UNIVERSIDADE FEDERAL RURAL DE PERNAMBUCO

Análise de Dados Coletados para a Melhoria de uma Suite de Testes em um site de e-Commerce

---

Relatório Técnico relativo ao Trabalho de Conclusão Curso do Bacharelado em Sistemas de Informação na modalidade Empresa

---

Aluno
Manoela Timossi Lubambo

Orientador
Cleviton Monteiro
Departamento de Estatística e Informática

20 de março de 2024

Manoela Timossi Lubambo

Análise de Dados Coletados para a Melhoria de uma Suite de Testes em um site de e-Commerce

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

Universidade Federal Rural de Pernambuco – UFRPE

Departamento de Estatística e Informática

Curso de Bacharelado em Sistemas de Informação

Orientador: Cleviton Monteiro

Recife

Fevereiro de 2024

Manoela Timossi Lubambo

Análise de Dados Coletados para a Melhoria de uma Suite de Testes em um site de e-Commerce

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

Aprovada em: 8 de Março de 2024.

BANCA EXAMINADORA

Cleviton Monteiro (Orientador)

Departamento de Estatística e Informática

Universidade Federal Rural de Pernambuco

Cleyton Magalhães

Departamento de Estatística e Informática

Universidade Federal Rural de Pernambuco

Resumo

Resumo. A qualidade é importante na Engenharia de Software para que os sistemas mantenham e cumpram os requisitos especificados, sejam confiáveis, eficientes e livres de defeitos. A garantia desta é feita através de uma série de padrões, práticas e processos. Como parte essencial do processo de garantia de qualidade, os testes de software têm o propósito de verificar a conformidade do software com os requisitos funcionais e não funcionais estabelecidos, tais como desempenho, segurança, usabilidade, confiabilidade, entre outros. Eles são conduzidos através da execução do software sob condições controladas, utilizando técnicas e estratégias específicas para detectar problemas e garantir sua correção. Uma de suas diversas abordagens dar-se através da automação de testes. Neste trabalho, é feito um relato detalhado sobre o processo de automatização de uma suite de testes, destacando os desafios enfrentados ao longo deste processo.E, é realizada uma análise minuciosa dos dados coletados referentes a automação e, por meio dessa análise, busca-se identificar a raiz dos problemas relacionados à falta de eficácia da automação apresentada onde são apontadas possíveis melhorias com base nos resultados obtidos, visando otimizar a eficácia do processo de automação de testes.

Abstract. Quality is important in Software Engineering so that systems maintain and meet specified requirements, are reliable, efficient and free from defects. This is guaranteed through a series of standards, practices and processes. As an essential part of the quality assurance process, software testing aims to verify the software's compliance with established functional and non-functional requirements, such as performance, security, usability, reliability, among others. They are conducted by running the software under controlled conditions, using specific techniques and strategies to detect problems and ensure their correction. One of its diverse approaches is through test automation. In this work, a detailed report is made on the process of automating a test suite, highlighting the challenges faced throughout this process. And, a thorough analysis of the data collected regarding automation is carried out and, through this analysis, we seek identify the root of the problems related to the lack of effectiveness of the automation presented, where possible improvements are identified based on the results obtained, aiming to optimize the effectiveness of the test automation process.

Sumário

1 Introdução

Com o avanço dos anos, a tecnologia e os sofwares estão cada dia mais presentes nas nossas rotinas seja facilitando a comunicação, tarefas ou nos proporcionando meios de diversão. Com a transformação digital, a Engenharia de Software tornou-se cada vez mais forte em todo o processo, com a arquitetura dos softwares, segurança, desenvolvimento e qualidade.

A qualidade de Software é uma parte crucial da engenharia de software, que envolve uma série de padrões, práticas e processos para que os softwares atendam os requisitos especificados, sejam confiáveis, eficientes e livres de defeitos. Dentro do contexto de qualidade de software, há a garantia de qualidade e o controle de qualidade. A garantia de qualidade (Quality Assurance) [1] pode ser definida como qualquer processo sistemático que determina se o produto ou serviço cumpre os requisitos especificados. Já o controle de qualidade (QC) é um processo mais restrito, pois se concentra na detecção de erros, ou requisitos perdidos em um produto.

Uma importante etapa na qualidade dos softwares são os Testes de Software. Podem ser definidos [2] como um processo, ou uma série de processos feitos para garantir que o código escrito faça o que foi projetado para, e que, consequentemente esse código impacte o sistema de forma negativa. Existem várias formas de execução e diferentes abordagens para os testes de software. No contexto de sites de *e-commerce*, para prezar pela qualidade, há uma série de testes que se adequam ao processo, como testes *end-to-end*, testes funcionais e testes de regressão por exemplo, sempre prezando pela manutenção e garantia do fluxo. Esses testes podem ser executados de forma automatizada ou manual.

A execução de alguns tipos de testes de forma manual demanda, normalmente, bastante tempo e, por muitas vezes, serem extremamente repetitivas, é recomendado a automação do máximo possível dos testes do sistema [3]. Este trabalho, abordará um projeto em um ambiente de site de *e-commerce* que teve seus testes automatizados em busca de otimização de tempo. Apesar de se provar eficiente, outro problema surgiu: A automação acabou não tendo tanta eficácia quanto o esperado. Assim, foram reunidos dados em relação as últimas 7 sprints do projeto com o objetivo de analisá-los e identificar onde o problema se encontra. Após essa análise e discussão destes dados, foram propostas melhorias para otimizar a automação e os processos do projeto.

1.1 Contexto

Sites de *e-Commerce* são um tipo de plataforma online onde os produtos ou serviços são vendidos e comprados pela internet. Esses sites fornecem uma interface para que os usuários possam navegar pelos produtos, selecionar itens, adicioná-los a um carrinho de compras virtual e, eventualmente, efetuar o pagamento para concluir a transação.

Quando consideramos testes de software no contexto destes sites e ambientes é importante destarcarmos a importância do teste de todo do fluxo *end-to-end* com o principal objetivo de reproduzir o fluxo de compra e as interações do cliente. Vale destacar pontos críticos do site sendo a *Homepage*, *Product Listing Page*, *Product Display Page*, Carrinho e *Checkout* que devem ser os principais focos dos testes, pois, sem o funcionamento devido deles, prejudicará o principal fluxo de fechamento de

pedido e acarretar um prejuízo grande a empresa tanto financeiro quanto de reputação. Para isso, diferentes abordagens podem ser utilizadas e podemos utilizar a pirâmide de testes destacada na figura 1 para ilustrá-las. Ela trata-se de uma forma de ilustrar de forma simplificada os tipos diferentes de testes e seus diferentes níveis. Primeiro,em sua base, temos os testes unitários, que são os testes realizados na menor parte testável de uma aplicação [4]. Em seguida, temos os testes de integração [4], que têm como objetivo testar um conjunto de unidades interagindo entre si. E, no topo da pirâmide [4], temos os testes *end-to-end* que têm como objetivo principal simular o comportamento de um usuário final em nossa aplicação. Dentro deles, temos diferentes tipos de testes, como Testes de Regressão, para verificação que novos recursos não impactem de forma negativa o funcionamento do site, Testes de Funcionalidade para a validação da funcionalidade de algum recurso no fluxo, Testes de Usabilidade para a avaliação da experiência do usuário, facilidade de navegação, e clareza das informações por exemplo.

