**Avaliação da portabilidade entre fornecedores de teste como
serviço na computação em nuvem**

Ricardo Ramos de Oliveira

Tese de Doutorado do Programa de Pós-Graduação em Ciências de
Computação e Matemática Computacional (PPG-CCMC)

SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito:

Assinatura: _____

Ricardo Ramos de Oliveira

Avaliação da portabilidade entre fornecedores de teste como serviço na computação em nuvem

Tese apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, como parte dos requisitos para obtenção do título de Doutor em Ciências – Ciências de Computação e Matemática Computacional. *VERSÃO REVISADA*

Área de Concentração: Ciências de Computação e Matemática Computacional

Orientador: Prof. Dr. Adenilso da Silva Simão

USP – São Carlos
Março de 2018

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados inseridos pelo(a) autor(a)

R175a Ramos de Oliveira, Ricardo
 Avaliação da portabilidade entre fornecedores de
 teste como serviço na computação em nuvem / Ricardo
 Ramos de Oliveira; orientador Adenilso da Silva
 Simão. -- São Carlos, 2018.
 229 p.

 Tese (Doutorado - Programa de Pós-Graduação em
 Ciências de Computação e Matemática Computacional) --
 Instituto de Ciências Matemáticas e de Computação,
 Universidade de São Paulo, 2018.

 1. Computação em Nuvem. 2. Teste como Serviço
 (TaaS). 3. Vendor Lock-in. 4. Automatização de
 Testes. 5. Experimento Controlado. I. da Silva
 Simão, Adenilso, orient. II. Título.

Ricardo Ramos de Oliveira

Evaluation of portability among testing as a service
providers in cloud computing

Doctoral dissertation submitted to the Institute of
Mathematics and Computer Sciences – ICMC-USP, in
partial fulfillment of the requirements for the degree of
the Doctorate Program in Computer Science and
Computational Mathematics. *FINAL VERSION*

Concentration Area: Computer Science and
Computational Mathematics

Advisor: Prof. Dr. Adenilso da Silva Simão

USP – São Carlos
March 2018

Dedico à minha mãe Luzia, ao meu pai José, à minha irmã Cecília que contribuíram, de um modo muito especial, para o meu crescimento como ser humano e como profissional.

Amo todos vocês!

AGRADECIMENTOS

Agradeço primeiramente a Deus por me conceder o dom da vida e por estar sempre ao meu lado, iluminando o meu caminho e guiando os meus passos, concedendo-me também força e coragem para enfrentar todos os desafios da vida.

Aos meus pais, Luzia da Silva Ramos de Oliveira e José Timóteo de Oliveira, e à minha irmã, Cecília Ramos de Oliveira, pessoas que sempre me apoiaram e ajudaram a realizar e a concretizar todos os meus sonhos e a lutar pelos meus objetivos. Sempre serei grato a todos vocês!

A todos os meus amigos, pelo companheirismo e pelas horas de descontração. Em especial, aos meus amigos: Roberto Naves Ungarelli, Renan Martins, José Augusto Fiorucci, Victor Hugo Santiago Costa Pinto, André de Lima Salgado, Elias Adriano Nogueira da Silva, Helder Jefferson Ferreira da Luz, Rafael Messias Martins, Robson Vinícius Vieira Sanchez, Álvaro Paçó Neto e ao Diogo de Jesus Pina pelas inúmeras e valiosas contribuições durante o desenvolvimento deste trabalho. Sempre me lembrarei de todos vocês. Obrigado a todos!

Ao meu orientador, o Professor Dr. Adenilso da Silva Simão, pelo seu caráter, profissionalismo, competência, paciência e dedicação durante as diversas atividades de pesquisa e desenvolvimento deste trabalho.

A Angela Cristina Pregnotato Giampetro por suas inestimáveis contribuições de revisões de todos os artigos em inglês gerados a partir desta tese. Seus valiosos comentários e sugestões foram de extrema importância e relevância para este trabalho.

Aos demais professores do Instituto de Ciências Matemáticas e de Computação (ICMC) da Universidade de São Paulo (USP) que contribuíram para minha formação, em especial, à Professora Dra. Renata Pontin de Mattos Fortes.

A todos do Laboratório de Engenharia de Software (LabES) e do grupo de pesquisa, pela amizade, pelo agradável convívio, pelas discussões e troca de experiência enriquecedoras.

Aos funcionários do ICMC-USP por todo apoio e suporte, em especial aos técnicos da Seção Técnica de Informática - STI.

Por fim, agradeço ao ICMC-USP e ao Instituto Federal de Educação, Ciência e Tecnologia do Sul de Minas Gerais (IFSULDEMINAS) do campus Poços de Caldas, diretamente responsáveis pela possibilidade de realização dessa pesquisa.

“A sabedoria do humilde levantará a sua cabeça e o fará sentar-se no meio dos grandes.”
(Ec.11:1)

RESUMO

OLIVEIRA, R. R. **Avaliação da portabilidade entre fornecedores de teste como serviço na computação em nuvem**. 2018. 229 p. Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2018.

