplo, testes aplicados a fim de validar a funcionalidade do sistema como o todo, como os testes de regressão. Desenvolvidos para testar a integridade dos requisitos do software ao longo de suas entregas, é evidente o aumento de complexidade e recurso dedicados conforme evolução do projeto. A automação do teste reduziria tempo e, consequentemente, custos no desenvolvimento dessa atividade.

Outro aspecto vantajoso da automação dos testes é sua capacidade de feedback, permitindo que o desenvolvedor ou testador possa identificar em tempo anomalias que devem ser corrigidas (DUVAL, 2007). Ademais, devido sua habilidade em validar regras de negócio, os testes automatizados também servem de documentação para as equipes de desenvolvimento.

Os demais tópicos deste capítulo abordam como a automação de testes é caracterizada conforme seu escopo de atuação.

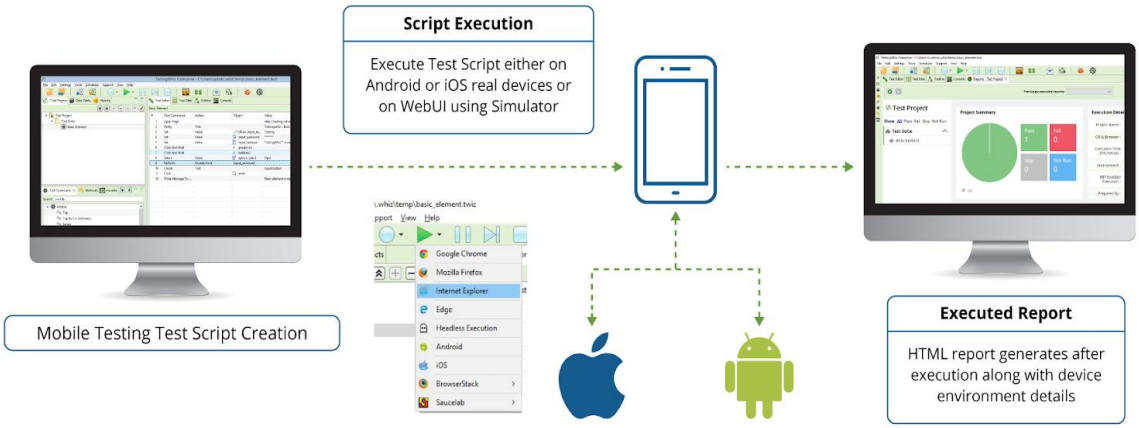
2.1 Automação de Testes para Mobile

De acordo com o relatório estatístico divulgado pela Statista (2023), os dispositivos móveis são responsáveis por 58,33% do tráfego mundial da Web. Um dos motivos que justifica esse resultado é a situação precária de países emergentes, fomentando um mercado digital voltado a uso da internet por meio de dispositivos móveis (BIANCHI, 2023). Não obstante, redes sociais, serviços de streaming e acesso a e-mail se destacam pela sua popularidade de consumo via celulares e tablets.

Os testes automatizado em aplicações mobile auxiliam na redução de ações manuais e onerosas quanto a validar a segurança e estabilidade do sistema. Sua estrutura geral é constituída pelo script de automação criado pelo testador, executado no software em ambiente controlado em dispositivos Android, iOS ou mesmo em um

emulador Web, gerando por fim relatório detalhado com os resultados. A ilustração dessa estrutura pode ser observada na FIGURA 4.

FIGURA 4 - Execução de testes automatizados em aplicativos móveis



Fonte: Mobile Automation Testing Steps and Process (2020).

Alguns tipos de testes automatizados podem ser aplicados em aplicativos móveis, cita-se:

- Teste de Interrupção: responsável por validar a capacidade do aplicativo em ser executado enquanto demais aplicativos são executados simultaneamente, a exemplo de notificações recebidas durante seu uso e que podem comprometer o funcionamento do aplicativo testado.
- Teste de Compatibilidade: determinado a validar a compatibilidade do aplicativo em certo sistema operacional, além de sua capacidade de sincronização com um ou mais aplicativos.
- Teste de Penetração: responsável por simular cenários de ataques virtuais a fim de encontrar qualquer vulnerabilidade que comprometa a segurança do aplicativo.

2.2 Automação de Testes para Backend

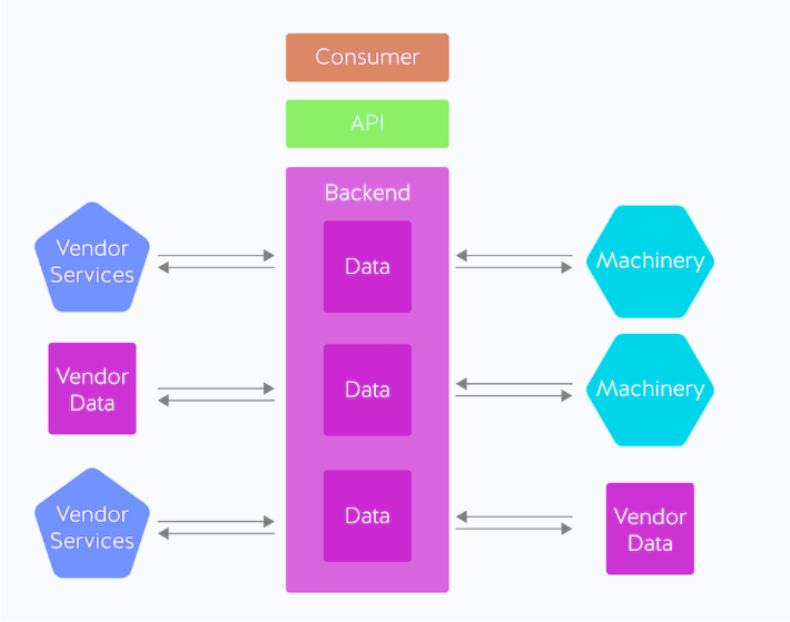
Com o intuito de validar em softwares a integridade das funcionalidades e seu desenvolvimento no código-fonte, os testes de backend buscam verificar se essa camada da aplicação apresenta vulnerabilidades nos processos de consulta e

armazenamento de dados à base integrada (KHANDELWAL, 2019). Devido à sua natureza, também são conhecidos como testes de banco de dados.

Os testes de backend são desenvolvidos e implementados desde as fases iniciais do projeto devido sua vantagem de identificar prontamente quaisquer problemas relacionados ao servidor de banco de dados. Por não necessitar obrigatoriamente de uma interface gráfica pode ser facilmente adotado testes automatizados, representando ganho de tempo e recursos.

Tratando-se de uma validação voltada a testar as regras de negócio e sua integração com o banco de dados, os testes de backend são comumente implementados em todos os tipos de software devido sua capacidade de validar o cerne da aplicação. A FIGURA 5 representa as interações que compõem a camada do backend e sua comunicação expressiva com a base de dados, corroborando a necessidade de testes e acompanhamento contínuo.

FIGURA 5 – Exemplificação das interações backend



Fonte: KHANDELWAL, Abhik (2019).

