Usando o teste ponta a ponta para garantia de confiabilidade de um Sistema Integrado de Gestão: uma prova de conceito

Yury Alencar Lima[1], **Elder de Macedo Rodrigues**[1], **Rafael Alves Paes de Oliveira**[2], **Maicon Bernardino da Silveira**[1]

[1]Universidade Federal do Pampa (UNIPAMPA)
Código Postal 97.546-550 – Alegrete – RS – Brasil

yuryalencar19@gmail.com, elderrodrigues@unipampa.edu.br

bernardino@acm.org

[2]Universidade Tecnológica Federal do Paraná (UTFPR)
Código Postal 85.660-000 – Dois Vizinhos – PR – Brasil

raoliveira@utfpr.edu.br

Abstract. *In order to improve management and decision making, companies tend to organize their information through Integrated Management Systems (ERPs). ERPs must be reliable systems, as they represent the business and influence decision making. However, ensuring the reliability of an ERP through tests is not a trivial task, as they are generally large systems. The use of automated end-to-end testing can be an effective solution. This study presents a proof of concept of the use of end-to-end tests, through an architecture, metrics and ways of performing interactions with each layer of the application. Thus resulting in a reduction in the time needed to resolve and detect errors in the system.*

Resumo. *Com o intuito de melhorar o gerenciamento e tomada de decisões, as empresas tendem a organizar suas informações através dos Sistemas Integrados de Gestão (ERPs). Os ERPs devem ser sistemas confiáveis, pois representam o negócio e influenciam nas tomadas de decisões. Entretanto garantir a confiabilidade de um ERP através de testes não é uma tarefa trivial, pois são geralmente sistemas de grande porte. O uso de testes automatizados de ponta a ponta pode ser uma solução eficaz. Este estudo apresenta uma Prova de Conceito (POC) do uso de testes ponta a ponta, através de uma arquitetura, métricas e métodos de realizar interações com cada camada da aplicação. Resultando assim em uma redução do tempo necessário para resolver e detectar erros no sistema.*

1. Introdução

A crise do *software* que ocorreu por volta da década dos anos de 1960 e foi referente à baixa qualidade das aplicações desenvolvidas, ocasionando algumas vezes até mesmo a inutilização dos sistemas [Naur and Randell 1969]. A fim de aumentar a confiabilidade dos sistemas, a *Engenharia de Software* (ES) surgiu para sistematizar o processo de desenvolvimento, de modo com que a qualidade das aplicações pudesse ser replicável, independentemente do contexto e domínio de aplicação [Sommerville 2011]. Uma das alternativas dentro da ES para garantir a confiabilidade das aplicações é por meio de atividades de verificação, validação e testes de *software*. Dentre as diversas ações de verificação, os testes funcionais têm como finalidade verificar se o sistema implementado funciona como o esperado a partir da especificação [Delamaro et al. 2013]. Esses testes são parte do ciclo de desenvolvimento e demandam um tempo significativo dentro do processo de desenvolvimento, aumentando a qualidade do produto, mas gerando altos custos [Pressman and Maxim 2016].

Com a finalidade de reduzir esses custos é possível realizar a automatização dos testes. A automatização reduz o tempo de verificação quando a adição de uma nova funcionalidade ou manutenção modifica o sistema, pois não são mais necessários testadores para verificar toda a aplicação, apenas executar novamente os *scripts* de teste [Ivory and Hearst 2001]. Entretanto, a automatização de testes de sistema visa identificar erros por meio da interface com o usuário [Delamaro et al. 2013]. Todavia essa prática pode não ser eficaz em garantir a qualidade necessária em alguns contextos, como, por exemplo, em aplicações *web* atuais mais precisas que requerem uma alta confiabilidade [Clerissi et al. 2017].