O processo de automatização de teste de software possui alto custo envolvido em sistemas de larga escala, pois exigem cenários de teste complexos e tempos de execução extremamente longos. Além disso, cada etapa do processo de teste requer recursos computacionais e um tempo considerável para a execução de muitos casos de teste, tornando-se um gargalo para as empresas de Tecnologia da Informação (TI). Neste contexto, os benefícios e oportunidades oferecidos pela combinação da computação em nuvem com o Teste como Serviço (*Testing as a Service*, TaaS), que é considerado um novo modelo de negócio e de serviço atraente e promissor, podem proporcionar um impacto positivo na redução do tempo de execução dos testes de maneira custo-efetiva e aumentar o retorno sobre o investimento ou *Return on investment* (ROI). Todavia, existe o problema de *vendor lock-in*, que é o aprisionamento do usuário à plataforma de um fornecedor específico ou serviço de teste, ocasionado pela dificuldade de migrar de um fornecedor TaaS para outro, limitando a utilização dessas novas tecnologias de maneira efetiva e eficiente, impedindo assim, a ampla adoção do TaaS. Como os estudos existentes não são rigorosos ou conclusivos e, principalmente, devido à falta de evidência empírica na área de serviço de teste, muitas questões devem ser investigadas na perspectiva da migração entre os provedores de TaaS. O objetivo deste trabalho é reduzir o impacto ocasionado pelo problema de *vendor lock-in* no processo de automatização de testes na computação em nuvem, na escrita, configuração, execução e gerenciamento dos resultados de testes automatizados. Neste contexto, foi desenvolvido o protótipo da abordagem intitulada Multi-TaaS por meio de uma biblioteca Java como prova de conceito. A abordagem Multi-TaaS é uma camada de abstração e a sua arquitetura permite abstrair e flexibilizar a troca de fornecedores de TaaS de forma portátil, pois permite encapsular toda a complexidade da implementação do engenheiro de software ao desacoplar o teste automatizado de qual plataforma TaaS ele será executado, bem como abstrair os aspectos da comunicação e integração entre as APIs REST proprietárias dos diferentes fornecedores de TaaS. Além disso, a abordagem Multi-TaaS possibilita também sumarizar os resultados dos testes automatizados de forma independente das tecnologias da plataforma TaaS subjacente. Foram realizadas avaliações comparativas da eficiência, efetividade, dificuldade e do esforço de migração entre as abordagens Multi-TaaS e abordagem convencional, por meio de experimentos controlados. Os resultados deste trabalho indicam que a nova abordagem permite facilitar a troca do serviço de teste, melhorar a eficiência e, principalmente, reduzir o esforço e os custos de manutenção na migração entre fornecedores de TaaS. Os estudos realizados no experimento controlado são promissores e podem auxiliar os engenheiros de software na

tomada de decisão quanto aos riscos associados ao *vendor lock-in* no TaaS. Por fim, a abordagem Multi-TaaS contribui, principalmente, para a portabilidade dos testes automatizados na nuvem e da sumarização dos resultados dos testes e, conseqüentemente, possibilita que o modelo de serviço TaaS na computação em nuvem seja amplamente adotado, de forma consciente, no futuro.

Palavras-chave: Computação em Nuvem, Teste como Serviço (TaaS), *Vendor Lock-in*, Automa-tização de Testes, Experimento Controlado.

ABSTRACT

OLIVEIRA, R. R. **Evaluation of portability among testing as a service providers in cloud computing**. 2018. 229 p. Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2018.

The automation of software testing involves high costs in large-scale systems, since it requires complex test scenarios and extremely long execution times. Moreover, each of its steps demands computational resources and considerable time for running many test cases, which makes it a bottleneck for Information Technology (IT) companies. The benefits and opportunities offered by the combination of cloud computing and Testing as a Service (TaaS), considered a new business and service model, can reduce the execution time of tests in a cost-effective way and improve Return on Investment (ROI). However, the lock-in problem, i.e., the imprisonment of the user in the platform of a specific vendor or test service caused by the difficult migration from one TaaS provider to another limits the effective use of such new technologies and prevents the widespread adoption of TaaS. As studies conducted are neither rigorous, nor conclusive, and mainly due to the lack of empirical evidence, many issues must be investigated from the perspective of migration among TaaS providers. This research aims at reductions in the impact of the vendor lock-in problem on the automation process of testing in cloud computing, writing, configuration, execution and management of automated test results. The prototype of the Multi-TaaS approach was developed through a Java library as a proof of concept. The Multi-TaaS approach is an abstraction layer and its architecture enables the abstraction and flexibilization of the exchange of TaaS providers in a portable way, once the complexity of the software engineer's implementation can be encapsulated. The two main advantages of Multi-TaaS are the decoupling of the automated test from the TaaS platform on which it will be executed and the abstraction of the communication and integration aspects among the proprietary REST APIs of the different TaaS providers. The approach also enables the summarization of automated test results independently of the underlying TaaS platform technologies. A comparative evaluation between Multi-TaaS and conventional migration approaches regarding the difficulty, efficiency, effectiveness and effort of migration among TaaS providers was conducted through controlled experiments. The results show the approach facilitates the exchange of test service, improves efficiency and reduces the effort and maintenance costs of migration among TaaS providers. The studies conducted in the controlled experiment are promising and can assist software engineers in decision-making regarding the risks associated with vendor lock-in in TaaS. The Multi-TaaS approach contributes mainly to the portability of automated tests in the cloud and summarization of their results. Finally, this research enables also the widespread adoption of the TaaS service model in cloud computing, consciously, in the future.

Keywords: Cloud Computing, Testing as a Service (TaaS), Vendor Lock-in, Test Automation, Controlled Experiment.