Dentro do projeto, com a complexidade crescente do planos de testes e do sistema, e, o aumento do volume dos testes, a automação tornou-se crucial para facilitar e otimizar sua execução. Com isso, uma automação foi desenvolvida com o intuito de otimizar tempo e trazer mais eficiência na hora de garantir a qualidade no projeto. Com ela, foi possível acelerar o processo de execução dos testes e aumentar a abrangência e cobertura dos *Test Cases*, além da capacidade de manter a consistência ao realizar testes repetíveis.

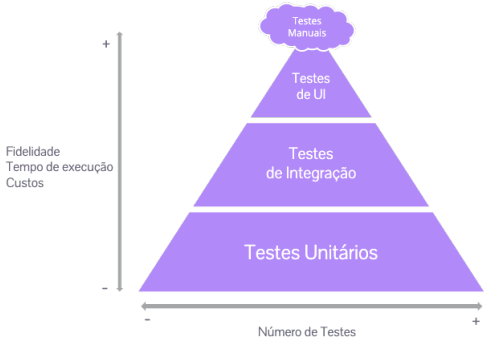

Figura 1: Pirâmide de testes (Dev Community, 2020).

1.2 Problema

O projeto trabalhado deu-se em um ambiente de um site de *e-commerce* voltado para o comércio *B2B* e *B2C*. Neste projeto, foram identificados dois problemas. O primeiro deles reside no considerável volume de testes a serem abrangidos de maneira manual, resultando em uma demanda significativa de tempo. Isto foi resolvido através da automatização destes testes, o que reduziu bastante o seu tempo de suas execuções. A automação cobre o fluxo *end-to-end* de todo o site, com o intuito de reproduzir o fluxo de compra do cliente e fazer com o site estivesse funcionando de forma eficiente e correta como a forma que foi definida e implementada.

No entanto, embora tenha demonstrado eficiência na redução do tempo, a introdução da auto-

mação trouxe consigo uma nova questão. Após a implementação da automatização, os dados coletados revelaram que apenas 10% do total de falhas (*bugs*) estão sendo identificadas pelos testes automatizados. Esses dados fundamentam uma análise aprofundada, visando identificar as razões desta limitação e apontar áreas passíveis de aprimoramento. Essas questões serão detalhadamente exploradas na seção 5, onde será feita uma análise dos dados e propostas melhorias.

1.3 Objetivo

Este trabalho tem como objetivo apresentar o trabalho realizado na empresa no desenvolvimento da Suite de Testes atual do projeto, e, conduzir uma análise dos dados coletados em relação aos diferentes ambientes de testes e abordagens, com o propósito de identificar causas subjacentes e propor melhorias significativas para otimizar a eficácia e abrangência dos testes realizados.

Os objetivos específicos podem ser definidos por:

- Relatar o trabalho de automação de testes realizado, detalhando a arquitetura e os desafios encontrados.
- Analisar a eficácia das abordagens utilizadas nos ambientes.
- Identificar cenários específicos em que cada abordagem se destaca ou apresenta deficiências.
- Apresentar propostas de melhoria para cada abordagem.

2 A empresa e Sua Atuação

O estudo de caso foi realizado em uma empresa de consultoria multinacional com filiais no Brasil, Índia e Estados Unidos, focada no modelo de negócio de e-commerce B2B. A empresa presta diversos tipos de serviços de consultoria separados em três principais focos: Tech, Marketing e Sales. Na parte Tech, seu foco é a implementação de eCommerce focando nas ferramentas de CRM Salesforce, em commercetools, uma plataforma de comércio headless baseada em nuvem, e a plataforma Oracle, que é voltada para criar aplicativos escaláveis orientados a dados executáveis localmente ou em nuvem. Seus principais serviços prestados são implementação de sites de e-commerce tanto B2B como B2C e D2C, suporte, manutenção de websites e micro serviços e, gerência e análise de dados.

O cliente que foi prestado consultoria é uma empresa americana de equipamentos de telecomunicações, líder global em segurança pública e empresarial focada em comunicações de rádio móvel, vídeo segurança e controle de acesso. Seu foco é na produção de rádios bidirecionais e rádios de público para socorristas e forças policiais. Ela também possui pacotes de softwares para centros de comando, mapeamento e vigilância. Sendo um dos principais clientes da consultoria, essa empresa possui cerca de dez times dedicados a diversos tipos de serviços relacionados ao seu site, como time de performace,CPQ, Subscriptions entre outros. Cada time possui PO, Líder Técnico, desenvolvedores, QAs e Scrum Master que utilizam o framework ágil Scrum e Waterfall para gerir as equipes.

3 Referencial Teórico

Nessa seção, abordaremos sobre a prática fundamental de testes de software e destacaremos a importância crítica dela para o desenvolvimento de software e suas diferentes abordagens e tipos disponíveis. Sua variedade permite que as equipes garantam que o software seja robusto, confiável e atenda principalmente os requisitos do cliente mantendo sua qualidade.

3.1 Processo de Qualidade de Software

O processo de qualidade de software [3] pode ser divido em três principais pontos: Planejamento da Qualidade, Garantia da Qualidade, Controle da Qualidade como apresentado na figura 2.

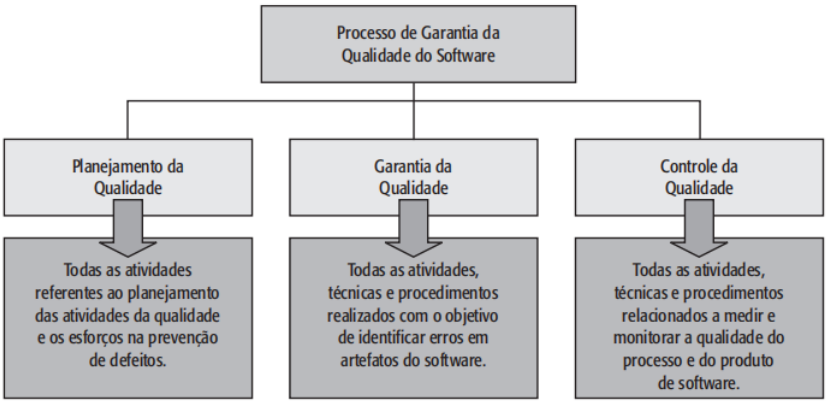

Figura 2: Pilares do Processo de Qualidade (Bartié, 2002).

No processo de planejamento, há a identificação de quais padrões de qualidade são relevantes para o projeto e a determinação de como satisfazê-los. É realizado em paralelo com outros processos de planejamento e tem como produto o Plano da Garantia da Qualidade de Software (SQA Plan – Software Quality Assurance Plan).