Um exemplo de *software* que possui a necessidade de testes constantes é o *Sistema Integrado de Gestão* (ERP – do inglês *Enterprise Resource Planning*), uma vez que ele possui todas as regras de negócio e dados da empresa que não podem ser perdidos, pois influenciam nas tomadas de decisão [Dechow and Mouritsen 2005]. Esses sistemas podem originar novos desafios para a automatização como o gerenciamento dos dados de teste, tempo de execução devido a grandes fluxos e imprecisão para delimitar em qual camada da aplicação se encontra o defeito [Palmér and Waltré 2015].

Uma possível abordagem para mitigar esses desafios, aumentar a eficácia e reduzir os custos é o uso de testes de ponta a ponta. Os testes ponta a ponta verificam as funcionalidades do sistema como um todo, analisando não somente a *interface* com o usuário mas também a estrutura interna da aplicação [Palmér and Waltré 2015]. Além do uso do sistema do ponto de vista do usuário, o teste também é capaz de verificar as outras camadas da aplicação como o banco de dados e chamadas à *Interface* de Programação de Aplicativos (API – do inglês *Application Program Interface*), o que pode facilitar a localização do defeito [Paul 2001].

A fim de analisar o impacto no uso dos testes de ponta a ponta na qualidade de ERPs, esse estudo apresenta uma prova de conceito realizada em um módulo crucial de um sistema real de gestão em nuvem. Além das métricas, questões e objetivos, são apresentadas as tecnologias utilizadas, estrutura dos testes, avaliação empírica e resultados. A fim de apresentar uma visão geral do uso de testes de ponta a ponta no cenário real, e quais foram os impactos do seu uso na qualidade de um produto de *software*. Além de viabilizar a aplicação desses testes, através das tecnologias e estrutura utilizada.

Com base nisso, o artigo foi estruturado da seguinte forma: A Seção 2 apresenta o referencial teórico apresentando os testes funcionais e de ponta a ponta, juntamente com as tecnologias utilizadas para a prova de conceito; A Seção 3 apresenta os trabalhos relacionados ao respectivo estudo; A Seção 4 apresenta a arquitetura dos testes ponta a ponta e como foi realizada a Avaliação Empírica; A Seção 5 apresenta os resultados baseado nas métricas juntamente com as ameaças à validade do estudo, e; A Seção 6 apresenta as considerações finais e os trabalhos futuros relacionados.

2. Referencial Teórico e Técnico

Esta seção é responsável por apresentar os conceitos e tecnologias que foram utilizadas para a realização desse estudo. Com base nisso, esta seção foi subdividida em Aspectos Conceituais apresentado na Seção 2.1 e Tecnologias Utilizadas presente na Seção 2.2.

2.1. Aspectos Conceituais

O presente estudo foi baseado nos conceitos de testes funcionais de *software* e testes de ponta a ponta, a fim de aumentar a confiabilidade e qualidade de um Sistema. Normalmente, as equipes de teste buscam o aumento da qualidade de um sistema através da aplicação de diferentes abordagens e técnicas de teste. Por exemplo, o teste funcional visa

revelar falhas e detectar defeitos em uma aplicação, aumentando a qualidade do produto antes da entrega para o cliente [Delamaro et al. 2013]. Além disso, essa prática auxilia na verificação da aplicação em relação ao requisitado pelo cliente [Rios et al. 2007]. Nesses testes, o sistema é considerado uma caixa-preta, ou seja, a aplicação é analisada do ponto de vista do usuário final, não sendo verificada sua estrutura interna, somente seu comportamento [Delamaro et al. 2013]. A especificação desse comportamento é derivada dos requisitos do cliente e transformadas em casos de teste, os quais são um conjunto de condições para a realização dos testes, possuindo alguns elementos principais como, por exemplo, as entradas do sistema, ações a serem realizadas e resultados esperados [Myers et al. 2011]. Assim, possibilita a detecção de problemas funcionais antes do produto ser entregue a seu usuário final, reduzindo os custos relacionados às correções após a entrega da aplicação, sendo esses os mais altos do ciclo de vida de *software* [Delamaro et al. 2013]. Esses conceitos influenciaram na criação dos casos de teste, levando em consideração o ponto de vista do usuário final. Resultando assim na verificação não somente dos comportamentos das telas e sim do fluxo total das ações realizadas pelo cliente.