LISTA DE ILUSTRAÇÕES

Figura 1 – Modelos de serviço na nuvem: IaaS, PaaS e SaaS	29
Figura 2 – Teste multiplataforma	30
Figura 3 – Nível de flexibilidade e <i>vendor-lock-in</i> na computação em nuvem	36
Figura 4 – Conceitos de teste de software	43
Figura 5 – O modelo de processo de desenvolvimento V-Model	45
Figura 6 – Integração de teste ao desenvolvimento de sistemas	47
Figura 7 – Componentes do Selenium	51
Figura 8 – Funcionamento da ferramenta Selenium	51
Figura 9 – Exemplos de Escalabilidade na Computação em Nuvem	55
Figura 10 – Modelo de componente portátil vertical	57
Figura 11 – Modelo de serviço interoperável horizontal	57
Figura 12 – Exemplo de interação entre atores e modelos de serviços na computação em nuvem	60
Figura 13 – Visão geral do uso de TaaS por meio de testes automatizados	63
Figura 14 – Diagrama UML dos testes e carga de dados da abordagem convencional	69
Figura 15 – Diagrama UML das configurações dos fornecedores de TaaS da abordagem convencional	72
Figura 16 – Abordagens de Federação e <i>multi-cloud</i> para TaaS	75
Figura 17 – Etapas do Processo de Teste na Nuvem	81
Figura 18 – Analogia do Selenium Webdriver com a abordagem Multi-TaaS	83
Figura 19 – Funcionamento básico da arquitetura Multi-TaaS a partir da perspectiva do teste automatizado	86
Figura 20 – Arquitetura da biblioteca Java Multi-TaaS	87
Figura 21 – Diagrama de classes da arquitetura Multi-TaaS	88
Figura 22 – Diagrama de classes dos Padrões de Projeto <i>Abstract Factory</i> e <i>Strategy</i>	92
Figura 23 – Sumarização dos Resultados da execução dos Testes na Nuvem	95
Figura 24 – Diagrama de classes dos Padrões de Projeto <i>Abstract Factory</i> e <i>Strategy</i>	97
Figura 25 – Projetos desenvolvidos para o experimento controlado	105
Figura 26 – Diagrama de classe do Estudo de Caso utilizando a biblioteca Multi-TaaS	106
Figura 27 – Etapas do processo de experimentação	109
Figura 28 – Programa para a geração de <i>Token</i>	120
Figura 29 – Visão geral do processo de migração das tarefas	126

Figura 30 – Histogramas da avaliação da eficiência - IFSULDEMINAS	130
Figura 31 – Histogramas da avaliação da eficiência - UFSCar	130
Figura 32 – Box plot da eficiência - IFSULDEMINAS	131
Figura 33 – Box plots da avaliação da eficiência - UFSCar	131
Figura 34 – Histogramas da avaliação do esforço - IFSULDEMINAS	137
Figura 35 – Box plot do esforço - IFSULDEMINAS	138
Figura 36 – Histogramas da avaliação da Dificuldade - 1º Experimento do IFSULDEMINAS	140
Figura 37 – Histogramas da avaliação da Dificuldade - 1º Experimento do IFSULDEMINAS	141
Figura 38 – Histogramas da avaliação da Dificuldade - 1º Experimento da UFSCar . . .	142
Figura 39 – Histogramas da avaliação da eficiência - UFSCar	143
Figura 40 – Box plots da avaliação da eficiência - UFSCar	143
Figura 41 – Histogramas da avaliação do esforço - 2º Experimento - UFSCar	146
Figura 42 – Box plots da avaliação do esforço - 2º Experimento - UFSCar	146
Figura 43 – Histogramas da avaliação da Dificuldade - 2º Experimento	148
Figura 44 – Histogramas da avaliação da Dificuldade - 2º Experimento	149
Figura 45 – Histogramas da avaliação da Dificuldade - 2º Experimento	150
Figura 46 – Overview of TaaS usage through automated testing	179
Figura 47 – Phases of the Systematic Review (Adapted from Lana et al. (LANA <i>et al.</i> , 2016))	180
Figura 48 – Search String	181
Figura 49 – Distribution of papers in the conduction phase	185
Figura 50 – Distribution of papers selected by database	185
Figura 51 – Publication per country of the authors	186
Figura 52 – Number of selected papers per workshops, journals, symposiums, conferen- ces, book chapters, theses and magazines.	186
Figura 53 – Distribution of primary studies according to categories	188
Figura 54 – Federation and Multi-Cloud Approaches to TaaS (Adapted from Grozev and Buyya (GROZEV; BUYYA, 2014a))	196
Figura 55 – Planejamento das Tarefas do Experimento	202
Figura 56 – Fluxograma das Tarefas do 1º Experimento	203
Figura 57 – Fluxograma das Tarefas do 2º Experimento	204

LISTA DE CÓDIGOS-FONTE

Código-fonte 1 – Exemplo de capacidades do Saucelabs e Browserstack	100
Código-fonte 2 – Exemplo de código de capacidades do Saucelabs (SAMS, 2015) . . .	211
Código-fonte 3 – Exemplo de código de capacidades do Browserstack (SAMS, 2015) .	212
Código-fonte 4 – Exemplo de código de capacidades do Testingbot (SAMS, 2015) . . .	212
Código-fonte 5 – Testes de CRUD de Universidade - Abordagem Multi-TaaS	216
Código-fonte 6 – Testes de CRUD de Universidade - Abordagem Convencional	217
Código-fonte 7 – Teste para geração de relatórios - Abordagem Multi-TaaS	219
Código-fonte 8 – Teste para geração de relatórios - Abordagem Convencional	220
Código-fonte 9 – Arquivo de configuração <code>startup.config</code>	223
Código-fonte 10 – Arquivo de configuração <code>taas.config</code> do Saucelabs	224
Código-fonte 11 – Arquivo de configuração <code>taas.config</code> do Browserstack	224
Código-fonte 12 – Arquivo de configuração <code>taas.config</code> do Testingbot	225
Código-fonte 13 – Arquivo de configuração <code>bd.config</code>	225
Código-fonte 14 – Exemplo de código das capacidades usadas no Saucelabs	227
Código-fonte 15 – Exemplo de código das capacidades usadas no Browserstack	228
Código-fonte 16 – Exemplo de código das capacidades usadas no Testingbot	229