Nessa camada da aplicação podem ser realizados testes que se enquadram tanto nas técnicas estrutural e funcional, abordadas no capítulo 1 deste trabalho. Exemplos de testes utilizados nesse contexto são os testes de API e SQL, requerendo dos testadores conhecimento em consultas SQL e estrutura de banco de dados (KHANDELWAL, 2019).

2.3 Automação de Testes para Frontend

Os testes automatizados em frontend caracterizam-se pela sua capacidade de validar aspectos gerais que envolvem a camada de interação com o usuário, como usabilidade e funcionalidade de seus elementos. Tem por objetivo verificar quaisquer falhas que possam impactar a interação e experiência do usuário antes que o software seja comercializado, assim, evidencia-se a importância de aplicar esse tipo de teste (KHANDELWAL, 2019).

Tem como objetivo a exploração das funcionalidades disponíveis ao usuário final, de maneira a cobrir cenários e requisitos projetados. Portanto, os testes visam verificar elementos como botões, menus e logins, além de situações adversas como falha de acesso com intuito de mensurar desempenho e qualidade da interface testada. São comumente usados testes de aceitação, regressão e unitários, exigindo dos testadores sólido conhecimento das regras de negócio da aplicação, de maneira a verificar se são aplicados com qualidade e se correspondem às expectativas de uma experiência do usuário de alto nível.

Os testes automatizados para esse tipo de validação são realizados na própria interface da aplicação, testando sua usabilidade e funcionalidades desenvolvidas para o usuário. Existem no mercado diversas opções de frameworks de automação que auxiliam nessa tarefa, contando com recursos de automação que reduzem tempo e recursos dedicados a esse tipo de teste. Outra vantagem é a geração de relatórios presente nesses frameworks, permitindo às equipes de QA o acompanhamento detalhado dos resultados obtidos. Portanto, cabem às equipes sólido conhecimento em frameworks de automação, de maneira que possam selecionar a mais adequada às necessidades do projeto.

2.4 Frameworks de Automação

Segundo Badkar (2023), frameworks de automação podem ser entendidos como o conjunto de padrões de codificação de scripts e ferramentas destinados à criação de testes, utilizados para evitar o esforço e tempo necessários se aplicados manualmente pelas equipes de Engenharia de Software e QA. Contando com outros benefícios como melhor manipulação de dados do teste e reutilização de código, os frameworks de automação representam ganho significativa para as equipes que se

dedicam a usar da arquitetura de automação de testes, conforme destaca Hamilton (2023):

- Redução de risco de falhas devido a não interferência manual durante a execução dos testes;
- Ganho de eficiência, com maior agilidade e desempenho devido sua capacidade de execução contínua, permitindo a repetibilidade dos testes;
- Redução dos custos de manutenção dos scripts;
- Reutilização de código pois, dependendo do seu desenvolvimento, pode ser reaproveitado em outros testes pelo framework;
- Maior cobertura dos casos de uso, pois a execução automatizada permite o alcance a diversos cenários em menor tempo se comparado com a performance manual.

Os frameworks de automação disponíveis no mercado contam com características próprias, capazes de atender a diversas organizações. Ainda, dependendo da complexidade e necessidades do projeto, é possível adotar um ou mais frameworks.

Na FIGURA 6 são apresentados alguns dos frameworks de automação conhecidos no mercado, expondo suas respectivas vantagens e desvantagens:

FIGURA 6 - Frameworks de Automação

Framework	Vantagens	Desvantagens
Selenium	Código aberto Suporta diversas linguagens de programação Suporta maioria dos navegadores Forte comunidade ativa	Gerenciamento de depedências com alta demanda de manutenção Dependência de Webdriver Dificuldades de interpretação de frameworks modernos como React e Angular
cypress.io	Fácil instalação Suporta diversos tipos de teste Permite gravação em vídeo dos testes	Suporta apenas as linguagens JavaScript e TypeScript Não suporta a abertura de várias guias e/ou janelas do navegador durante o teste Dificuldade de configuração em caso de testes em paralelo
Playwright	Suporta diversas linguagens de programação Sem dependência de Webdriver Suporta execução de testes em navegadores diferentes	Suporta apenas a versão de código aberto do Safari (WebKit) Dificuldade em reconhecer idiomas diferentes durante a execução
WEBDRIVER I/O	Código aberto Suporta maioria dos navegadores Altamente extensível, com opções confiáveis de pacotes e plug-ins	Suporta apenas as linguagens JavaScript e TypeScript Documentação pouco amigável para iniciantes Dificuldade de configuração
appium	Código aberto Suporta diversas linguagens de programação Suporta testes baseados em nuvem	Dificuldade de configuração Dificuldade de integração de relatórios Suporte dificultado em aplicativos híbridos

Fonte: Elaborado pela autora (2023), com base em Badkar, 2023.

A construção dos scripts em frameworks de automação pode ser desenvolvida segundo às possibilidades da ferramenta. Abaixo, algumas dessas abordagens :

- Data-Driven Testing Framework: nesta abordagem é possível armazenar os dados que serão testados em uma unidade externa, dessa forma separando os dados de teste do script que será executado, facilitando a inclusão de testes ao longo do projeto e sua manutenção;
- Keyword-Driven Testing Framework: com um abordagem semelhante à data-driven testing, os frameworks que usam desse método também são capazes de separar seus dados de teste em uma unidade externa, mantendo isolado o conjunto de códigos. Nesse contexto, os códigos são invocados por meio de keywords, podendo ser reaproveitados em outros cenários de teste.
- Hybrid Testing Framework: Os frameworks de automação que usam dessa abordagem híbrida combinam as abordagens de data-driven e keyword-driven testing, permitindo maior nível de abstração e produtividade em seus scripts.

- Linear Scripting Framework: Também conhecida como framework de gravação e reprodução, nessa abordagem o framework conta com a construção do script baseado na gravação de ações manuais realizadas pelo testador. O script, portanto, é resultado das interações do testador que simulam os casos de testes no navegador, gravadas pelo framework para reproduzi-las no futuro. Nesta abordagem, a construção e manutenção dos testes a longo prazo são dificultados pelas limitações inerentes da técnica.
- Module-Based Testing Framework: Nesta abordagem, como o nome sugere, o framework de automação permite a criação de vários módulos, isolando assim os scripts, executados conforme combinação desses módulos. Seu aspecto modular favorece a produtividade e manutenção dos testes.

3. AUTOMAÇÃO DE TESTES COM ROBOT FRAMEWORK

Desenvolvido pela empresa Nokia Siemens Networks em 2005, o framework de automação Robot Framework é fruto da tese de mestrado de Pekka Klärcks publicada no mesmo ano. Baseada em Python, apresenta-se como ferramenta para teste e automação de processos de software e reconhecido como framework de automação genérico devido sua versatilidade em automatizar testes para aplicações mobile, web, APIs e desktop.