Assim como o teste funcional, os testes de ponta a ponta também possuem como principal objetivo a detecção de defeitos dentro de uma aplicação, entretanto essa abordagem leva em consideração todas as camadas do *software* [Paul 2001]. Essa abordagem visa identificar falhas em qualquer camada do produto de *software*, por meio da automação e verificação não somente da interação do usuário com a interface, mas também da comunicação com as APIs (*Application Programming Interface*) e banco de dados [Palmér and Waltré 2015]. Assim, os conceitos de teste ponta a ponta surgiram como uma alternativa dentro da POC para localizar precisamente os defeitos da aplicação. Dessa forma, reduzindo tanto no tempo necessário para o desenvolvedor ou testador encontrar a origem do problema, quanto na identificação do impacto do erro dentro da aplicação.

2.2. Tecnologias Utilizadas

A fim de viabilizar a aplicação tanto dos testes funcionais de *software*, quanto dos testes de ponta a ponta automatizados, o presente estudo utilizou um conjunto de ferramentas. Essa seção visa apresentar as tecnologias Robot *Framework*, Selenium WebDriver, Database Library e HTTP Library Requests, bem como detalhar qual sua função na automação dos testes no contexto do trabalho realizado.

O *Robot Framework* é uma tecnologia *Open Source* com o objetivo de realizar a criação de testes de aceitação de sistema. O seu uso é baseado em uma sintaxe tabular e de fácil aprendizado [Bisht 2013]. O *framework* também utiliza de linguagem natural tanto para a definição das *Keywords* que são como métodos de teste, quanto para os *Steps* que são os passos dentro de cada *Keyword*. Dentre os pontos positivos para seu uso, destacam-se a compatibilidade com várias bibliotecas externas e a possibilidade de criação de bibliotecas para domínios específicos [Bisht 2013]. Essa personalização pode ser realizada por meio de *scripts* criados em linguagem Python [Lutz 2001] ou, então em Java [Gosling et al. 2000]. Nessa prova de conceito foi utilizado o *Robot Framework* devido a possibilidade de aplicação em diversos tipos de sistemas como *web*, *desktop* e mobile. A tecnologia foi utilizada juntamente com o Python decorrente da grande quantidade de bibliotecas de suporte para o respectivo domínio. Além disso, o *framework* teve grande influência na escrita dos casos de teste representando o domínio sem restrições, beneficiando também a documentação das funcionalidades testadas.

O Selenium WebDriver é uma tecnologia responsável por realizar a manipulação do navegador web, por meio de interações com a *interface* gráfica da aplicação *web*,

representando as ações do usuário [Avasarala 2014]. A tecnologia captura os elementos gráficos por meio dos atributos presentes dentro do *HyperText Markup Language* (HTML), que por sua vez representam a *interface* com o usuário, viabilizando a realização de manipulações e ações com esses elementos. Sua adoção dentro da prova de conceito foi decorrente da existência de uma biblioteca nativa para o *Robot Framework* e ser uma recomendação da *World Wide Web Consortium* (W3C) sendo esta a principal organização de padronização de aplicações *web* [W3C 2018]. Assim, todas as ações realizadas na aplicação por meio da *interface* gráfica representando o usuário final dentro da POC foram através do Selenium WebDriver.