O segundo processo é o de garantia da qualidade, que engloba a estruturação, sistematização e execução das atividades que terão como objetivo garantir o adequado desempenho de cada etapa do desenvolvimento, satisfazendo os padrões de qualidade definidos. Nele se enquadram testes de verificação e testes de validação dentro do ciclo de desenvolvimento.

Por último há o processo de controle da qualidade que se concentra no monitoramento e desempenho dos resultados do projeto para determinar se ele está atendendo aos padrões de qualidade no processo de desenvolvimento. É um processo contínuo e sistemático acompanhamento da eficiência do desenvolvimento em diversos pontos de controle, possibilitando às gerências e profissionais envolvidos acompanhar variações de qualidade e promover ações corretivas e preventivas para manter o nível de qualidade desejado. É importante lembrar-se que, devemos garantir a qualidade de todas [3] as etapas do processo de desenvolvimento do software, de forma a garantir que não haja início uma nova fase caso esta ainda não tenha sido bem finalizada e entendida.

3.2 Teste de Software

Para manter a qualidade de um software, os testes são fundamentais, desempenhando um papel direto e crucial na garantia da qualidade. Essas atividades são integradas em diversas fases do ciclo de vida do desenvolvimento, desempenhando um papel importante na verificação das funcionalidades do sistema.

O teste de software representa um conjunto de processos essenciais [2] que visam assegurar que o código desenvolvido realize as funções previamente projetadas, evitando que o código cause falhas ou impacte negativamente o sistema. De acordo com Bartié [3], o teste é um processo sistemático e planejado cujo objetivo único é a identificação de erros. Esses processos são parte integral do desenvolvimento de software, contribuindo para garantir a qualidade, confiabilidade e desempenho da aplicação antes de ser disponibilizada aos usuários. A eficácia dessas execuções controladas depende significativamente da postura do testador [2], que deve identificar erros, problemas de funcionalidade, segurança, entre outros. Nesse sentido, é crucial abordar os testes com a premissa de que o programa contém erros, em vez de testar apenas para confirmar seu funcionamento.

No âmbito dos testes de software, é essencial avaliar a viabilidade de identificar todos os erros em um programa. É comum que essa tarefa seja impraticável e quase sempre impossível [2], devido a restrições de tempo e à limitação da capacidade humana para identificar todos os cenários possíveis. Diante disso, cabe ao QA assumir certas premissas sobre o programa e a forma como os casos de teste serão elaborados.

Para mitigar esses desafios, existem diversas estratégias, técnicas e abordagens de teste que devem ser cuidadosamente planejadas antes da execução. A escolha da técnica e abordagem apropriadas é determinada pelo escopo do que será testado, pelos requisitos do sistema, pelas regras de negócios e pela funcionalidade. Duas das técnicas mais amplamente utilizadas são os Testes de Caixa Preta e os Testes de Caixa Branca. Essas abordagens diferenciadas proporcionam uma análise abrangente e sistêmica, contribuindo para a identificação eficaz de erros e garantindo a qualidade do software.

3.2.1 Teste de Caixa Preta

De forma conceitual, testes de Caixa Preta concentram-se nos requisitos funcionais [5] e na avaliação do software a partir da perspectiva externa, e também são chamados de Testes Funcionais. Este tipo de teste não leva em consideração como o software foi implementado nem o seu código fonte, ou seja, o QA sabe somente o que o software propõe-se a fazer [5].

As principais características dos testes de caixa preta são o foco nas entradas e saídas. Por exemplo, um teste com um *input* de data errada, um CPF errado e verificar como o sistema se comportará nessas situações [5]. Esse tipo de teste também é frequentemente utilizado para verificar se os requisitos estão sendo atendidos como requisitos do cliente, regras de negócio ou casos de uso.

O teste de caixa preta pode ser aplicado em qualquer estágio do ciclo de vida do desenvolvimento de software, desde o início do projeto até a manutenção contínua do software. Isso ajuda a identificar

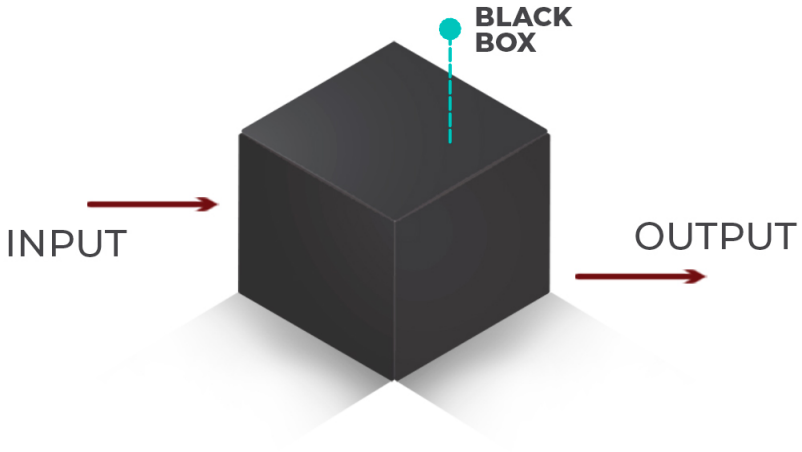

Figura 3: Fluxo de um teste de caixa preta (Base2, 2005).

problemas de usabilidade, requisitos não atendidos e outros problemas que podem surgir durante o uso do software.

3.2.2 Teste de Caixa Branca

Ao contrário do Teste de Caixa Preta, no teste de Caixa Branca há conhecimento do código implementado [5]. Ele também é chamado de Teste Estrutural e nesses testes, o QA tem acesso à implementação do software [2], desta forma os testes podem ser mais pontuais e rigorosos, pois o testador sabe os pontos mais vulneráveis da implementação.

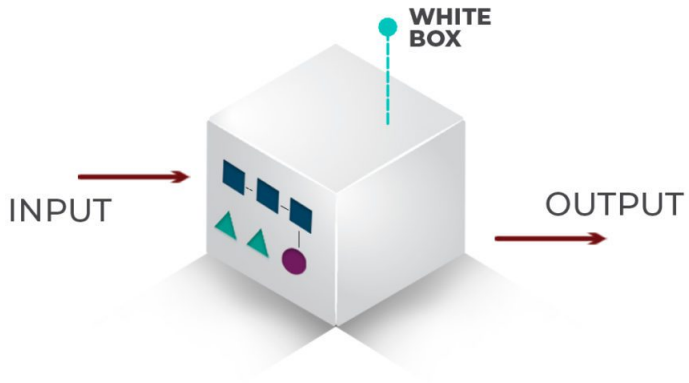

Figura 4: Fluxo de um teste de caixa branca (Base2, 2005).

Com esse tipo de teste é possível encontrar bugs ocultos no código fonte e analisar minuciosamente a estrutura do código, algoritmos utilizados e a lógica interna. Com isso, há alta cobertura do código, garantindo que a maior parte do código-fonte seja testada. Essa abordagem ajuda a identificar erros lógicos, falhas de fluxo de controle e problemas de programação.

Os testes de Caixa Branca são vistos principalmente em Testes unitários, que são testes desenvolvidos para testar unidades individuais do código, como funções ou métodos e, podem ser

desenvolvidos por QAs ou desenvolvedores.

