UNIVERSIDADE FEDERAL DE VIÇOSA
CAMPUS DE RIO PARANAÍBA
SISTEMAS DE INFORMAÇÃO

RICARDO B.P. DE CARVALHO - 5948
FELIPE A.J. - 5155
LEANDRO G.O. - 4835

TESTE DE SISTEMA WEB UTILIZANDO FUNCIONALIDADES DA FERRAMENTA SELENIUM

RIO PARANAÍBA
2022

RICARDO B.P. DE CARVALHO - 5948
FELIPE A.J. - 5155
LEANDRO G.O. - 4835

TESTE DE SISTEMA WEB UTILIZANDO FUNCIONALIDADES DA FERRAMENTA SELENIUM

Trabalho Prático da disciplina de Verficação, Validação e Teste de *Software*

Orientador: Professor Leandro Furtado

RIO PARANAÍBA
2022

Lista de ilustrações

Sumário

1 Introdução

A ferramenta Selenium traz uma abordagem de aplicação de testes automatizados em interfaces Web (Selenium (2019)). A Selenium aplica testes funcionais (Caixa Preta) com a perspectiva de um usuário final e o foco deste trabalho foi explorar o nível de testes de sistema por meio da IDE Selenium, interação com Webdriver e do *Grid*.

Estas partições da ferramenta oferecem recurso para: geração facilitada de scripts para testes por meio da interação com o navegador (IDE), bibliotecas para execução de scrits que interagem com o navegador (abordagem pelo Webdriver) e replicação de testes em diversas máquinas, físicas ou virtuais (abordagem pelo *Grid*) (Selenium (2019)).

2 Tipos de Teste

Antes do estudo aplicado da ferramenta foi observado o contexto de aplicação de testes automatizados, formado por: Testes de Unidade (Unit Tests), Testes de Integração (Service Tests) e Testes de Sistema (UI Tests). Conforme a Figura 1 podemos observar a relação entre tipos de teste e esforço para sua aplicação: as categorias próximas à base possuem menor tempo de esforço, já as mais ao topo requerem maior esforço. Isso se deve ao nível de integração das funcionalidades que compõem o produto final. Antes

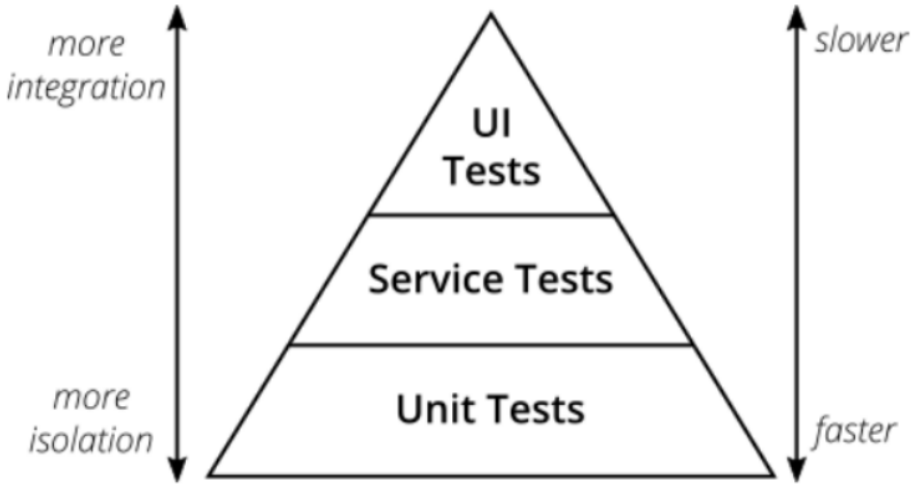

Figura 1 – Pirâmide de Tipos de Teste

Flower (2018)

de prosseguir esclarecemos que testes de sistema verificam se a versão atual é capaz de executar processos ou casos de uso completos do ponto de vista do usuário. Por sua vez os testes de integração verificam se as interfaces entre funcionalidades, que o sistema oferece, funcionam de acordo com o planejado quando exercitadas. E os testes de unidade verificam os componentes mais básicos e singulares do sistema. Este teste antecede os restantes no ciclo de vida de um projeto justamente por ser menos oneroso e antecipar disparos de erros que ocorreriam nos outros tipos de teste.