O *Database Library* é uma biblioteca externa que provê ao *Robot Framework* a realização da comunicação com diferentes tipos de banco de dados [See 2019]. A biblioteca possui duas versões uma para Python e outra focada em Java. Ambas são compostas por uma série de palavras chave que realizam a comunicação com o banco de dados da aplicação [See 2019]. Explorando as palavras reservadas do *Robot Framework* é possível realizar diversas validações nessa camada da aplicação, desse modo sua adoção na verificação do ERP utilizado nessa prova de conceito foi necessária. Viabilizando assim, a adoção dos conceitos de teste ponta a ponta, levando em consideração outra camada da aplicação, sendo essa responsável pela comunicação com o banco de dados.

O *HTTP Library Requests* é uma biblioteca que tem como finalidade realizar chamadas por meio do *HyperText Transfer Protocol* (HTTP) que tem o intuito de contactar serviços implementados pela parte lógica da aplicação, que neste caso estão presente nas APIs utilizadas [Evcimen 2020]. Desse modo, é possível verificar também o processamento com chamadas para a API que provê o serviço. Assim, analisando outra camada geralmente presente nas aplicações *web*. Essa biblioteca também é voltada para o uso por meio do *Robot Framework* nas versões em Python e em Java. Como o *Database Library* essa biblioteca também é composta por um conjunto de palavras chave para simplificar essa atividade provendo uma fácil automação [Evcimen 2020]. Com base nisso também foi necessária para a prova de conceito, a fim de testar os retornos da lógica da aplicação e tornando possível o teste de ponta a ponta do Sistema de Gestão Integrado em Nuvem.

3. Trabalhos Relacionados

Dentre os trabalhos relacionados ao estudo conduzido, é possível citar [Clerissi et al. 2017] que apresenta a importância das aplicações *web* dentro do cotidiano das pessoas e a necessidade de garantia da qualidade desses sistemas. Esse estudo também contém os conceitos de teste de ponta a ponta, frisando sua eficácia dentro do cenário de qualidade. Com base nisso, os autores apresentam uma abordagem com o foco na geração de *scripts* de testes de ponta a ponta, utilizando especificações textuais ou, então baseadas em *Unified Modeling Language* (UML) [Clerissi et al. 2017].

[Debroy et al. 2018] apresenta os desafios relacionados à automação de aplicações *web*, resultando em um conjunto de lições aprendidas e informações técnicas de como gerenciar os testes em ambientes ágeis de desenvolvimento e, assim manter a confiabilidade da aplicação testada [Debroy et al. 2018].

4. Prova de Conceito

Nessa seção são apresentados todos os detalhes relacionados à prova de conceito. Pontualmente, apresentam-se: (1) a estrutura do projeto de testes; (2) o módulo do ERP denominado RadarXML, no qual as tecnologias foram aplicadas e, por fim, (3) a estruturações e condução da avaliação empírica.

4.1. Estrutura para o uso do teste ponta a ponta

A fim de permitir a replicabilidade do projeto de testes para outros domínios e facilitar o uso de padrões de projetos, foi definida a estrutura apresentada nessa seção.

Essa arquitetura conta com a divisão dos testes em seis diretórios que são apresentados a seguir: (1) **config:** diretório que contém os arquivos *Robot* referentes à configuração do projeto com informações generalizadas como *host* e *URL* base da aplicação, usuários e o navegador que será utilizado nos testes; (2) **libs:** Nesse diretório é onde são inseridos os *scripts* em Python que realizam procedimentos do domínio não detectados em outras bibliotecas do Robot Framework, como busca em arquivos JSON, geração de CNPJs, dentre outros; (3) **pages:** Esse diretório é voltado para o uso do padrão de projeto *Page Object*, no qual cada página da aplicação é um arquivo, e todas suas ações são inseridas em formas de métodos, possibilitando um melhor reúso e entendimento dos *scripts* de teste; (4) **services:** Representa o diretório de serviços utilizados dentro dos testes, geralmente utilizados para realizar as pré-condições dos cenários, por meio de interação com o banco de dados, API e *Interface* Gráfica; (5) **tests:** Todos os *Steps* dos testes e *Keywords* são definidas em arquivos neste diretório, o uso de *Keywords* proporciona a definição de ações mais representativas para a validação com o usuário final; (6) **resources:** Esse diretório possui todos os arquivos contendo os cenários de teste utilizando termos do domínio da aplicação, juntamente com uma breve documentação das funcionalidades.