3.3 Testes de Regressão

Testes de Regressão ou Testes regressivos tratam-se da reexecução de um subconjunto (total ou parcial) de testes previamente executados. Seu objetivo é assegurar que as alterações ou inserções de determinados segmentos do produto (durante uma manutenção ou melhorias implementadas) não afetaram outras partes do fluxo [3].

Há dois tipos de testes de regressão: Testes de Regressão total e parcial. O Teste de Regressão total há a execução de todos os testes sem que haja qualquer exceção [3]. Apesar de minimizar risco nesse processo, o custo da sua execução é bastante ampliado. Para esse tipo de teste, é preciso atentar-se ao seu custo e a forma de sua execução, pois, executá-los manualmente requer um investimento de tempo maior em comparação com a execução automatizada.

Já os testes de regressão parcial, há a execução de somente um subconjunto de casos de teste [3]. Nesse subconjunto deve-se identificar uma afinidade de negócio existente entre os casos de teste e as alterações realizadas. Nesse método, é preciso atentar-se ao alto risco de algum caso de teste siginificativo ficar de fora, fragilizando o processo. Abaixo, é possível ver a comparação entre esses dois métodos de testes de regressão e as duas principais abordagens utilizadas: testes automatizados e manuais 5.

	Regressão Total	**Regressão Parcial**
Teste Manual	• baixo risco de não-cobertura • alto custo de execução • execução lenta • reutilização zero • interferências humanas	• alto risco de não cobertura • custo de execução menor • execução lenta • reutilização zero • interferências humanas
Teste Automatizado	• baixo risco de não-cobertura • baixo custo de execução • execução rápida • reutilização total • sem interferências humanas	• alto risco de não-cobertura • menor custo de execução possível • execução mais rápida possível • reutilização total • sem interferências humanas

Figura 5: Comparação entre abordagens de testes de regressão e testes automatizados e manuais (Bartié, 2002).

3.4 Testes Automatizados

A automação de testes é uma ferramenta bastante eficaz para o aprimoramento da qualidade do produto final por conta da sua abrangência e possibilidade de repetições e execuções de vários

cenários em um pequeno intervalo de tempo. O fato da economia do tempo já compensa o esforço utilizado para sua implementação [3].

Ela pode ser definida como a utilização de ferramentas de testes que simula usuários ou atividades humanas de forma a não empregar procedimentos manuais no processo de execução dos testes [3] e é algo crítico para obtermos no processo de garantia da qualidade do software desenvolvido [3].

É recomendado o investimento ao máximo possível na automação dos processos não importando o tipo de abordagem, para evitar [3] o retrabalho, pois toda vez que há implementação de uma nova funcionalidade, outras funcionalidades relacionadas devem ser testadas, o que exige grande esforço de repetição. Para que haja melhoria dos procedimentos manuais, é preciso reduzir o volume de cenários [3], o que reduzirá o nível de cobertura dos testes e aumentando a probabilidade de novos erros.

Nesse tópico, exploraremos as ferramentas utilizadas no dia a dia do projeto para a implementação e manutenção de sua automação.

3.4.1 Ferramentas de automação de Testes: WebDriver.IO e Node.js

A automação de testes consiste no uso de softwares específicos capazes de controlar e gerenciar determinados testes e para sua implementação, são usadas ferramentas de automação de testes. Essas ferramentas auxiliam no desenvolvimento e execução de scripts que são executados automaticamente pela ferramenta. Dentro do projeto, podemos destacar duas principais ferramentas: Webdriver.IO e Node.js.

WebdriverIO é uma framework Open Source de automação de testes e2e (end-to-end), testes unitários e teste de componentes web [6]. Nele é possível rodar testes baseados no WebDriver, Webdriver BiDi, Chrome DevTools e Appium, uma ferramenta de automação de testes mobile. O Webdriver possue suporte para frameworks que utilizam a metodologia BDD/TDD e é possível rodá-lo localmente ou em sistemas cloud como Sauce Labs, Browser Stack, Testing Bot e LambdaTest [6].

Além disso, o Webdriver possui uma estratégia chamada Smart Selector que simplifica a interação com componentes React ou a executação de consultas de selector. Essas interações ocorrem através de um protocolo de automação padronizado que garante que elas se comportem de forma nativa e não sejam apenas emuladas em JavaScript.

Outra ferramenta importante é o Node.js. Ele é software de código aberto, multiplataforma, e que permite a execução de códigos JavaScript fora de um navegador web. Com um tempo de execução de JavaScript assíncrono orientado a eventos, o Node.js foi projetado para construir aplicativos de rede escalonáveis [7].

Quando se trata de automação de testes, Node.js desempenha um papel significativo devido à sua capacidade de executar testes automatizados usando diversas ferramentas e framework end-to-end de automação que permitem simular a interação do usuário com a aplicação, automatizando ações como clicar em botões, preencher formulários e verificar resultados como o Webdriver.io [6] e o Cypress.

3.5 Trabalhos Relacionados

Os softwares estão cada vez mais presentes em nossas vidas com o passar do tempo. Porém, para que haja a funcionalidade plena deles, é necessário que não se tenha erros. Para isso, na Engenharia de Software, há a etapa de Testes de Software.

Para se ter uma implementação de automação de testes eficiente, Kim, Na e Ryoo [8] propõem que primeiramente, haja controle de fluxo da execução, pois dessa forma, há redução na duplicação de código e módulos nas aplicações de teste. Além disso, é defendido a utilização de CI (*Continuous Integration*) seguindo a seguinte estrutura: Ao subir um código ao repositório, o servidor CI detecta as mudanças implementadas no código e executa uma *build* e executa a automação, após isso, é gerado um relatório com as métricas para medir qualidade, examinar práticas e avaliar riscos. Casos erros apareçam, o servidor CI gera um *feedback* e encaminha por e-mail para os membros do projeto. Além disso, eles defendem a utilização de uma BVT (Build de verificação de testes), uma suite de execução de testes em cada nova *build* do produto para verificar de forma geral a qualidade da *build*.

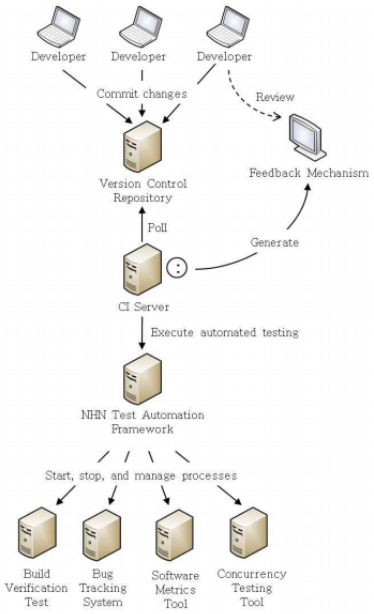

Figura 6: Fluxo de CI (Kim, Na, Ryoo, 2009).