Com sua versão 2.0 publicada em código aberto em 2008, o framework passa a ganhar popularidade destacando-se pela sua compatibilidade, conquistando comunidade assídua de usuário. Um dos aspectos que justifica sua notoriedade é sua característica de multiplataforma, considerando que o framework é baseado na linguagem de programação Python, herdando assim sua habilidade de implementação em qualquer plataforma (MONTEIRO, 2020).

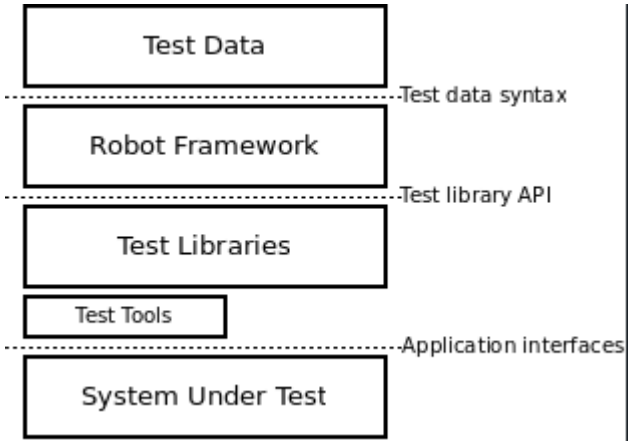
Mantido atualmente pela Robot Framework Foundation, organização sem fins lucrativos, o framework conta com patrocinadores formados por empresas e organizações, garantindo sua versão gratuita e desenvolvimento ativo. Conta ainda com conferência anual própria chamada Robocon, com participantes presencias e virtuais que compartilham de novas funcionalidades e usos do framework de testes.

Sua estrutura tabular e escrita de testes usando palavras-chave fazem do Robot Framework uma ferramenta com baixa barreira de aprendizado, facilitando seu uso por pessoas com pouco conhecimento em linguagens de programação. Conta ainda com bibliotecas nativas, extensíveis e customizáveis, cumprindo seu propósito de abranger amplamente a automação de diversas aplicações. Não obstante, pode ter bibliotecas implementadas em Python, Java ou outras linguagens (ROBOT FRAMEWORK FOUNDATION, 2022).

3.1 Arquitetura

O Robot Framework, por se tratar um framework de automação genérico, está livre de qualquer dependência de outras aplicações ou linguagens de programação. Sua arquitetura tem uma abordagem tabular, representada pela FIGURA 7.

FIGURA 7 - Arquitetura do Robot Framework



Fonte: Robot Framework User Guide, 2023.

Sua estrutura tabular também é aplicada na construção dos dados do testes, o que facilita a sua criação e manutenção. O processo de execução é iniciado pelo processamento dos dados do teste informado, executa-os e por fim gera relatório com os resultados. O script executado não tem conexão direta com o sistema testado; é intermediado pelas bibliotecas invocadas pelo teste. As bibliotecas podem fazer a interface diretamente com a aplicação ou fazer uso de outras ferramentas, como drivers (ROBOT FRAMEWORK, 2023).

3.2 Bibliotecas

Para desenvolver scripts em Robot Framework, o testador pode fazer uso de bibliotecas internas do framework ou ferramentas externas, que se encontram em pacotes que podem ser baixados e implementados no projeto. Não obstante, o testador pode criar bibliotecas próprias para atender a necessidades não supridas pelas bibliotecas disponíveis (MONTEIRO, 2020). Cada biblioteca compreende um conjunto de métodos em Python que são invocados no scripts de teste por meio de keywords. Essa abordagem é uma das características que facilita o processo de criação de testes, pois atenua a responsabilidade do testador quanto ao desenvolvimento de código; sua preocupação é focada em escrever testes que sejam descritivos, apontando o que farão as keywords e não como farão (STANISLAV e ZELJKO, 2011).

As FIGURAS 8 e 9 apresentam o processo de importação e uso de bibliotecas no framework:

FIGURA 8 - Exemplo de importação da biblioteca SeleniumLibrary

```
*** Settings ***
Documentation     Implementação de bibliotecas no Robot Framework
Library           SeleniumLibrary

Load in Interactive Console
*** Variables ***
${BROWSER}     chrome
${URL}         https://login.live.com/

*** Test Cases ***
Run | Debug | Run in Interactive Console
Abrir Navegador Chrome
    Abrir Tela Login

*** Keywords ***
Load in Interactive Console
Abrir Tela Login
    Open Browser  ${URL}  ${BROWSER}
```

Fonte: Autora. 2023.

FIGURA 9 - Exemplo de importação de biblioteca própria

```
*** Settings ***
Documentation     Implementação de bibliotecas no Robot Framework
Library           MinhaBiblioteca.py

*** Test Cases ***
Run | Debug | Run in Interactive Console
Realizar Consulta
    Tratar Dados Cliente

*** Keywords ***
Load in Interactive Console
Tratar Dados Cliente
    Descriptografar Dados  JK3UEFD4  FR55K9F
```

Fonte: Autora. 2023.

A FIGURA 8 exemplifica a importação da biblioteca externa SeleniumLibrary, uma das bibliotecas mais utilizadas do framework para testes Web (MONTEIRO, 2020). Seu uso requer instalação prévia usando o pip, gerenciador de pacotes do Python. A FIGURA 9 apresenta a importação de biblioteca criada pela autora, escrita

em Python e salva no diretório do projeto, referenciada no script para permitir sua execução.

3.3 Editores para construção de scripts de testes

Existem alguns editores disponíveis no mercado para desenvolver scripts em Robot Framework. Cada editor apresenta suas vantagens, sendo de escolha do testador qual a melhor ferramenta. A seguir, relação de alguns dos editores mais conhecidos, expondo seus recursos e benefícios:

- Visual Studio Code: editor construído pela Microsoft e em código aberto, essa ferramenta conta com alguns plugins para uso do framework, destacando-se a Robot Framework Language Server. Além de suportar a sintaxe do framework, fornece ferramentas como depuração de testes, autocompletar e sugestões de códigos;
- PyCharm: editor desenvolvido pela JetBrains como IDE voltada para o desenvolvimento em Python e Django e tem como opção de desenvolvimento em Robot Framework o plugin IntelliBot. Apresenta os mesmo benefícios que o Visual Studio Code, diferenciando-se por entregar uma IDE completa e com recursos mais robustos enquanto ambiente de desenvolvimento;
- RIDE (Robot Framework IDE): mantido pela Robot Framework Foundation, o RIDE é um ambiente de desenvolvimento em código aberto construído para criação e execução de testes do framework. Oferecendo os mesmos recursos disponíveis nos outros editores, o RIDE conta também com ferramentas de assistência para criação de casos de teste, além de fornecer um ambiente de desenvolvimento dedicado ao framework.

3.4 Métodos de escrita de testes com Robot Framework

Assim como demais frameworks, o RF oferece diferentes abordagens de escrita de scripts. A definição da melhor estratégia de desenvolvimento dos testes requer análise macro dos componentes que envolvem tanto o framework quanto o software a ser testado. Aspectos como arquitetura do software devem ser considerados, de maneira que os testes possam cobrir amplamente os casos de usos mapeados para validação com propriedade (BERNARDO, 2011). A abordagem

inadequada pode comprometer questões como manutenibilidade e escalabilidade do script a longo prazo, considerando que os testes devem aderir às mudanças constantes e inerentes de projetos de software.