Dentro desses diretórios podem ser criadas outras pastas a fim de organizar as páginas, serviços, cenários e os *Steps* dos testes de acordo com o módulo a ser testado.

4.2. RadarXML

O Sistema Integrado de Gestão em Nuvem, no qual esta prova de conceito foi realizada é um produto real utilizado por mais de 200 clientes. Além disso, este ERP é um sistema completo de gestão para varejo, que requer uma alta confiabilidade. Dentre os módulos existentes na aplicação a POC foi focada apenas no RadarXML. Esse módulo é responsável por consultar em tempo real todas as notas fiscais na Secretaria da Fazenda direcionadas à empresa que utiliza o sistema. Partindo disso, o estoque, preços dos produtos, parte financeira, contábil e fiscal são atualizadas gerando uma série de indicadores e relatórios. Além disso, também é possível realizar a importação manual de notas fiscais, ajustes de fornecedores, produtos e faturamento por meio do módulo.

4.3. Avaliação Empírica

Essa seção apresenta as informações relacionadas à Avaliação Empírica, como o *Design* Experimental, Objetivos e Métricas e por fim os procedimentos realizados para facilitar a análise dos resultados encontrados.

4.3.1. *Design* Experimental

O ERP utilizado na avaliação impõe restrições aos testes realizados somente através da *interface* gráfica da aplicação, como não permitir a exclusão de nenhum usuário, nota fiscal, fornecedor, dentre outras informações presentes. Isto acontece para possibilitar uma maior integridade e disponibilidade dos dados. Entretanto, isso implica que os testes sempre deveriam inserir novos dados geralmente aleatórios, mas alguns dados como notas fiscais acabam sendo mais complicados de serem gerados automaticamente. Uma alternativa seria realizar sempre a limpeza do banco de dados antes de realizar a execução de cada teste, todavia esta prática não é viável devido ao tempo de espera para a base de

dados voltar a ficar disponível, levando em consideração as características da infraestrutura onde estas informações estão hospedadas. Assim, o teste de ponta a ponta foi uma solução idealizada tanto para mitigar o problema de gerenciamento de dados de teste, quanto para utilizar dados reais nas execuções, simulando assim ações similares a aquelas executadas pelos usuários reais. A fim de reduzir o esforço para a aplicação da solução proposta foram escolhidas as tecnologias apresentadas na Seção 2.2.

4.3.2. Objetivos e Métricas

O objetivo dessa prova de conceito foi melhorar a qualidade e confiabilidade de ERP utilizado por diversos clientes, por meio de testes ponta a ponta, verificando cada camada da aplicação. A fim de analisar o uso dos testes ponta a ponta e das tecnologias escolhidas foram definidas as seguintes questões (Q): **Q01:** Quantos erros foram detectados no módulo RadarXML devido ao uso dos testes de ponta a ponta? **Q02:** Qual a diferença no tempo necessário para realizar a manutenção dos *scripts* de teste, comparando os testes de ponta a ponta e funcionais tradicionais?

Com a finalidade de responder a **Q01** foi utilizado o número de erros encontrados. Por fim a **Q02** pode ser analisada com base no tempo em horas necessárias para resolver um problema em comum, tanto na automação ponta a ponta, quanto na realizada somente por meio da *Interface* Gráfica.