Sendo assim, no livro de Engenharia de *Software* do Google (MANSHRECK, 2020) é proposta a organização de esforço segundo a pirâmide da Figura 2 (com correspondência na ordenação dos tipos de teste em relação à Figura1).

Apesar da sugestão lógica acima nem todos contextos produtivos enxergam essa disposição em pirâmide de esforço de teste. Existem abordagens como a Ice-Cream Cone (Figura 3, lado esquerdo) que foca na aplicação de testes manuais exploratórios e de testes

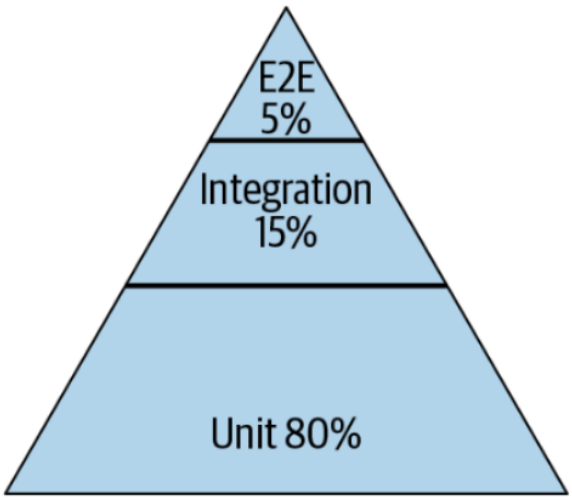

Figura 2 – Pirâmide de Tipos de Teste, com porcentagem

(MANSHRECK, 2020)

automatizados de sistema para que o produto saia do papel para a produção rapidamente. Infelizmente o modelo traz o efeito colateral de teste contínuo mesmo após a entrega. Outro exemplo é a Hourglass (Figura 3, lado direito) que não aplica tantos testes de integração quanto o recomendado devido à alta integração das unidades, tornando esse tipo de teste muito custoso. Mesmo assim ele se apresenta como sendo menos nocivo que o Ice-Cream Cone (MANSHRECK, 2020).

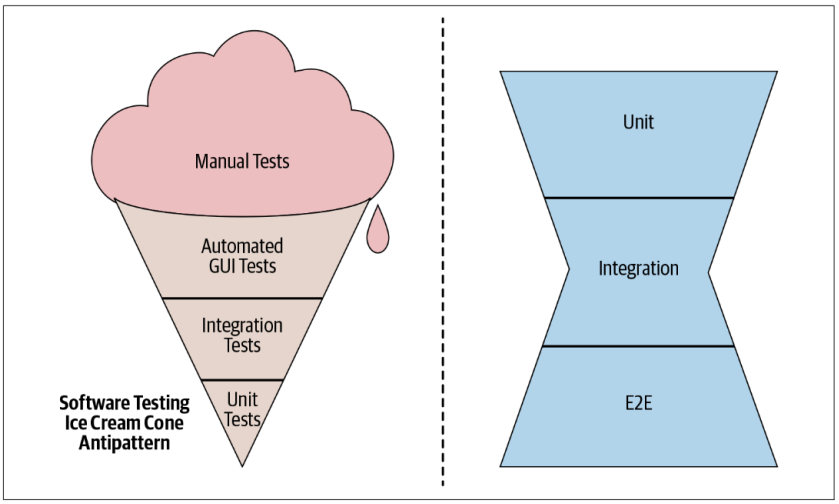

Figura 3 – *Antipatterns: Ice Cream Cone & Hourglass*

(MANSHRECK, 2020)

3 Abordagem Prática

3.1 Selenium IDE

Para eleger um alvo de teste foi considerado um pequeno projeto em JavaScript[1] que traz poucas funcionalidades e uma abertura para aplicação de Partição de Equivalência. Devido essa característica da limitação de funcionalidades seria mais eficiente aplicar o máximo de testes manuais possível (optar por script apenas para testar grande quantidade de entradas), mas para elaboração de um tutorial é um produto qualificado para se evidenciar as facilidades que a extensão Selenium IDE traz.

As etapas necessárias para teste de sistema foram: Elaboração de Casos de teste (via Partições de Equivalência), gravação da interação com interface por meio da Selenium IDE no Chrome e projeto em Local Host, adição de comandos no script com finalidade de leitura de retornos, execução do script e discussão dos resultados.