As principais abordagens de escritas utilizadas pelo RF são Keyword-Driven, Data-Driven e Gherkin. Ademais, o framework também aceita a abordagem híbrida, permitindo ao testador combinar abordagens conforme necessidade.

- Keyword-Driven: esta abordagem utiliza de palavras-chave para emular ações e/ou etapas de um caso de uso. Conforme arquitetura do framework, essas palavras-chaves são escritas em linguagem natural e são reservadas à seção de *Test Cases.* Tais palavras-chaves são implementadas na seção *Keywords* em sequência, representando os passos descrito na seção de testes, conforme ilustra a FIGURA 10:

FIGURA 10 - Abordagem Keyword-Driven

```
*** Test Cases ***
Run | Debug | Run in Interactive Console
Validar Login
    Acessar página de login
    Preencher campo de usuário
    Clicar no botão de proximo
    Preencher campo de senha
    Clicar no botão de entrar

*** Keywords ***
Load in Interactive Console
Acessar página de login
    Set Selenium Speed    0.5 s
    Open Browser  ${URL_LOGIN}  ${BROWSER}

Load in Interactive Console
Preencher campo de usuário
   Input Text    //input[contains(@type,'email')]    teste@mail.com

Load in Interactive Console
Clicar no botão de proximo
    Click Element    //input[contains(@type,'submit')]

Load in Interactive Console
Preencher campo de senha
    Input Text    //input[contains(@name,'passwd')]   pswd123

Load in Interactive Console
Clicar no botão de entrar
    Click Element    //input[contains(@type,'submit')]
```

Fonte: Autora. 2023.

- Data-Driven: esta abordagem permite o teste massivo de dados externos, como planilhas Excel ou CSV, executando o mesmo script com diferentes massas de dados. Na seção *Test Case* é definido o cenário e argumentos que serão recebidos da massa de dados externas. Em *Keywords*, tais argumentos são usados para realizar o teste. A FIGURA 11 exemplifica essa abordagem:

FIGURA 11 - Abordagem Data-Driven

```
*** Settings ***
Library           DataDriver
Library           SeleniumLibrary
Test Template     Validar Login

Load in Interactive Console
*** Variables ***
${BROWSER}      chrome
${URL_LOGIN}    https://login.live.com/

*** Test Cases ***
Run | Debug | Run in Interactive Console
Testar login com o user ${email} e senha ${password}

*** Keywords ***
Load in Interactive Console
Validar Login
    [Arguments]    ${email}    ${password}
    Set Selenium Speed    0.5 s
    Open Browser      ${URL_LOGIN}  ${BROWSER}
    Input Text        //input[contains(@type,'email')]    ${email}
    Click Element     //input[contains(@type,'submit')]
    Input Text        //input[contains(@name,'passwd')]   ${password}
    Click Element     //input[contains(@type,'submit')]

```

Fonte: Autora. 2023.

- Gherkhin: utilizando da sintaxe Gherkin, essa abordagem permite descrever casos de testes usando linguagem natural. Abordagem comumente aplicada em testes de comportamento, adotando as palavras reservadas *Given*, *When, Then, And* para descrever os passos do teste na seção de *Test Case,* conforme apresenta a FIGURA 12:

FIGURA 12 - Aborgadem Gherkin

```
4   *** Variables ***
5   ${BROWSER}      chrome
6   ${URL_LOGIN}    https://login.live.com/
7
8
9   *** Test Cases ***
    Run | Debug | Run in Interactive Console
10  Validate Login
11      When the user accesses the login page
12      Then input the e-mail
13      When click in next
14      Then input the password
15      When click in sign in
16
17  *** Keywords ***
    Load in Interactive Console
18  the user accesses the login page
19      Set Selenium Speed    0.5 s
20      Open Browser  ${URL_LOGIN}  ${BROWSER}
21
    Load in Interactive Console
22  input the e-mail
23      Input Text    //input[contains(@type,'email')]    teste@mail.com
24
    Load in Interactive Console
25  click in next
26     Click Element    //input[contains(@type,'submit')]
27
    Load in Interactive Console
28  input the password
29      Input Text    //input[contains(@name,'passwd')]   pswd123
30
    Load in Interactive Console
31  click in sign in
32      Click Element    //input[contains(@type,'submit')]
33
```

Fonte: Autora. 2023.

Na FIGURA 13 são apresentadas algumas das vantagens e desvantagens encontradas em cada técnica:

FIGURA 13 - Métodos de Escrita de Scripts

Abordagem	Vantagens	Desvantagens
Keyword-Driven	Fácil leitura e interpretação do script Possibilidade de reutilizaçao das palavras chaves em outros scripts de teste Recomendado para testes extensos e complexos	Requer maior tempo de desenvolvimento Necessita de maior supervisão para manter padrões e nomenclaturas nos scripts conforme complexidade e extensão dos testes
Data-Driven	Permite testar combinações de múltiplas fontes externas de dados sem criar vários cenários Ganho em manutenção do teste por ser mantido em um único arquivo	Dificuldades de configuração a depender da biblioteca utilizada Pode apresentar dificuldades de compreensão para iniciantes pouco familiarizados com o conceito de testes Data-Driven
Gherkin	Facilita o processo de escrita por utilizar linguagem natural Permite a descrição clara dos cenários e requisitos do teste	Requer maior tempo de desenvolvimento seguindo a sintaxe Gherkin Barreira da linguagem com palavras reservadas em língua inglesa, dificultando sua escrita em outros idiomas

Fonte: Elaborado pela autora (2023), baseada em Marco Ruben Sobral Monteiro, 2020

Evidencia-se a distinção entre as técnicas de escrita possíveis no RF, corroborando a necessidade de avaliação prévia sobre a abordagem mais adequada. Seus recursos e inconveniências devem ser considerados já nas etapas iniciais do desenvolvimento de testes (IBM, 2022).

4. ASPECTOS DE QUALIDADE EM AUTOMAÇÃO DE TESTES COM ROBOT FRAMEWORK

A criação de scripts automação de testes é uma atividade ampla e com alta expectativa de empresas e organizações. A promessa de mensurar a qualidade de aplicações com qualidade, rapidez e competência pressiona equipes de QA e de Engenharia de Software a adotar estratégias que cumpram esse compromisso. Ademais, conforme abrangências de testes e complexidade aumentam, automações de testes são orquestrados como projetos de software, apartados da aplicação principal testada. Portanto, a criteriosidade e padrões de qualidade são reforçados nesse contexto.

Projetos de testes automatizados de qualidade caracterizam-se por suprimir aspectos que inibem sua capacidade de agregar valor à organização enquanto validadores de software. A adoção de boas práticas e padrões de arquitetura apresentam-se como ferramentas que auxiliam no planejamento e construção dos testes automatizados, elevando sua escalabilidade e longevidade.