4.3.3. Procedimentos

Os procedimentos que serão apresentados foram executados durante duas homologações distintas do ERP em nuvem na empresa responsável pelo produto. A quantidade de erros encontrados por meio dos testes de ponta a ponta, referente à **Q01**, foram coletadas com base nos resultados dos automatizados e computadas em uma planilha somente após a confirmação do defeito. Devido à grande quantidade de testes e diversas alterações para serem verificadas, na segunda homologação foram detectados falsos positivos provocados por alterações em funcionalidades já automatizadas. As métricas referentes à **Q02**, foram coletadas por meio do tempo de detecção e correção dos defeitos encontrados nos testes automatizados. Nessa atividade foram necessários dois automatizadores. As informações coletadas foram armazenadas em uma planilha somente após o teste corrigido ser executado conforme o comportamento esperado.

5. Resultados e Discussões

Após a execução dos procedimentos acima, foi realizada uma análise dos testes de ponta a ponta no impacto na qualidade. O resultado dessa análise e as ameaças à validade da prova de conceito são apresentados nessa respectiva seção.

5.1. Análise de Impacto na Qualidade

Durante as homologações foram encontrados cinco erros no módulo RadarXML, que foi considerado como um bom resultado, tendo em vista que antes de chegar à esta etapa já haviam sido executados diversos testes em ambientes de desenvolvimento e de qualidade. Além disso, como a empresa utiliza de práticas ágeis de desenvolvimento como entrega e integração contínua, o tempo entre as homologações é relativamente menor, podendo levar de uma a duas semanas incluindo o tempo destinado a implementação das funcionalidades. Sabendo disso, os erros encontrados pelos testes de ponta a ponta não foram somente relativos à *Interface* Gráfica da aplicação, o que impactou na redução do tempo

Métrica	Testes de Ponta a Ponta	Testes Funcionais
Quantidade de Erros Encontrados	5	0
Tempo de Manutenção	30 minutos	1 hora

Tabela 1. Métricas Coletadas

de correção, devido ao relatório de execução apresentar em qual camada foi manifestada a falha. Assim, respondendo à **Q01**.

Com a atualização constante do ERP, nesse espaço de tempo foram detectados falsos negativos. Com base nisso, foi possível realizar a medição em relação a manutenção dos testes ponta a ponta com *Robot Framework*, com os testes funcionais utilizando o *framework Cucumber*. O que resultou em uma diferença de tempo de trinta minutos a mais para resolver o mesmo problema utilizando o *Cucumber*, respondendo à **Q02**. Isto aconteceu pois como a aplicação não permite a exclusão dos dados, os testes realizados somente através da *Interface* Gráfica utilizavam dados gerados automaticamente. O que aumentou a complexidade para o entendimento do cenário através da execução dos testes. O uso de dados gerados aleatoriamente também dificultou a análise do cenário dentro da aplicação, pois é necessário saber exatamente quais eram estes dados e o que representavam. Enquanto nos testes de ponta a ponta todas as pré-condições foram realizadas diretamente no banco de dados ou com chamadas de serviços através das APIs. O que tornou não só o teste mais rápido mas também possibilitou a inserção de dados reais e representativos que foram facilmente encontrados. As métricas coletadas podem ser visualizadas através da Tabela 1.

5.2. Ameaças à Validade

Naturalmente, devido ao escopo específico dos estudos, diversas ameaças à validade foram ponderadas. Considerando as ameaças de *Construção*, que considera a relação entre a teoria proposta e os resultados observados, estes pesquisadores consideram a existência de ameaças baixas e poucas chances da existência de alguma relação casual entre a implementação da estratégia automatizada de teste e os defeitos revelados. Essa mesma análise é estendida para a avaliação das ameaças às validades *Internas* do estudo, apesar de não terem sido realizadas avaliações estatísticas, acredita-se que o tratamento realizado teve influência direta nos resultados aferidos. No tocante às ameaças *Externas* (generalização de resultados fora do escopo estudado), estes autores consideram as ameaças como sendo médias e moderadas. É importante destacar que estudos mais amplos precisam ser realizados para aferir a generalização dos resultados e da aplicabilidade da estratégia e diminuição de tais ameaças. Sendo assim, a generalização do trabalho configura o principal ponto a ser melhorado em trabalhos futuros nessa mesma linha investigativa. Por fim, considerando as ameaças à *Conclusão* do estudo, é possível afirmar que tais ameaças são consideradas baixas, haja vista que os resultados e os defeitos na aplicação industrial utilizada como cobaia somente puderam ser revelados a partir da estratégia de teste automatizada proposta.