LISTA DE TABELAS

Tabela 1 – Capacidades dos fornecedores de TaaS	65
Tabela 2 – Capacidades em comum dos fornecedores de TaaS	66
Tabela 3 – Mapeamento das APIs REST dos fornecedores de TaaS	68
Tabela 4 – Número de participantes e amostras - Experimento UFSCar	115
Tabela 5 – Configurações das capacidades utilizadas nos testes automatizados em cada uma das tarefas de migração	118
Tabela 6 – Tabela de análises estatísticas realizadas para os experimentos do IFSULDEMINAS e da UFSCar	127
Tabela 7 – Resultados do experimento controlado (eficiência e esforço de migração) - IFSULDEMINAS	128
Tabela 8 – 1º Experimento controlado – Resultados da eficiência (tempo de migração) - UFSCar	128
Tabela 9 – 1º Experimento controlado – Resultados do esforço de migração - UFSCar .	128
Tabela 10 – 2º Experimento controlado – Resultados da eficiência (tempo de migração) - UFSCar	128
Tabela 11 – 2º Experimento controlado – Resultados do esforço de migração - UFSCar .	128
Tabela 12 – Resultados dos testes- <i>t</i> e Wilcoxon para o 1º experimento controlado	133
Tabela 13 – Estimativa, erro padrão e <i>p</i> -valor do teste de significância (IFSULDEMINAS)	134
Tabela 14 – Estimativa, erro padrão e <i>p</i> -valor do teste de significância do 1º experimento controlado	134
Tabela 15 – Estimativa, erro padrão e <i>p</i> -valor do teste de significância (IFSULDEMINAS)	138
Tabela 16 – Resultados da dificuldade do 2º período - IFSULDEMINAS	140
Tabela 17 – Resultados da dificuldade do 4º período - IFSULDEMINAS	141
Tabela 18 – Resultados da dificuldade do 1º experimento - UFSCar	141
Tabela 19 – Resultados dos testes- <i>t</i> e Wilcoxon para o 2º experimento controlado - UFSCar	144
Tabela 20 – Estimativa, erro padrão e <i>p</i> -valor do teste de significância do 2º experimento controlado - UFSCar	144
Tabela 21 – Resultados dos testes- <i>t</i> e Wilcoxon para o 2º experimento controlado - UFSCar	147
Tabela 22 – Estimativa, erro padrão e <i>p</i> -valor do teste de significância do 2º experimento controlado - UFSCar	147
Tabela 23 – Resultados da dificuldade do 2º período - IFSULDEMINAS	148
Tabela 24 – Resultados da dificuldade do 4º período - IFSULDEMINAS	148
Tabela 25 – Resultados da dificuldade do 2º experimento - UFSCar	149

Tabela 26 – Quality Assessment Form	187
Tabela 27 – Dados do 1º experimento controlado - IFSULDEMINAS	205
Tabela 28 – Dados do 1º experimento controlado - UFSCar	206
Tabela 29 – Dados do 2º experimento controlado - UFSCar	208

LISTA DE ABREVIATURAS E SIGLAS

API	<i>Application Programming Interface</i>
COCOMO	<i>COConstructive COst MOdel</i>
CORBA	<i>Common Object Request Broker Architecture</i>
DCOM	<i>Distributed Component Object Model</i>
DSL	<i>Domain Specific Languages</i>
GoF	<i>Gang of Four</i>
HTML	<i>HyperText Markup Language</i>
HTTP	<i>Hypertext Transfer Protocol</i>
IaaS	<i>Infrasctruture as a Service</i>
IP	<i>Internet Protocol</i>
J2EE	<i>Java 2 Platform, Enterprise Edition</i>
LOC	<i>Lines Of Code</i>
MDE	<i>Model-Driven Engineering</i>
OCCI	<i>Open Cloud Computing Interface</i>
OWL	<i>Web Ontology Language</i>
PaaS	<i>Platform as a Service</i>
QA	<i>Quality Assurance</i>
REST	<i>Representational State Transfer</i>
RMI	<i>Remote Method Invocation</i>
ROI	<i>Return On Investment</i>
RPC	<i>Remote Procedure Call</i>
RTaaS	<i>Regression Testing as a Service</i>
SaaS	<i>Software as a Service</i>
SCJP6	<i>Sun Certified Java Programmer SE 6</i>
SOA	<i>Service-Oriented Architecture</i>
SOAP	<i>Simple Object Access Protocol</i>
SUT	<i>System Under Test</i>
TaaS	<i>Testing as a Service</i>
TCP	<i>Transmission Control Protocol</i>
UAT	<i>User Acceptance Tests</i>
WSDL	<i>Web Services Description Language</i>
WWW	<i>World Wide Web</i>