Os demais tópicos deste capítulo expõem as características do Robot Framework e as questões que o cercam, abordando os aspectos de qualidade que se apresentam como desafio à construção de testes automatizados com o uso do framework. Manutenção, confiabilidade, capacidade de feedback e outros aspectos são analisados como instrumento a essa reflexão.

4.1 Estrutura do Código

A criação de casos de testes usando RF requer conhecimento prévio de sua hierarquia estrutural, de maneira a identificar as responsabilidades desempenhadas por cada camada dessa estrutura. A dominância de tais aspectos viabiliza a criação de testes robustos pois seguem a metodologia adotada pelo framework, facilitando o processo de leitura e execução dos testes.

O desenvolvimento do projeto de testes deve considerar inicialmente a organização dos arquivos e diretórios, seguindo arquitetura hierárquica orientada a seções:

- Settings: Seção dedicada a informar dados de configuração do teste e outras informações gerais. São incluídas informações como documentação para explicação geral sobre os testes e seu propósito, as bibliotecas que serão

usadas e palavras-chave reservadas para definir testes específicos que serão executados por padrão no início e fim da suíte de teste, respectivamente, *Suite Setup* e *Suite Teardown.* A FIGURA 14 exemplifica sua estrutura:

FIGURA 14 - Seção Settings

```
*** Settings ***
Documentation     Implementação de bibliotecas no Robot Framework
Library           SeleniumLibrary
Suite Setup       Open Browser
Suite Teardown    Close Browser
```

Fonte: Autora. 2023.

- Variables: Seção opcional e reservada para definição de variáveis com escopo global, utilizando a sintaxe ${STR}, conforme ilustra FIGURA 15:

FIGURA 15 - Seção Variables

```
*** Variables ***
${BROWSER}      chrome
${URL_LOGIN}    https://login.live.com/
```

Fonte: Autora. 2023.

- Test Cases: Nesta seção são definidos os casos de uso dos testes, por boa prática criada em um arquivo separado com a extensão .robot. A definição dos casos de testes neste arquivo é considerada pelo framework como uma suíte de testes, pois apenas neste arquivo se encontram todos os cenários testados pela ferramenta. Este arquivo, no contexto hierárquico, é visto pelo framework como a camada de mais alto nível do projeto, considerando que este será o arquivo chamado para interpretação e execução do teste. A FIGURA 16 exemplifica sua estrutura:

FIGURA 16 - Seção Test Cases

```
*** Test Cases ***
Run | Debug | Run in Interactive Console
Validar Login
    Acessar página de login
    Preencher campo de usuário
    Clicar no botão de proximo
    Preencher campo de senha
    Clicar no botão de entrar
```

Fonte: Autora. 2023.

- Keywords: Seção responsável em receber todas as keywords descritas na seção Test Cases e implementá-las com o uso das bibliotecas importadas no projeto. É a seção mais complexa, pois é neste contexto que serão descritos os passos a serem executados pelo script. Entretanto, é uma seção opcional, pois as keywords criadas na seção de Test Cases podem conter seus testes correspondentes. Porém, a depender da extensão e complexidade dos testes, é aconselhável o desenvolvimento dessa seção em um arquivo separado, de maneira a desacoplar os casos de uso de seus testes, facilitando sua manutenção. A FIGURA 17 demonstra essa seção:

FIGURA 17 - Seção Keywords

```
*** Keywords ***
Load in Interactive Console
Acessar página de login
    Set Selenium Speed    0.5 s
    Open Browser  ${URL_LOGIN}  ${BROWSER}

Load in Interactive Console
Preencher campo de usuário
   Input Text    //input[contains(@type,'email')]    teste@mail.com
```

Fonte: Autora. 2023.

Um aspecto do framework que deve ser considerado é seu processo de análise de dados do teste, que segue a sintaxe de segregar as palavras-chave de seus argumentos. Para reconhecer e classificar os parâmetros é necessário que palavras-chave e argumentos tenham o distanciamento entre si de pelo menos dois espaços, reforçando sua estrutura tabular a nível de escrita do script.

4.2 Manutenção

Como tarefa inerente ao processo de automação de testes, aspectos de manutenibilidade são mensurados pela facilidade do script de ser adequado, expandido ou melhorado conforme avanços do projeto. Neste contexto, preza-se pela capacidade do teste de se adequar às novas regras de negócio e se escalável à inclusão de novos casos de testes.

A manutenibilidade e escalabilidade de testes podem ser alcançadas por meio de boas práticas de código, adoção de design patterns e outras medidas reconhecidas

no âmbito da Engenharia de Software e aplicáveis à área de Quaility Assurance. A adoção dessas medidas é indispensável para ganho de produtividade e menor esforço às tarefa de manutenção de scripts.

Boas práticas e padrões de projetos

O desenvolvimento de scripts de automação requer a preocupação de aspectos gerais de boas práticas de codificação, semelhante ao processo de desenvolvimento de softwares tradicionais. Questões como declaração adequada de variáveis, independência entre as ações e comunicação clara e descritiva dos casos de testes são alguns dos pontos considerados quando desenvolvidos testes de automação.

O Robot Framework apresenta algumas diretrizes e boas práticas descritas em sua documentação oficial e compartilhadas por seu idealizador, Pekka Klärck:

- Nomenclatura:
    - Suíte de Testes: aconselha que o nome das suítes seja claro e descritivos. Podem ser longos, mas não muito extensos para não comprometer o gerenciamento de arquivos.
    - Casos de Teste: seguem o mesmo preceito da suíte de teste enquanto sua nomeação clara e objetiva.
    - Palavras-chave: também aconselhado a clareza e objetividade para sua nomeação. Ademais, devem ser claras em descrever o que ela faz, e não como fará determinada ação.
    - Variáveis: nomeação clara, mas não muito longa.
        - Variáveis locais: nomeadas com letra minúscula, com acesso a apenas determinado escopo de teste
        - Variáveis globais, suíte ou nível de teste: nomeadas com letra maiúscula, cada uma com seu escopo definido conforme nível correspondente.

A definição de padrões de projeto é uma alternativa que apresenta soluções inerentes ao processo de desenvolvimento de softwares (ERICH et al., 1995), passível de aplicação em outras áreas tecnológicas, incluindo a automação de testes. Segundo Bernardo (2011, p.55), “ajudam a criar testes de qualidade típicos para cenários rotineiros em testes de software”.

Testes automatizados em Robot Framework podem ter diversas abordagens, considerando sua validação em diferentes tipos de aplicações. Padrões de projetos nesse contexto podem ser definidos pela equipe de QA nas fases iniciais do projeto, considerando questões como escalabilidade e manutenibilidade. Em projetos que envolvem testes de interface é possível aplicar Page Object Model (POM), que consiste em separar os elementos e ações nesses elementos de cada página Web testada, isolando-os como estratégia de manutenção.