6. Considerações Finais

Com base nos impactos detectados na qualidade do ERP em nuvem, o uso dos testes de ponta a ponta se mostrou eficaz para o respectivo contexto. O tempo de execução, detecção do local do defeito e uso de dados reais e representativos foram os pontos fortes encontrados do uso destas tecnologias. Entretanto, a sua aplicação pode requisitar um grupo de testadores com mais conhecimento relacionado a automação e à arquitetura do sistema ser testado. Baseado nos benefícios encontrados o uso dos testes de ponta a ponta

nas aplicações web ainda serão utilizados, e como trabalhos futuros foi planejado novas provas de conceito do seu uso em outros tipos de *software* como *mobile* e *desktop*.

Referências

Avasarala, S. (2014). *Selenium WebDriver practical guide*. Packet.

Bisht, S. (2013). *Robot framework test automation*. Packet.

Clerissi, D., Leotta, M., Reggio, G., and Ricca, F. (2017). Towards the generation of end-to-end web test scripts from requirements specifications. In *IEEE 25th International Requirements Engineering Conference Workshops*, pages 343–350. IEEE.

Debroy, V., Brimble, L., Yost, M., and Erry, A. (2018). Automating web application testing from the ground up: Experiences and lessons learned in an industrial setting. In *IEEE 11th International Conference on Software Testing, Verification and Validation*, pages 354–362.

Dechow, N. and Mouritsen, J. (2005). Enterprise resource planning systems, management control and the quest for integration. *Accounting, organizations and society*, pages 691–733.

Delamaro, M., Jino, M., and Maldonado, J. (2013). *Introdução ao teste de software*. Elsevier Brasil.

Evcimen, B. (2020). HTTP Library (Requests). `https://github.com/bulkan/robotframework-requests/#readme`. Online; acessado 28 Fevereiro de 2020.

Gosling, J., Joy, B., Steele, G., and Bracha, G. (2000). *The Java language specification*. Addison-Wesley Professional.

Ivory, M. Y. and Hearst, M. A. (2001). The state of the art in automating usability evaluation of user interfaces. *ACM Computing Surveys (CSUR)*, pages 470–516.

Lutz, M. (2001). *Programming python*. "O'Reilly Media, Inc.".

Myers, G. J., Sandler, C., and Badgett, T. (2011). *The art of software testing*. John Wiley & Sons.

Naur, P. and Randell, B. (1969). Software engineering: Report of a conference sponsored by the nato science committee, garmisch, germany, 7th-11th october 1968. *NATO Science Committee*.

Palmér, T. and Waltré, M. (2015). Automated end-to-end user testing on single page web applications.

Paul, R. (2001). End-to-end integration testing. In *Proceedings Second Asia-Pacific Conference on Quality Software*, pages 211–220. IEEE.

Pressman, R. and Maxim, B. (2016). *Engenharia de Software*. McGraw Hill Brasil.

Rios, E., Cristalli, R., Moreira, T., and Bastos, A. (2007). Base de conhecimento em teste de software. 2ª edição S. Paulo.

See, F. A. V. (2019). Database Library. `http://franz-see.github.io/Robotframework-Database-Library/`. Online; acessado 29 Fevereiro de 2020.

Sommerville, I. (2011). *Software Engineering 9th Edition*. Pearson.

W3C (2018). WebDriver W3C Recommendation. `https://www.w3.org/TR/webdriver1/`. Online; acessado 29 Fevereiro de 2020.