SUMÁRIO

1	INTRODUÇÃO	27
1.1	Contextualização	27
1.2	Motivação	32
1.3	Definição do problema e justificativa para a pesquisa	35
1.4	Objetivos do Trabalho	36
1.5	Principais Contribuições	38
1.6	Estrutura da Tese	39
2	REVISÃO BIBLIOGRÁFICA	41
2.1	Considerações Iniciais	41
2.2	Teste de Software	41
2.2.1	<i>Verificação e Validação</i>	<i>41</i>
2.2.2	<i>Fundamentos do Teste de Software</i>	<i>42</i>
2.2.3	<i>Fases de Teste de Software</i>	<i>43</i>
2.2.4	<i>Caso de Teste, Conjunto de Teste e Oráculo</i>	<i>46</i>
2.2.5	<i>Critérios e Técnicas de Teste de Software</i>	<i>46</i>
2.2.6	<i>Automatização de Testes de Aceitação do Usuário (UAT)</i>	<i>48</i>
2.2.6.1	<i>Ferramentas da Selenium</i>	<i>50</i>
2.3	Padrões de Projeto	52
2.4	Fundamentos da Computação em Nuvem	53
2.5	Teste de Software na Computação em Nuvem	57
2.5.1	<i>Teste como Serviço (TaaS)</i>	<i>58</i>
2.5.1.1	<i>Consumidores de Teste como Serviço</i>	<i>59</i>
2.5.1.2	<i>Fornecedores de Teste como Serviço</i>	<i>60</i>
2.5.1.2.1	Fornecedor de TaaS - Saucelabs	61
2.5.1.2.2	Fornecedor de TaaS - Browserstack	62
2.5.1.2.3	Fornecedor de TaaS - Testingbot	62
2.5.1.3	<i>Funcionamento do Teste como Serviço (TaaS)</i>	<i>62</i>
2.5.2	<i>Mapeamento das capacidades dos Fornecedores de TaaS</i>	<i>64</i>
2.5.3	<i>Mapeamento das API REST dos Fornecedores de TaaS</i>	<i>66</i>
2.6	Exemplo de Abordagem Convencional de Migração entre Fornece- dores de TaaS	67
2.6.1	<i>Implementação da Abordagem Convencional - 1ª Parte</i>	<i>69</i>

2.6.2	<i>Implementação da Abordagem Convencional - 2ª Parte</i>	72
2.7	<i>Taxonomia Arquitetônica de Desenvolvimento na Nuvem</i>	74
2.8	<i>Considerações Finais</i>	76
3	ARQUITETURA DE TESTE MULTI-TAAS	77
3.1	<i>Considerações Iniciais</i>	77
3.2	<i>Métricas e estimativas de migração de software</i>	78
3.3	<i>Processo de Teste na Nuvem</i>	80
3.4	<i>O Problema de Lock-in no contexto de Teste como Serviço</i>	81
3.5	<i>Arquitetura da biblioteca Multi-TaaS</i>	84
3.5.1	<i>Visão Geral da Arquitetura Multi-TaaS</i>	84
3.5.2	<i>Funcionamento da Arquitetura Multi-TaaS</i>	87
3.5.2.1	<i>Escrita dos testes automatizados e da carga e descarga dos dados de teste</i>	87
3.5.2.2	<i>Configurações das capacidades e execução dos testes na nuvem</i>	90
3.5.2.3	<i>Gerenciamento de execução, coleta de informações e sumarização dos resultados de testes</i>	93
3.6	<i>Vantagens e Desvantagens da Biblioteca Multi-TaaS</i>	98
3.7	<i>Procedimentos para utilização da biblioteca Multi-TaaS</i>	102
3.8	<i>Implementação do Protótipo e Estudo de Caso</i>	103
3.9	<i>Considerações Finais</i>	106
4	AVALIAÇÕES EXPERIMENTAIS	109
4.1	<i>Considerações Iniciais</i>	109
4.2	<i>Experimentos Controlados</i>	110
4.2.1	<i>Definição e Planejamento do Estudo</i>	111
4.2.2	<i>Objetivos</i>	111
4.2.3	<i>Metas</i>	111
4.2.4	<i>Hipóteses</i>	112
4.2.5	<i>Variáveis do Experimento</i>	113
4.2.6	<i>Seleção dos Indivíduos - IFSULDEMINAS</i>	114
4.2.7	<i>Seleção dos Indivíduos - UFSCar</i>	114
4.2.8	<i>Tarefas Experimentais do 1º Dia de Curso</i>	116
4.2.9	<i>Tarefas Experimentais do 2º Dia de Curso</i>	118
4.2.10	<i>Descrição da Instrumentação do Experimento</i>	121
4.2.11	<i>Validade dos Experimentos</i>	122
4.3	<i>Preparação e Configuração dos Ambientes</i>	123
4.3.1	<i>O Ambiente de Migração - IFSULDEMINAS</i>	124
4.3.2	<i>O Ambiente de Migração - UFSCar</i>	124
4.4	<i>Execução dos Experimentos</i>	125