4.3 Performance e confiabilidade na automação de testes

Existem grandes expectativas sob os testes automatizados no tocante à sua confiabilidade, na qualidade e consistência dos resultados obtidos de suas execuções. Para manter sua excelência e agregar valor à sua atividade, os testes devem mapear situações esperadas, possíveis e excêntricas, de maneira a abranger todos os cenários possíveis do software afim de detectar eventuais anomalias e, não obstante, reforçar a confiabilidade do sistema (BERNARDO, 2011).

O Robot Framework apresenta abordagens próprias que permitem desenvolver testes estáveis e robustos, elevando sua confiabilidade enquanto validador, gerando assim resultados consistentes. Técnicas de esperas, tentativas controladas e execuções paralelas são algumas dessas abordagens, descritas nas seções seguintes.

Verificações e esperas com o uso de Keywords “Wait”

Testes automatizados caracterizam-se pela sua agilidade enquanto executores. Entretanto, validações como testes de interface necessitam incluir estratégias de espera que possam aguardar pelo carregamento de determinados elementos e estados em tela para prosseguir com os testes. Nesse contexto, o Robot Framework apresenta duas bibliotecas que entregam keywords iniciadas com o termo *Wait* que auxiliam enquanto mecanismos de espera e tentativa contínua:

- BuilIn: Incluída na relação de bibliotecas padrão do framework, a BuiltIn fornece conjunto de keywords gerais e comumente usadas. Nessa relação de keywords encontra-se a *Wait Until Keyword Succeeds*, que tem como objetivo executar dada palavra-chave e, em caso de falha, realizar nova tentativa. Essa

keyword recebe os parâmetros de quantidade de tentativa e intervalo de tempo entre elas, conforme exemplo da FIGURA 18:

FIGURA 18 - Exemplo de Espera: Wait Until Keyword Succeeds

```
Preencher campo de usuário
   ${variable}=    Wait Until Keyword Succeeds  3x  1s
   ...    Input Text    //input[contains(@type,'email')]    teste@mail.com
```

Fonte: Autora. 2023.

- SeleniumLibrary: Utilizada para testes de interface Web, essa biblioteca oferece uma relação de keywords iniciadas com *Wait* como auxílio estratégico ao carregamento de elementos da página. Permite a espera de carregamento de elementos específicos da página, visibilidade desses elementos em tela e outras abordagens. Na FIGURA 19 é demonstrado o uso de uma dessas palavras-chave:

FIGURA 19 - Exemplo de Espera: Wait Until Element Is Visible

```
input the e-mail
    Wait Until Element Is Visible    //input[contains(@type,'email')]
    Input Text    //input[contains(@type,'email')]    teste@mail.com
```

Fonte: Autora. 2023.

A diferença entre o uso das bibliotecas está em seu escopo de atuação, considerando que a SeleniumLibrary é dedicada a testes Web, seu uso de espera está condicionado a esse âmbito de validação, enquanto a BuiltIn, por ser genérica, pode ter sua keyword de espera explícita usada em demais campos de validação.

Execução paralela

A depender de sua densidade e complexidade, a execução dos testes pode levar muitas horas, comprometendo o cronograma apertado de projetos de software. Alguns frameworks permitem a execução paralela de seus testes como alternativa, otimizando seu tempo de execução.

O Robot Framework oferece como opção o Pabot, executor de testes em paralelo desenvolvido exclusivamente para o framework. Permite a execução de testes paralelos por meio de processos em uma única máquina. A quantidade de

processos pode ser informada via linha de comando, porém seu valor está limitado à capacidade computacional da máquina. Por padrão, o Pabot realiza a divisão da execuções a nível de suíte de testes, executando cada teste do suíte sequencialmente. A ferramenta apresenta ainda outra opções, como paralelismo a nível de teste de um mesmo suíte, execução de mais de uma suíte e outras divisões de execução.

O paralelismo de testes caracteriza-se como complexo, pois depende de fatores como infraestrutura e capacidade computacional da máquina ou máquinas utilizadas para comportar os testes em paralelo. Não obstante, cabe a revisão dos scripts utilizados, analisando se o paralelismo pode comprometer o desempenho e funcionamento do teste. Dados de massa e acessos simultâneos ao software são exemplos que podem afetar sua usabilidade.

4.4 Capacidade de Feedback

Um dos aspectos relevantes no uso de testes automatizados é sua capacidade de feedback. A geração de relatórios com os resultados obtidos dos testes serve de insumo para as equipes de QA e de desenvolvimento observarem a qualidade do software testado. Entretanto, conformem evoluem os testes automatizados, maior a necessidade de supervisão e análise desses resultados.

A criticidade em analisar o resultado das execuções está, principalmente, em avaliar os motivos das falhas indicadas pelos testes. O apontamento de uma falha pode não ser necessariamente um problema do software e sim uma falha do próprio script. Outra possibilidade a ser analisada é identificar se o erro é resultado de alguma instabilidade de infraestrutura que comprometa saúde do ambiente em que o software é testado.

Sendo a análise e monitoramento dos testes uma etapa extensa e complexa, uma alternativa a atenuar esse trabalho é potencializar a capacidade do projeto de testes em fornecer relatórios com subsídios e dados suficientes para análise crítica dos resultados que exprime. O Robot Framework nesse aspectos oferece opções e níveis de feedback discutidos nos tópicos a seguir.

Geração de Logs

O Robot Framework conta como padrão a geração automática de relatório dos resultados ao final do teste. O framework utiliza de níveis de registro para identificar o resultado de cada teste, que são:

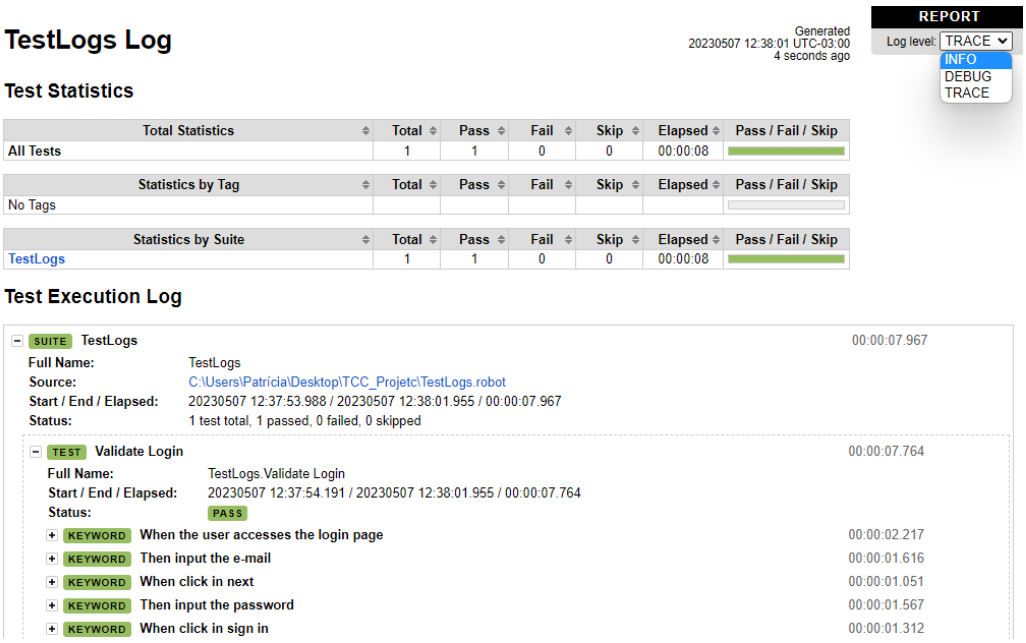
- Fail: Usado para identificar quando uma keyword falha.
- Warn: Usado para identificar algum aviso que não comprometa o resultado do caso de teste.