Em questão de arquitetura em ambientes ágeis, Adad, Boehm, Sievers e Wheaton [9], propõem uma arquitetura de testes baseada no desenvolvimento de software visando a melhoria e otimização da automação de testes. Eles defendem que, assim como o desenvolvimento de software, a automação de testes deve dividir-se em camadas. É defendido que, não se deve focar apenas na interface gráfica, um erro bastante cometido em diversos projetos na hora de automatizar, pois isso trata apenas uma parte substancial do sistema. Além disso, é importante destacar outras camadas que geralmente não são focadas pelo time de automação: A camada de Negócio, Dados e *Web Services*.

É importante que o time de automação entenda a lógica do negócio através dos requisitos do sistema para que haja o alcance do objetivo no final do desenvolvimento. Para que a automação obtenha sucesso, é necessário exercitar as variadas *interfaces* para que haja a redução nas lacunas de teste.

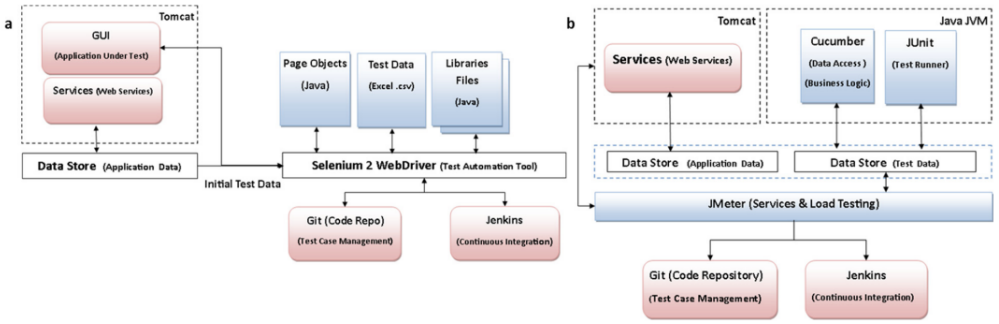

Figura 7: Divisão da arquitetura de testes em camadas (Azad M. M., Barry B., Sivers M., Wheaton M., 20214).

No quesito de dados, o foco deve ser testar não somente a funcionalidade de subir e remover dados de uma base de dados, mas também testar a qualidade e a integridade desses dados. Desde o começo o arquiteto de automação deve participar das discussões sobre a *design* do *schema*. Nessa fase de design, a documentação pode ser revisada de uma perspectiva de qualidade.

Em relação ao *Web Services*, eles podem ser testados de duas formas diferentes, através dos testes funcionais e testes de performance. Nos testes funcionais, o objetivo é validar que o serviço está de fato mostrando os resultados esperados em relação as entradas dadas. Para os testes de performance, a intenção é verificar que o serviço performa como esperado através cargas ou circunstâncias divergentes.

4 Trabalho na empresa

4.1 Problemas Enfrentados

No projeto, os testes eram executados de forma manual em grande maioria e uma pequena parte de forma automatizada. Normalmente cada time possuía um QA alocado e responsável por todo o fluxo de qualidade como execução dos testes, criação de planos de testes, abertura de *bugs* e implementação na automação. No dia a dia, alguns desafios foram enfrentados.

Primeiramente, por haver somente um QA alocado nesse time em específico, o QA acabava sobrecarregado em desempenhar ambas as funções: Testes Manuais e Testes Automatizados. Além disso, não havia conhecimento e visibilidade da arquitetura do projeto como um todo, somente alguns serviços específicos. Os testes automatizados acabavam sendo deixado em segundo plano, pois, os testes manuais eram, geralmente, novas *features* que precisavam seguir o fluxo de forma mais urgente. Para que houvesse otimização desses testes e o QA tivesse mais tempo para executar o seu trabalho com excelência, foi implementada uma suite de testes pelos QAs das diferentes frentes do mesmo projeto.

No quesito automação, o grande desafio foi a quantidade de scripts já desenvolvidos que precisavam de manutenção. Devido a novas implementações no sistema, mudanças a interface acabavam ocorrendo, o que fazia com que elementos do site como botões, barras de pesquisa, localização de

produtos, eram trocados de lugar e acabavam danificando esses scripts. Além disso, outros casos de teste precisavam ser automatizados, o que era preciso atentar-se sempre a documentação, que, algumas vezes não existia, estava incompleta ou defasada e aos *locators* que eram mal identificados.

Após a sua implementação bem sucedida, o trabalho do QA melhorou bastante, porém outro problema surgiu. Apesar de cobrir a maior parte dos *testcases* do site, a automação apesar de eficiente, não se mostrava eficaz. A maioria dos *bugs* achados no ambiente de QA foram encontrados de forma manual. Logo, além da automação desses testes, é preciso identificar a raiz do problema, analisar suas causas para que sejam propostas melhorias para tornar a automação eficaz, pois, é necessário otimizar o desempenho e maximizar os benefícios obtidos na sua implementação.

4.2 Trabalho Realizado

Para os solucionar o problema criado pela demanda e grande volume dos casos de testes, foi realizado um trabalho visando implementar uma automação de testes robusta no projeto. A princípio, na automação desenvolvida, a contribuição foi ativa, iniciando com a escrita dos casos de teste, com foco na criação de novos casos relacionados a dois produtos novos com características específicas. Esses produtos incluíam um sistema de assinatura conectado a um portal do parceiro já existente, além da validação dos novos tipos de assinaturas.

Posteriormente, houve forte participação na implementação dos casos de teste junto com a equipe, com o desenvolvimento de Scripts em Node.js utilizando o framework Webdriver.io, concentrando-se no fluxo completo da loja.

Além disso, houve participação na manutenção dos scripts de automação existentes, sendo papel de todos os QAs ficarem atentos a possíveis necessidades de manutenção no código já escrito. Além de contribuir com a criação de novos scripts de teste, também houve contribuição na manutenção de códigos devido a pequenas mudanças no site em diversos setores.

Além das contribuições mencionadas anteriormente, foi conduzida uma análise detalhada dos dados coletados, priorizando a eficiência da automação, com o intuito de sugerir melhorias. Observou-se que, embora a automação esteja bem estruturada, enfrentou desafios de eficácia em cenários específicos, como evidenciado pelos dados coletados e analisados na seção 5. A discussão sobre a estrutura e implementação da automação será aprofundada na próxima seção, especialmente focada na arquitetura de testes.

4.3 Arquitetura da Suite de Testes

No projeto, o time de QA é responsável por executar diversos tipos de testes. Suas principais atuações são na escrita de casos de testes, execução de testes de performance, execução de testes manuais e de implementação de testes automatizados. Esses testes possuem casos de teste que são divididos entre seções relacionadas ao fluxo do site de e-commerce apresentado na figura 8: *Homepage, Product Listing Page, Product Display Page*, Carrinho, *Checkout* e confirmação do pedido. Cada seção possui casos de teste específicos que cobrem os principais pontos críticos para que o site se mantenha funcional. Como ferramenta para armazenamento é utilizado o TestRail que

é conectado ao JIRA para que haja rastrabilidade dos tickets trabalhados.