4.5	Análise dos Resultados e Avaliação das Hipóteses - IFSULDEMI-NAS e UFSCar	126
4.5.1	<i>Análise Geral da Efetividade</i>	129
4.5.2	<i>Análises Estatísticas do 1º Experimento (Escrita, Configuração e Execução de Testes)</i>	129
4.5.3	<i>Análises Estatísticas do 2º Experimento (Gerenciamento, Coleta de informações e Sumarização dos Resultados)</i>	142
4.6	Considerações Finais	149
5	CONCLUSÕES	151
5.1	Conclusão Geral	151
5.2	Principais Contribuições	154
5.3	Produção Científica	155
5.4	Dificuldades e Limitações	156
5.5	Trabalhos Futuros	158
	REFERÊNCIAS	161
APÊNDICE A	REVISÃO SISTEMÁTICA SOBRE TESTE COMO SERVIÇO (TAAS)	175
A.1	A Research Roadmap of Vendor Lock-in Solutions to Testing as a Service (TaaS)	175
A.2	A Research Roadmap of Vendor Lock-in Solutions to Testing as a Service (TaaS)	176
A.2.1	<i>Abstract</i>	176
A.2.2	<i>Introduction</i>	177
A.2.3	<i>Testing as a Service</i>	178
A.2.4	<i>Systematic Review Process</i>	180
A.2.4.1	<i>Phase 1 - Planning</i>	180
A.2.4.1.1	Research Questions	180
A.2.4.1.2	Search String and Strategy	181
A.2.4.1.3	Control papers selected	181
A.2.4.1.4	Inclusion and Exclusion criteria	182
A.2.4.1.5	Paper Quality Assurance	183
A.2.4.1.6	Selection of Primary Studies	183
A.2.4.1.7	Data Extraction Process	183
A.2.4.2	<i>Phase 2 - Conducting</i>	184
A.2.4.2.1	General analysis of the Included Studies	184
A.2.4.2.2	Quality Assessment	186
A.2.4.3	<i>Phase 3 - Presentation of Results</i>	187

A.2.4.3.1	<i>RQ₁</i>	187
A.2.4.3.2	<i>RQ₂</i>	192
A.2.4.3.3	Research Agenda	194
A.2.5	<i>Ideas for new solutions to lock-in of TaaS</i>	195
A.2.6	<i>Threats to Validity</i>	196
A.2.7	<i>Related Work</i>	197
A.2.8	<i>Concluding Remarks and Future Work</i>	198
APÊNDICE B	DOCUMENTOS COMPLEMENTARES	201
B.1	Fluxogramas dos Experimentos	201
APÊNDICE C	TABELAS DOS DADOS DOS EXPERIMENTOS	205
C.1	Tabelas de Dados dos Experimentos Controlados	205
APÊNDICE D	CÓDIGOS DAS CAPABILITIES	211
D.1	Códigos das Capacidades	211
APÊNDICE E	TESTES AUTOMATIZADOS DA ABORDAGEM MULTI-TAAS E CONVENCIONAL	215
E.1	Automatização de Testes das abordagens Multi-TaaS e Convencional	215
APÊNDICE F	ARQUIVOS DE CONFIGURAÇÃO DA ARQUITETURA MULTI-TAAS	223
F.1	Arquivos de Configuração da Arquitetura Multi-TaaS	223
APÊNDICE G	CONFIGURAÇÃO DAS CAPABILITIES UTILIZADAS NA ABORDAGEM CONVENCIONAL	227
G.1	Configuração das Capacidades utilizadas na Abordagem Convencional	227

INTRODUÇÃO

1.1 Contextualização

A computação em nuvem surgiu como um novo paradigma, utilizado para se referir a um modelo de computação em rede, no qual o software é executado em um ou mais servidores em substituição a um dispositivo de computação local, como um computador, *tablet* ou *smartphone*. O que torna esta ideia interessante é que ela propõe a transformação da computação em um serviço como outro qualquer, como água e energia elétrica. As principais ideias por trás deste conceito são permitir que as empresas possam adquirir recursos de computação sob demanda por meio da internet e permitir o pagamento de acordo com o volume de utilização (ARMBRUST *et al.*, 2009).

O teste de software é uma atividade desafiadora para muitos projetos de engenharia de software, especialmente para sistemas de larga escala (PRIYANKA; CHANA; RANA, 2012), pois exigem cenários de teste complexos. Além disso, a quantidade de casos de testes pode variar de algumas centenas a milhares, exigindo recursos computacionais significativos e tempos de execução extremamente longos, aumentando significativamente os custos envolvidos no processo de teste na nuvem. A computação em nuvem oferece os recursos de hardware virtualizado, armazenamento ilimitado de forma eficaz e serviços de software, que podem contribuir para a redução do tempo de execução de grandes conjuntos de testes de maneira efetiva, além de aumentar o retorno sobre o investimento, *Return On Investment* (ROI).

Novos horizontes de oportunidade estão se abrindo para o teste de software, a partir da computação em nuvem. Os testes baseados em nuvem têm o potencial de oferecer uma atraente combinação de menores custos, pagamento por uso e eliminação de despesas de capital inicial com infraestrutura, licenças de ferramentas, configuração e manutenção de ambientes de teste. A redução de custos é o principal fator que influencia as empresas na adoção da nuvem. A computação em nuvem não só mudou a forma de obtenção de recursos de computação, como

também a forma de gerenciar e, principalmente, entregar serviços de computação, tecnologias e soluções (SHAWISH; SALAMA, 2014).

Levando-se em consideração as informações apresentadas, a definição mais aceita para descrever a computação em nuvem é composta por sete características fundamentais: três **modelos de serviço** e quatro **modelos de implantação** (MELL; GRANCE, 2009). Os modelos de serviço são compostos por:

- **Software como Serviço ou *Software as a Service* (SaaS):** modelo de entrega que provê o fornecimento de aplicativos de software/serviços à nuvem para serem consumidos sob demanda. Exemplos: Netflix¹, Google Apps², Facebook³, Oracle⁴, Salesforce⁵, entre outros;
- **Plataforma como Serviço ou *Platform as a Service* (PaaS):** modelo de entrega que permite utilizar a infraestrutura de nuvem para desenvolver e implantar novas aplicações próprias ou para prover suporte para nível de SaaS. Exemplos: Microsoft Azure⁶, Heroku⁷, CloudFoundry⁸, OpenShift⁹, entre outros;
- **Infraestrutura como Serviço ou *Infrasctruture as a Service* (IaaS):** modelo de entrega que oferece recursos computacionais como processamento, armazenamento, rede e etc. Exemplos: Amazon¹⁰, IBM SmartCloud¹¹, RackSpace¹², entre outros.