É possível ainda definir, passando como parâmetro em linha de comando a opção –loglevel (L), o nível de log que será gerado pelo RF, cada qual trazendo informações com grau de detalhamento específico:

- Info: Nível configurado como padrão retornando mensagens comuns, sem muito detalhamento
- Debug: Usado para testes de depuração, detalhando as ações das bibliotecas.
- Trace: Com maior nível de detalhamento, registrando as bibliotecas, palavras-chave com seus parâmetros e retornos.

A FIGURA 20 exemplifica a visualização do relatório, que apresenta um menu no canto superior à direita que permite selecionar o nível de log a ser analisado. Essa opção só é aparente caso o testador tenha passado como comando o registro de log com nível superior a Info.

FIGURA 20 - Exemplo de Registro e Níveis de Log



Fonte: Autora. 2023.

O detalhamento de cada nível pode ser visto na FIGURA 21:

FIGURA 21 - Detalhamento de Log

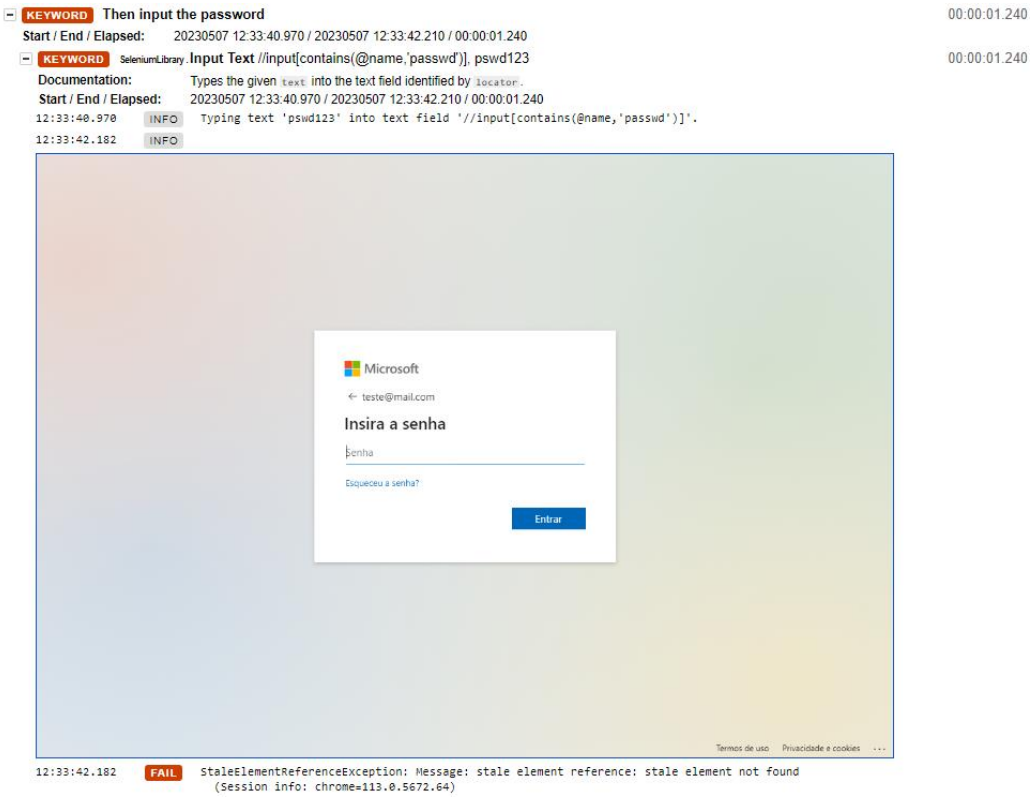
Fonte: Autora. 2023.

Produção de Evidências

A depender do projeto de teste a geração de evidências dos testes executados é necessária, seja para análise futura ou registro formal de sucesso da validação. O Robot Framework conta com duas bibliotecas que podem auxiliar na produção de evidências, descritas abaixo:

- SeleniumLibrary: Esta biblioteca conta como recurso a execução de uma palavra-chave caso um dos testes envolvendo a biblioteca incorra em falha. Por padrão, a palavra-chave acionada nesse cenário é a *Capture Page Screenshot,* realizando um print da tela no momento do erro, salvo no projeto e referenciado no registro de log, conforme FIGURA 22:

FIGURA 22 - Exemplo de Evidência em Cenário de Falha



Fonte: Autora. 2023.

A produção de evidências nesta biblioteca não se limita a cenários de falha, podendo ser aplicada para evidenciar elementos específicos de páginas ou a tela inteira, conforme exemplo da FIGURA 22.

- Screenshot: Reservada para executar ações de captura de tela, essa biblioteca permite registrar prints da máquina onde os testes são executados, expandido seu uso além de navegadores Web. Semelhante a SeleniumLibrary, essa biblioteca pode ser usada para salvar a execução e resultado de cada teste, com a possibilidade de passar parâmetros como diretório que serão armazenadas.

CONCLUSÃO

O desenvolvimento de projetos de automação de testes é cercado de barreiras e desafios que testam sua elegibilidade como ferramenta de valor a empresas e organizações. Sua suscetibilidade a erros, somada a condução de atividades que fogem às boas práticas de desenvolvimento são alguns dos riscos que devem ser ponderados no momento de concepção dos projetos.

A convivência intrínseca com a tecnologia fomenta a expansão e importância da área de Quality Assurance e suas práticas de teste de software. Em concomitante, crescem as expectativas de sua aplicabilidade e retorno de valor às organizações, servindo de reguladora estratégica de seus produtos e serviços digitais.

A garantia de qualidade, robustez e manutenibilidade de projetos de testes automatizados foram aspectos amplamente discutidos neste trabalho, ressaltando que a observância dessas questões é fator determinante à qualidade dos testes automatizados.

SUGESTÃO PARA FUTUROS TRABALHOS

Recomenda-se para trabalhos futuros o estudo de outras aplicações do Robot Framework, como suas habilidades de integração a ferramentas como Jenkins e GitHub Actions para desenvolvimento de pipelines Continuous Integration. Ademais, o estudo de novas bibliotecas como a Browser e sua versatilidade multinavegador para testes Web.