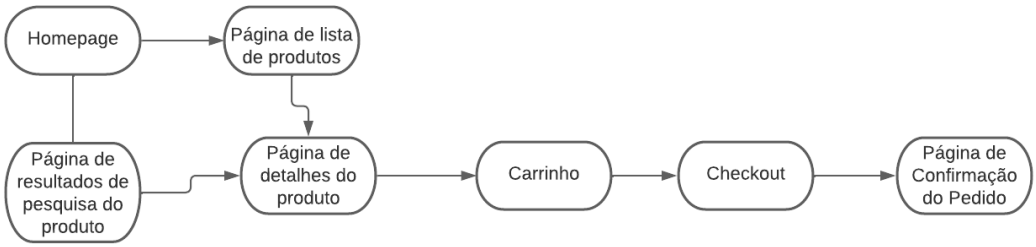

Figura 8: Fluxo do E-commerce (Elaborado pelo autor).

Para esses casos escritos e armazenados, o time de QA implementou e, continua implementando a medida que há necessidade, scripts de testes automatizados. Há cerca de mais de 100 scripts atualmente, que cobrem todo o fluxo do site de ponta a ponta. Esses scripts são escritos em Node.js[7], usando o framework do Webdriver.io para executá-los.

O Webdriver.io [6] é uma framework open source de automação progressiva bastante popular na área de automação de testes, pois, ela pode ser utilizada tanto para automação de aplicações web quanto para aplicação de testes mobile e suporta todos os tipos de browsers. A arquitetura da automação é baseada no padrão Page Object [10] como mostra a figura 9, um padrão bastante popular por melhorar a manutenção de testes e reduzir a duplicação de código.

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;

/**
 * Page Object encapsulates the Sign-in page.
 */
public class SignInPage {
  protected WebDriver driver;

  // <input name="user_name" type="text" value="">
  private By usernameBy = By.name("user_name");
  // <input name="password" type="password" value="">
  private By passwordBy = By.name("password");
  // <input name="sign_in" type="submit" value="SignIn">
  private By signinBy = By.name("sign_in");

  public SignInPage(WebDriver driver){
    this.driver = driver;
     if (!driver.getTitle().equals("Sign In Page")) {
      throw new IllegalStateException("This is not Sign In Page," +
            " current page is: " + driver.getCurrentUrl());
    }
  }