Os três modelos de serviço estão representados na Figura 1 a seguir.

Além dos modelos de entrega, a computação em nuvem pode ser classificada também quanto ao tipo do modelo de implantação: pública, privada, comunitária e híbrida, conforme descrito a seguir (GAO; BAI; TSAI, 2011; ARMBRUST *et al.*, 2009; MELL; GRANCE, 2009; RIMAL; CHOI; LUMB, 2009):

- **Nuvens privadas:** são nuvens internas baseadas em uma rede privada por trás de um *firewall* de uso exclusivo de uma organização;
- **Nuvens públicas:** são as nuvens com acesso disponibilizado ao público em geral por meio da internet;

¹ Netflix: <https://www.netflix.com/>

² Google Apps: <http://www.google.com/intx/pt-BR/enterprise/apps/business/>

³ Facebook: <https://www.facebook.com/>

⁴ Oracle: <https://www.oracle.com/cloud/saas.html>

⁵ Salesforce: <http://www.salesforce.com/br/>

⁶ Windows Azure: <http://www.windowsazure.com/>

⁷ Heroku: <http://www.heroku.com/>

⁸ CloudFoundry: <http://cloudfoundry.org/index.html>

⁹ OpenShift: <https://www.openshift.com/>

¹⁰ Amazon: <http://aws.amazon.com/pt/ec2/>

¹¹ IBM SmartCloud: <http://www.ibm.com/cloud-computing/>

¹² RackSpace: <http://www.rackspace.com/pt/>

Figura 1 – Modelos de serviço na nuvem: IaaS, PaaS e SaaS

Fonte: Vitormeriat (2011)

- **Nuvens híbridas:** prevê uma utilização mista e integrada, ou seja, a combinação de serviços de nuvens públicas com recursos de nuvens privadas; e
- **Nuvens Comunitárias:** são as nuvens compartilhadas por organizações com interesses comuns.

Devido aos diferentes modelos de implantação como nuvem pública, privada, híbrida e comunitária, são necessários diferentes tipos de teste de software. Cada etapa do desenvolvimento na nuvem possui diferentes requisitos de teste. Dessa forma, os testes convencionais não são suficientes em um ambiente de nuvem. As empresas precisam adotar uma abordagem de teste de ponta a ponta, ou seja, desde os requisitos até a implantação do sistema web no ambiente de nuvem (BALARAJU, 2011).

Os engenheiros de software precisam compreender a forma como os diferentes tipos de teste de softwares, no ambiente de computação em nuvem, impactam nos negócios das empresas para identificar quais abordagens de testes devem ser utilizadas em cada situação. Torna-se necessário complementar os testes tradicionais com os testes na nuvem (BALARAJU, 2011).

Testar um novo software muitas vezes requer dispositivos de alto custo apenas por um tempo limitado, tais como: servidores, armazenamento e rede (KIM *et al.*, 2009). Estes recursos de computação são subutilizados após o teste, incorrendo, assim, um custo extra no orçamento. Esse é o cenário no qual o teste na nuvem surgiu como uma nova abordagem para o teste de software, em que a otimização dos recursos computacionais no ambiente de nuvem são aproveitados para reduzir significativamente os custos (PRAKASH, 2010).

Em essência, o teste na nuvem é uma forma de teste de software em que o teste é executado utilizando-se os recursos, máquinas ou servidores da infraestrutura de nuvem. Um dos objetivos principais do Teste como Serviço ou *Testing as a Service* (TaaS) em um ambiente de nuvem é testar um determinado software oferecendo as mesmas vantagens da nuvem e

trazendo consigo os mesmos benefícios que a nuvem traz para os seus clientes (KATHERINE; ALAGARSAMY, 2012).

A quantidade de diferentes combinações e variações de múltiplos sistemas operacionais, dispositivos e navegadores (*cross browser*), todos com as suas respectivas versões, conforme ilustrado na Figura 2, exigem alto custos de investimento em infraestrutura de servidores, memória e processamento para a execução local dos testes automatizados. É evidente que a hospedagem convencional não funcionaria em tal cenário devido aos recursos pré-estabelecidos, oferecidos por esses tipos de serviços, ocasionando, assim, uma receita exorbitante.

Em vez disso, os testes multiplataforma na nuvem são abrangentes e adquirem uma importância primordial neste sentido (BALARAJU, 2011), possibilitando aos seus usuários o privilégio de escolher as suas respectivas configurações multiplataforma e alugar ou terceirizar a execução dos seus testes, além de possibilitar o pagamento sob a demanda.

Figura 2 – Teste multiplataforma

Fonte: Blokland, Mengerink e Pol (2013)

Atualmente, o teste na nuvem tem sido amplamente utilizado para aplicar testes de estresse, carga de trabalho e desempenho. Avanços recentes no campo dos testes funcionais, tais como testes unitários, testes de integração, testes de sistema e os Testes de Aceitação do Usuário ou *User Acceptance Tests* (UAT) estão sendo alvo de pesquisas acadêmicas e empresariais (AKERELE; RAMACHANDRAN; DIXON, 2013).