A exploração do método de partição de equivalência consistiu em inserir um representante de cada partição de entrada. Aquelas classificadas como válidas correspondem ao intervalo: [1,100], já as inválidas podem ser representadas por qualquer membro no conjunto da negação de [1,100], ou seja: valores menores que 1 e maiores que 100. Para explorar o domínio também foi adicionado um caso de teste contendo uma letra.

Durante a discussão de resultados percebemos que o projeto claramente não passou por testes de unidade. Foram quatro das funcionalidades (cálculo da média, localização do valor mínimo e localização do valor máximo) que apresentaram defeito. Foi considerando que é apenas o funcionamento singular do cálculo da média e leitura de entrada (não existe interação entre funções nesses processos) que chegamos a conclusão que o problema seria cada unidade em particular.

3.2 Selenium Webdriver

Para início desta etapa o script de testes de entradas inválidas foi exportado no formato da linguagem de programação Python. A partir do uso de bibliotecas Python e Selenium foi possível implementar loops (estruturas de repetição) para inserções múltiplas. Com isso exploramos um recurso não abordado pela IDE e que otimiza o processo de criação de scripts e, diretamente, o de testes.

[1] https://github.com/patriani/AnalisandoNumeros.git

3.3 Selenium *Grid*

O *Grid* permite que testes sejam executados remotamente e em diversas máquinas paralelamente. Isso permite que diferentes ambientes sejam testados como diferentes Sistemas Operacionais e diferentes navegadores. Para aplicação deste recurso foi necessário instalar o servidor do *Grid* Selenium[2] e a versão mais estável das dependências da linguagem Java. Com isso deve ser possível utilizar o console do *Grid* para execução de scripts e configurações.

Considerando os componentes: cliente, roteador, e nó(abstração de máquina controlada pelo lado cliente), os principais objetivos e funcionalidades do Selenium *Grid* são:

- Ponto de entrada central para os testes: Um único cliente pode gerenciar e controlar navegadores de várias máquinas simultâneas, com sistemas operacionais distintos.
- Gerenciar e controlar os nós: Ele gerencia e controla as maquinas nós, ou as maquinas de testes, para uma coleta mais precisa de todos os testes executados pelo Selenium.
- Escalonamento: Selenium permite uma execução de scripts em massa, Assim Escalonando os testes, e otimizando tempo e recursos em testes que necessitam testes em várias maquinas com sistemas operacionais distintos.
- Executar testes em paralelo: Selenium permite testes paralelos de scripts em maquinas diferentes.
- Teste de plataforma cruzada: Selenium *Grid* tem suporte a vários Sistemas operacionais que possuem ou aceitem o Java em sua arquitetura.
- Balanceamento de carga: Selenium *Grid* permite realizar testes de exaustão ou limitar o número de testes que cada máquina deve fazer.

[2] https://www.selenium.dev/downloads/

4 Conclusão

Apesar do estudo limitado foi retornado um conhecimento vasto a partir da prática dos conceitos expostos em aula. Entendemos a visão que um testador precisa ter ao interagir com um produto, mesmo que indiretamente (Teste de Aceite), que envolve: o planejamento das etapas de teste, estudo dos fluxos do produto e elaboração de casos de teste. Também entramos em contato com boas práticas que a própria documentação da Selenium oferece. Uma dessas boas práticas envolve a minimização de dados provenientes do navegador para redução de esforço. Essa prática também favorece tornando os testes mais generalizáveis (propensos a reuso). Por fim, também foi vantajoso explorar uma ferramenta popular no mercado e que apresenta facilidades para a aplicação de testes funcionais.

5 Créditos

Este capítulo tem a finalidade de esclarecer a participação de cada nome citado na "autoria":

- Leandro G.O.: responsável por elaborar automação web com *webdriver* em python;
- Felipe A.J.: responsável pelo aprofundamento na ferramenta *Grid*;
- Ricardo B. Patriani de Carvalho: responsável pela contextualização teórica, projeto de teste funcional, colaboração nos testes via *Webdriver* e formatação do documento em LaTeX.

Referências

FLOWER, M. ***The Pratical Test Pyramid***. 2018. Disponível em: <https://martinfowler.com/articles/practical-test-pyramid.html>.

MANSHRECK, H. W. T. **Software Engineering at Google**. [s.n.], 2020. Disponível em: <https://abseil.io/resources/swe_at_google.2.pdf>.

SELENIUM. **Documentação**. 2019. Disponível em: <https://www.selenium.dev/selenium-ide/docs/en/introduction/getting-started>.