REFERÊNCIAS BIBLIOGRÁFICAS

BADKAR, Akshay. Popular Test Automation Frameworks. **BrowserStack**, 2023. Disponível em: https://www.browserstack.com/guide/best-test-automation-frameworks. Acesso em 01 de mai. de 2023.

BASTOS, Aderson; RIOS, Emerson; CRISTALLI, Ricardo; MOREIRA, Trayahú. **Base de conhecimento de teste de software**. 2ª edição. São Paulo: Martins, 2007.

BERNARDO, Paulo Cheque; KON, Fabio. A Importância dos Testes Automatizados. **Engenharia de Software Magazine**, 1(3), pp. 54-57. Universidade de São Paulo, 2008. Disponível em: https://www.ime.usp.br/~kon/papers/EngSoftMagazine-IntroducaoTestes.pdf. Acesso em 06 de nov. de 2022.

BERNARDO, Paulo Cheque. Padrões de testes automatizados. **Instituto de Matemática e Estatística da Universidade De São Paulo**, 2011. Disponível em: https://www.teses.usp.br/teses/disponiveis/45/45134/tde-02042012-120707/publico/TestesAutomatizados_PauloCheque_Dissertacao.pdf. Acesso em 06 de nov. de 2022.

BIANCHI, Tiago. Percentage of mobile device website traffic worldwide from 1st quarter 2015 to 4th quarter 2022. **Statista**, 2023. Disponível em: https://www.statista.com/statistics/277125/share-of-website-traffic-coming-from-mobile-devices/. Acesso em 01 de mai. de 2023.

CHINNASWAMY, C N; NAGENDRA Mandara; Sreenivas, T H. Robot Framework: A boon for Automation. **International Journal of Scientific Development and Research (IJSDR)**, 2018. Disponível em: https://www.ijsdr.org/papers/IJSDR1811080.pdf. Acesso em 06 de nov. de 2022.

DUVAL, Paul M**. Continuous Integration: Improving Software Quality and Reducing Risk**.7ª edição. Boston: Addison-Wesley Professional, 2007.

ERICH Gamma, Richard Helm, Ralph Johnson, and John Vlissides. **Design Patterns – Elements of Reusable Object-Oriented Software**. Professional Computing Series. Addison-Wesley, 1995.

FONSECA, Maria Adriana Neto. Desenvolvimento de testes automatizados para backend. **Universidade Nova de Lisboa**, 2021. Disponível em: https://run.unl.pt/handle/10362/120492. Acesso em 06 de nov. de 2022.

GAIDARGI, Juliana. Tudo sobre teste de aceitação. **Infonova**, 2021. Disponível em: https://www.infonova.com.br/tutoriais/tudo-sobre-teste-de-aceitacao/. Acesso em 14 de abr. de 2023.

HAMILTON, Thomas. Integration Testing: What is, Types with Example. **Guru99**, 2023. Disponível em: https://www.guru99.com/integration-testing.html. Acesso em 14 de abr. de 2023.

HAMILTON, Thomas. Test Automation Framework: What is, Architecture & Types. **Guru99**, 2023. Disponível em: https://www.guru99.com/test-automation-framework.html. Acesso em 01 de mai. de 2023.

HAMILTON, Thomas. Unit Testing Tutorial – What is, Types & Test Example. **Guru99**, 2023. Disponível em: https://www.guru99.com/unit-testing-guide.html. Acesso em 14 abr. 2023.

IBM, **IBM Engineering Test Management versão 7.0.2,** 2022. Disponível em: https://www.ibm.com/docs/pt-br/elms/elm/7.0.2?topic=scenarios-test-management. Acesso em: 21 abr. 2023.

IBM, **IBM Engineering Test Management versão 7.0.0,** 2021. Disponível em: https://www.ibm.com/docs/pt-br/elms/elm/7.0.0?topic=testing-developing-test-plans. Acesso em: 21 abr. 2023.

IBM, **IBM Engineering Test Management versão 7.0.0,** 2021. Disponível em: https://www.ibm.com/docs/pt-br/elms/elm/7.0.0?topic=testing-getting-started-managing-tests. Acesso em: 21 abr. 2023

KHANDELWAL, Abhik. What is Backend Testing? | Backend Testing tools and Types. **Testing Genez**, 2019. Disponível em: https://testinggenez.com/what-is-backend-testing-and-types/. Acesso em 01 de mai. de 2023

LOVELAND, Scott**. Software Testing Techniques: Finding the Defects that Matter**. Ed. Charles River Media, 2005.

MAXIM, B.R e PRESSMAN, R. S. **Engenharia de Software – Uma Abordagem Profissional.** 8ª. ed. Porto Alegre: AMGH Editora Ltda, 2016.

MILI, A.; TCHIER, F. **Software Testing: Concepts and Operations**. Ed. Wiley, 2015.

MOBILE Automation Testing Steps and Process, **Utor,** 2020. Disponível em: https://u-tor.com/topic/mobile-automation-steps. Acesso em 30 de abr. 2023

MONTEIRO, Marco Ruben Sobra. Desenvolvimento de testes automatizados para frontend. **Universidade Nova de Lisboa**, 2020. Disponível em: https://run.unl.pt/bitstream/10362/123471/1/Monteiro_2021.pdf. Acesso em 02 de mai. de 2023.

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

PERCENTAGE of mobile device website traffic worldwide from 1st quarter 2015 to 4th quarter 2022, **Statista**, 2023. Disponível em: https://www.statista.com/statistics/277125/share-of-website-traffic-coming-from-mobile-devices/. Acesso em 01 de mai. de 2023.

ROBOT FRAMEWORK FOUNDATION, **Robot Framework**, 2022. Introduction. Disponível em: https://robotframework.org/. Acesso em 05 de nov. de 2022.

ROBOT FRAMEWORK USER GUIDE. **Robot Framework 6.0.2**, 2023. Disponível em:https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html. Acesso em: 4 maio 2023.

SNYDER, J. T. **A Practical Guide to Testing Object-Oriented Software**. Addison-Wesley, 2002.

SOMMERVILLE, I. **Engenharia de Software**. 9. ed. Pearson, 2011.

STANISLAV, S; ZELJKO, H. Usage of Robot Framework in Automation of Functional Test Regression. **ICSEA 2011 : The Sixth International Conference on Software Engineering Advances**, 2011.

REFERÊNCIAS BIBLIOGRÁFICAS COMPLEMENTARES

BARTIÉ, Alexandre. **Garantia da qualidade de software: adquirindo maturidade organizacional**. Rio de Janeiro: Campus, 2002.

HAMILTON, Thomas. Test Management Process in Software Testing. **Guru99**, 2023. Disponível em: https://www.guru99.com/test-management-phases-a-complete-guide-for-testing-project.html. Acesso em 21 de abr. de 2023.

MANIFESTO ÁGIL. Disponível em: https://manifestoagil.com.br/. Acesso em: 24 abr. 2023.

ROADMAP.SH, **ROADMAP.sh - Developer Roadmaps**, 2023. QA Engineer. Disponível em: https://roadmap.sh/qa. Acesso em 11 de abr. de 2023.