  /**
    * Login as valid user
    *
    * @param userName
    * @param password
    * @return HomePage object
    */
  public HomePage loginValidUser(String userName, String password) {
    driver.findElement(usernameBy).sendKeys(userName);
    driver.findElement(passwordBy).sendKeys(password);
    driver.findElement(signinBy).click();
    return new HomePage(driver);
  }
}
```

Figura 9: Exemplo de código utilizando o padrão Page Object (Selenium, 2024).

A Page Object[10] é uma classe orientada a objetos que serve como interface para uma página do seu AUT (Application Under Test). Os testes, quando precisam interagir com a UI da página, chamam os seus métodos. Sua grande vantagem é que, se houver qualquer alteração na UI da página, os testes (scripts) em si não precisarão ser alterados, apenas o código de dentro dela. Além disso, há clara separação entre o código de testes e os códigos específicos para as páginas, por exemplo, os locators.

Locator é o nome utilizado para códigos que possuem um endereço de um elemento de uma página web. Através dele é possível fazer com que as frameworks de teste encontrem o elemento da página para que haja a interação e execução da ação necessária. Logo, a estrutura da automação é dividida da seguinte forma, há uma pasta para o armazenamento dos locators localizados em cada página do fluxo principal, Page Objects para cada uma delas e os scripts em outra pasta.

Para execução dos testes, é utilizado a ferramenta Jenkins[11], um sistema de servidor de automação de código aberto que possui diversos plugins para suporte de builds, deploy e automação. Essa ferramenta utiliza a prática CI (continuous integration)/CD (continuous delivery) o que nos permite realizar os testes de forma contínua, integrando-os diretamente.

Para a implementação da arquitetura da suíte de testes apresentada, foram enfrentados alguns desafios significativos ao longo de sua execução. O volume considerável de casos de teste, somado à complexidade das interações entre as diferentes seções do site de e-commerce, gerou dificuldades na manutenção e na execução eficiente dos testes automatizados. Para solucionar isso, foi necessário aumentar o número de QAs trabalhando no desenvolvimento dos scripts e seguir um fluxo padrão e principal (da página inicial a finalização do pedido) adotado por um usuário para fechar um pedido seguido pela automação denominado *Happy Flow*. Esse fluxo passa apenas pelos cenários usuais e triviais e utiliza a região *North America* como base. Além disso, as mudanças frequentes na interface, como a introdução de novos elementos ou alterações nos botões, muitas vezes resultavam na quebra dos scripts de automação, exigindo atualizações constantes para garantir sua funcionalidade. Outro problema recorrente foi a falta de identificadores estáveis (locators) para os elementos da página, o que tornava a localização e interação com esses elementos menos confiáveis e propensos a falhas.

Para solucionar esses problemas, foram criados *tickets* no Jira do tipo tarefa para que os desenvolvedores adicionassem *ids* únicos pelo código para que elementos como botões, barras de pesquisa que eram interagidos durante a execução fossem localizados de forma rápida e fácil. Esses desafios impactaram diretamente a eficácia da automação de testes, necessitando uma abordagem mais robusta e adaptável para lidar com as mudanças na interface e garantir a manutenção sustentável dos scripts de teste.

4.4 Análise dos Resultados dos Testes

Com o objetivo de avaliar o desempenho da automação apresentada na seção anterior e a raiz do problema citado na seção 4.1, foram coletados dados quantitativos das últimas 7 Sprints, e, foi possível executar uma análise em relação aos bugs e as metodologias de teste utilizadas.

Levando em consideração as últimas sprints, foram encontrados 363 defeitos de forma manual e 43 de forma automatizada, o que mostra que os bugs achados através da automação correspondem apenas a 10% do total, como mostra a figura 10. Essa proporção é bastante baixa levando em consideração toda a implementação da automação e a quantidade dos *bugs* totais.

Outra análise realizada foi em relação aos ambientes do projeto. O sistema é dividido em 4 ambientes: Dev, dois ambientes para testes e Produção. Os testes (tanto automação quanto manual) são apenas executados nos ambientes voltados para teste, logo, os *bugs* achados em produção foram achados de forma manual pelos usuários ou na hora da validação das *features*. Nesses ambientes

Tipo de Teste	Número de Bugs encontrados
Automação	43
Manual	363
TOTAL	406

Figura 10: Total de Bugs Encontrados por Cada Tipo de teste em todos os ambientes (Elaborada pelo autor).

de teste, foram achados 40 defeitos pela automação e 257 de forma manual, como mostra a figura 11. Quando comparado a quantidade total de bugs da figura 10 que apresenta o total de 406 com a quantidade total da figura 11 é possível observar que os ambientes de Teste detem cerca de 73% dos bugs totais reportados o que é um valor considerável para um ambiente de teste.

No entanto, é preocupante observar que, nos ambientes de teste como representado na figura 11, os bugs encontrados de forma automatizada correspondem a apenas 13%, o que são números extremamente baixos assim como os encontrados no total dos 3 ambientes. Com isto, é possível identificar um problema evidente na automação, pois, há uma grande discrepância quando comparamos a quantidade de defeitos achados de forma manual e automatizada tanto nos ambientes de teste quanto no sistema como um todo (Ambientes de Teste e Produção).

Tipo de Teste nos Ambientes de QA	Número de Bugs encontrados
Automação	40
Manual	257
Total	300

Figura 11: Total de Bugs Encontrados por Cada Tipo de Teste no Ambiente de QA (Elaborada pelo autor).

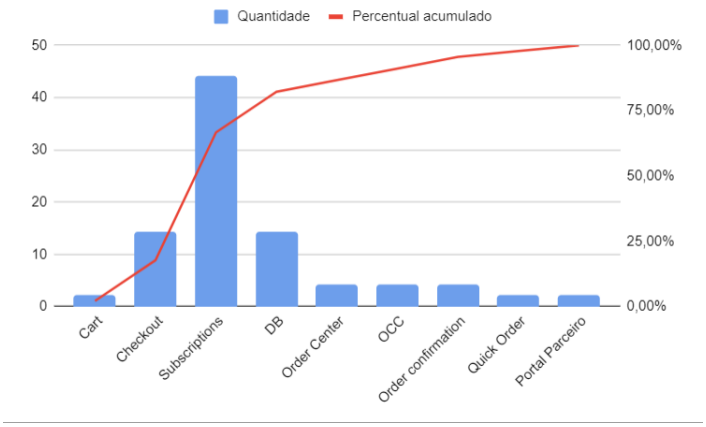

Figura 12: Gráfico de Bugs em Relação a Página do Site (Elaborada pelo autor).

Entrando mais afundo no fluxo do sistema, foram tabulados dados sobre a quantidade de erros por página do sistema. Através da figura 12 podemos observar que, que a maior parte dos bugs ocorrem nas subscriptions, fluxo importante para o funcionamento do modelo de negócio do cliente por essas assinaturas serem um dos principais produtos ofertados. Esses defeitos ocorrem pelo fato

das subscriptions seguirem um modelo complexo de assinatura e estarem ligadas a outros sistemas, como o Portal Parceiro, um portal que serve como administrador das assinaturas e dos usuários. A maioria dos defeitos são identificados na integração entre eles. Quando essa integração falha, algumas assinaturas para determinados usuários param de aparecer, por exemplo.

Em seguida é possível observar que, a maior quantidade de bugs ocorre no checkout, devido a integração com sistemas diversos de terceiros, como gateways de pagamento e processadores de pagamento, o que é um problema grave, pois esta etapa representa o momento crucial em que um cliente decide concretizar uma transação, convertendo o interesse em uma compra efetiva. Portanto, qualquer falha nesse estágio pode resultar em perda imediata de receita.

O principal desafio ao automatizar testes nos planos de assinatura e no checkout reside na complexidade e na variabilidade dos cenários envolvidos nessas áreas críticas do sistema. Testes que envolvem algum tipo de integração com sistemas externos requerem simulações de transações reais e validação de respostas, além disso, o processo de checkout envolve fluxos complexos, como verificação de saldo, aplicação de descontos, cálculo de impostos e processamento de pedidos.

Região	Número de Bugs encontrados
North America	90
Vazio	10
Outras	0

Figura 13: Bugs Encontrados por Região nas Últimas 7 Sprints (Elaborada pelo autor).

Além dos pontos citados acima, foram também tabulados dados para a identificação da ocorrência dos erros por regiões geográficas do sistema, pois, era necessário identificar se esses erros eram localizados ou generalizados. É observado que, como mostra na figura 13 a maior parte dos bugs por região é em *North America*, isso se dá pelo fato de que essa região é considerada a ”base”do sistema e, por isso, os testes automatizados só serem executados nela. Nessa região é requerido a maior parte dos tickets do tipo Story, novas implementações e, consequentemente por ser mais implementada e alterada, há aparição de maiores quantidades de bugs.

Para complementar os dados e resultados apresentados, foram coletados tickets de bugs achados de forma manual dentro do período de abril de 2023 até outubro de 2023 (6 meses), correspondendo as últimas 7 sprints do projeto com o intuito de investigar mais a fundo as raízes dos pontos levantados. Nesses tickets, foram achados 3 principais causas da proporção pequena de defeitos achados pela automação.

5 Discussão sobre os Dados

Considerando os resultados apresentados na seção anterior, onde existe uma proporção muito pequena (10% a 13%) de defeitos sendo encontrados pela automação em relação a todos os *bugs* encontrados no sistema, e além disso, na criticidade dos pontos onde estes defeitos são encontrados (*subscriptions* e *checkout*) e na análise dos tickets coletados manualmente, podemos destacar três principais causas:

- O fluxo seguido pela automação que só cobre os cenários usuais e triviais do sistema.

- Falta de cobertura em outros idiomas pela automação.
- Baixa cobertura da automação de testes nas funcionalidades recém desenvolvidas.

No primeiro ponto, destacamos o *Happy Flow* e o seu fluxo usual devido ao fato que apenas os cenários mais simples estão sendo cobertos e testados por ele. Logo, os cenários mais complexos envolvendo diferentes fluxos e serviços integrados como *trigger* de e-mails e *gateways* de pagamento acabam ficando de fora. Além disso, por conta da automação utilizar a região *North America* como base, bugs relacionados a diferentes regiões acabam ficando de lado. Alguns defeitos acabam só sendo capturados de forma manual, pois, as diferentes regiões disponíveis no site (*North America (NA), Asia-Pacific (APAC), Europe, Middle East, and Africa (EMEA), Latin America and the Caribbean (LAC)*) acabam não sendo testadas pela automação, o que, acaba explicado o motivo de que há bem mais defeitos achados de forma manual.

O segundo ponto foi encontrado após uma investigação nos tickets achados manualmente. Observa-se que, muitos bugs achados dessa maneira, são causados principalmente por conta de falta de tradução, *spelling*, ou, por exemplo é quando há um botão com um tamanho específico (width) que foi desenvolvido baseado em uma região, NA (North America) por exemplo, e por conta disso, quando ocorre a tradução o botão age de forma indevida como a *string* escapando do espaço delimitado, por conta do seu tamanho. A automação acaba não cobrindo esses cenários de diferentes regiões e idiomas.

O último ponto, se dá pelo fato de que, por conta da baixa cobertura da automação, nem todos os tickets tipo Story possuem automação implementada ainda por serem novas features (recursos) adicionados, logo a execução dos seus testes são executadas de forma manual, pois, não há cobertura da automação. Logo, quando há algum defeito achado relacionado a esse ticket ou a esse novo recurso, ele é aberto o marcando como achando manualmente no JIRA.

Considerando os pontos expostos anteriormente, são propostas melhorias na automação com objetivo de diminuir a diferença evidente entre bugs achados manualmente.

5.1 Melhorias Propostas

Após a análise detalhada dos ambientes e forma que os bugs são identificados são propostas melhorias que podem ser úteis ao projeto.

Em primeiro lugar, é preciso aprimorar a cobertura dos testes automatizados. Apesar de custoso [12], expandir os casos de testes para além do fluxo padrão incluindo cenários mais complexos e fluxos excepcionais que os usuários podem seguir ajudará a encontrar e identificar bugs que acabam sendo deixados e não não capturados nos cenários atuais. É importante também não só investir na maior cobertura do fluxo *end-to-end*, mas também no mapeamento de possíveis cenários que podem ser cobertos por testes unitários ou testes de integração que são menos custosos.

Para isso, é necessário documentar os cenários que ficam de fora do fluxo principal e das diferentes regiões do site, e, em seguida, criar casos de testes para que o time de QA implemente-os. Para a implementação da automação, devemos seguir um modelo de *CI (Continuous integration)*[8], os testes cases, após documentados, devem ser divididos em tarefas no JIRA e, decidido através das cerimônias de Scrum quais devem entrar como prioridade na Sprint.

Já para o desenvolvimento, podemos manter a ferramenta já utilizada e conhecida pela equipe, o Webdriver.io, pois, dessa forma, podemos aproveitar os *scripts* que já estão rodando o fluxo principal e/ou os utilizar como base de outros cenários.

No caso das traduções, é preciso considerar a regionalização dos produtos específicos. Para isso, é preciso mapearmos cenários e tickets que utilizam algum produto exclusivo da região e incluí-lo nos casos de testes da automação. Ao testar uma História ou um cenário, é preciso também garantir que automação cubra todas as regiões do site e levar em consideração os seus produtos específicos tanto para cenários triviais a mais complexos. Fazendo isso, problemas que as traduções causam atualmente em relação a elementos de interface como botões e campos de busca poderão ser capturados pela automação, pois, haverá cobertura desses cenários, deixando o QA com mais tempo para focar em outras necessidades do projeto, por exemplo.

Outra melhoria que será de extra importância ao projeto é promover uma cultura de revisão contínua [3] e aprimoramento do processo de testes [12] com a integração dos feedbacks dos QAs e identificando as áreas que precisam de constante melhoria. Além disso, vale a implementação de checkpoints de validação dos cenários de testes com os desenvolvedores pois esta prática ajuda a melhorar a cobertura de casos de teste.

Ao implementar essas sugestões, será possível melhorar a eficácia do sistema de testes como um todo, tanto manualmente como através da automação, reduzindo a disparidade na detecção de bugs entre os métodos de automação e manual.

6 Conclusão

Este trabalho teve o objetivo de apresentar as atividades realizadas na empresa no desenvolvimento da Suíte de testes e os desafios enfrentados: A extensão do time de QA alocado no projeto levando a despriorização da automação, o grande volume de scripts e cenários que precisavam ser desenvolvidos, a dificuldade com identificadores únicos e falta de documentação ao longo do seu desenvolvimento. Além disso, o trabalho teve o intuito de fazer análise com os dados levantados do projeto com objetivo de mostrar a nítida importância de melhorias na automação para aprimorar a sua eficiência. Os principais problemas observados foram:

- Baixo número de *bugs* pegos pela automação em relação aos bugs manuais.
- Falta de cobertura da automação em todo fluxo do site.
- Grande quantidade de *bugs* em traduções.

Os dados levantados levaram em conta as 7 últimas sprints o que equivale, aproximadamente a 6 meses, do projeto através dos relatórios automáticos do JIRA. A primeira dificuldade enfrentada foi a burocracia para ter acesso aos dados. Sabemos que, por serem dados relacionados à performance do projeto de alguma forma, acabam sendo dados sensíveis. Após a coleta, foram gerados gráficos e tabelas e feita uma análise e os problemas observados citados no texto foram identificados através dela.

A Partir do entendimento dos dados e identificação dos problemas, foi possível propor melhorias para que a eficiência da automação fosse aumentada. A princípio, foi proposta a melhoria da co-

bertura do fluxo através do aumento de casos de testes. Para as falhas de traduções, foi proposta a implementação dos cenários na automação e divisão dos testes por região, pois, apenas a região *North America* era coberta. Além disso, foi proposta a implementação de uma cultura de revisão contínua e aprimoramento dos processos de teste para que a automação esteja sempre atualizada e cobrindo a maior parte e, depois, todo o fluxo do site.

Essas melhorias propostas foram concebidas considerando o cenário atual da automação e a sua comparação com os trabalhos levantados para esse projeto. A automação atual não possui controle de fluxo nem integração contínua[8] e é focada na interface gráfica e não há a cobertura das diferentes *interfaces* do sistema, o que não é recomendado por Adad, Boehm, Sievers e Wheaton [9]. Por isso, para os próximos passos, recomenda-se, além da implementação das melhorias sugeridas, uma revisão na arquitetura da automação como um todo implementando o controle de fluxo e a integração contínua de builds, por exemplo e a implementação de testes automatizados não somente na interface do sistema, mas também testes em performance e testes em APIs no *back-end*. Isso garantirá uma cobertura mais abrangente dos cenários, maior eficácia da automação, segurança e estabilidade dos testes, bem como uma expansão da automação para além da interface, contemplando outros aspectos críticos do sistema.

Referências Bibliográficas

[1] ProductPlan. (2024) Quality assurance - what is quality assurance? [Online]. Available: https://www.productplan.com/glossary/quality-assurance/#:~:text=Quality%20assurance%20is%20a%20broad,missed%20requirements%20in%20a%20product.

[2] T. B. Glenford J. Myers, Corey Sandler, *The Art of Software Testing*. Wiley, 2012.

[3] A. Bartié, *Garantia da qualidade de software: adquirindo maturidade organizacional*. Elsevier, 2002.

[4] OneDayTesting. (2019) O que É a pirÂmide de testes? [Online]. Available: https://blog.onedaytesting.com.br/piramide-de-teses/

[5] C. V. d. S. O. Luiz Diego Vidal Santos, *Introdução a Garantia e Qualidade de Software*. Cia do Ebook, 2017.

[6] Webdriver. Why webdriverio. [Online]. Available: https://webdriver.io/docs/why-webdriverio/

[7] Node. (–) About node.js. [Online]. Available: https://nodejs.org/en/about

[8] N. J. C. . R. S. M. Kim, E. H., "Implementing an effective test automation framework," *33rd Annual IEEE International Computer Software and Applications Conference*, 2009.

[9] M. S. M. W. Azad M. Madni, Barry Boehm, "n-tiered test automation architecture for agile software systems," *Conference on Systems Engineering Research*, 2014.

[10] selenium. Page object models. [Online]. Available: https://www.selenium.dev/documentation/test_practices/encouraged/page_object_models/

[11] Jenkins. Jenkins user documentation. [Online]. Available: https://www.jenkins.io/doc/

[12] S. M. CUNHA, "Engenharia de software – uma abordagem À fase de testes," *UFMG*, 2010.