O Teste como Serviço é um tipo de teste na nuvem considerado como um novo modelo de negócios e serviços. Surgiu a partir dos benefícios oferecidos pela otimização de recursos da infraestrutura do ambiente de nuvem, como por exemplo, o pagamento conforme a demanda de execução de testes (GAO; BAI; TSAI, 2011; GAO *et al.*, 2014). O TaaS também pode ser

definido como uma ferramenta de teste eficiente e efetiva, oferecida como serviço por meio da Internet, sendo muito utilizada para testar aplicações web e *mobile* (GAO; BAI; TSAI, 2011; GAO *et al.*, 2014). Além disso, o TaaS permite garantir a qualidade do teste de software e atender aos padrões de qualidade, requisitos e necessidades dos seus consumidores.

Segundo Gao *et al.* (2013), as principais vantagens do uso do modelo de Teste como Serviço (TaaS) são: a redução de custos, ambiente de teste escaláveis, testes automatizados sob demanda com alta disponibilidade, modelo de pagamento por uso (*pay-per-use*) entre outras diversas vantagens. No modelo TaaS é muito utilizado o termo *pay-as-you-test* e o pagamento geralmente ocorre por tempo e quantidade de testes automatizados executados no ambiente de nuvem.

Outro benefício de testes na nuvem, também a ser lembrado, é que a organização é cobrada sobre uma base de *pay-as-you-go*, ou pagamento por uso, pelo fornecedor de nuvem. A escalabilidade da plataforma “alugada” possibilita que o preço cobrado pelo serviço varie linearmente com a quantidade e tempo de uso da plataforma de nuvem, possibilitando, dessa forma, que a empresa fornecedora reduza o desperdício de recurso quando a infraestrutura está ociosa. Essa característica permite reduzir o tempo e esforço despendido pela equipe de *Quality Assurance* (QA), responsável por criar o ambiente adequado de desenvolvimento e teste, o que melhora consideravelmente o tempo de entrega do produto de software desenvolvido ao mercado (AKERELE; RAMACHANDRAN; DIXON, 2013).

No entanto, entre os desafios que impedem a ampla adoção dessa nova tecnologia está o problema de *vendor lock-in*. Esse problema é ocasionado pela falta de padronização, portabilidade (capacidade de mover ativos) e interoperabilidade (capacidade de permutar ativos) entre as plataformas de TaaS (PETCU *et al.*, 2013; DILLON; WU; CHANG, 2010; FERRY *et al.*, 2013; ARMBRUST *et al.*, 2009; SILVA; ROSE; CALINESCU, 2013a). Desse modo, o *vendor lock-in* aprisiona seus clientes, estabelecendo, uma dependência muito forte do consumidor com as tecnologias proprietárias do fornecedor do serviço de teste na nuvem. Essa dependência dificulta a troca de um fornecedor TaaS para outro, principalmente devido aos altos custos envolvidos no processo de migração.

O problema de *vendor lock-in* é uma séria ameaça ao teste de software na nuvem e pode tornar-se crítico quando um provedor decide aumentar de repente os preços ou mostrar sérios problemas de disponibilidade técnica. Entre as principais causas do *vendor lock-in* no contexto de TaaS, estão a falta de padronização das capacidades e a *Application Programming Interface* (API) ou APIs proprietárias dos serviços de teste baseadas em *Representational State Transfer* (REST), como por exemplo, a API REST dos fornecedores de TaaS Saucelabs¹³, Browserstack¹⁴ e Testingbot¹⁵. Além disso, o risco de *vendor lock-in* e a forte dependência de um consumidor a

¹³ API REST Saucelabs: <<https://docs.saucelabs.com/reference/rest-api/>>

¹⁴ API REST Browserstack: <<https://www.browserstack.com/automate/rest-api/>>

¹⁵ API REST Testingbot: <<https://testingbot.com/support/api>>

um determinado fornecedor de serviço de teste impedem que as novas tecnologias de teste na nuvem sejam amplamente adotadas.

1.2 Motivação

Atualmente, em decorrência do paradigma de computação em nuvem, estão ocorrendo diversas mudanças nas metodologias de desenvolvimento de software. O número de aplicações hospedadas na nuvem aumenta a cada dia, em razão da forma como os recursos computacionais são fornecidos e utilizados (KRISHNA; JAYAKRISHNAN, 2013). Neste contexto, a portabilidade, assim como a interoperabilidade, são atributos de qualidade de software que vem ganhando cada vez mais interesse tanto da indústria quanto da comunidade acadêmica. A portabilidade refere-se à capacidade do engenheiro de software transportar uma aplicação de um servidor e implantá-la e executá-la com sucesso em outro servidor da mesma especificação (SNELL; GLOVER, 2003). Já a interoperabilidade refere-se à capacidade de dois ou mais sistemas heterogêneos se comunicarem por meio de serviços para oferecerem uma funcionalidade em comum (SNELL; GLOVER, 2003).

De acordo com Snell e Glover (2003), uma aplicação portátil não é necessariamente interoperável e vice-versa. A diferença sutil entre estes dois atributos de qualidade de software reside no aspecto de comunicação considerado apenas pela interoperabilidade. Apesar do conceito de computação em nuvem ter sido criado a partir da iniciativa e necessidade mercadológica, está recebendo cada vez mais atenção da comunidade científica (ARMBRUST *et al.*, 2009). Neste contexto, existem, ainda, diversas questões de pesquisa em aberto relacionadas ao tema (ARMBRUST *et al.*, 2009; GAO *et al.*, 2013). Uma dessas lacunas de pesquisa está relacionada à falta de portabilidade e interoperabilidade na computação em nuvem, ocasionada pelo problema de *vendor lock-in*.

O problema de *vendor lock-in* ocorre principalmente devido às tecnologias proprietárias e às diferentes interfaces adotadas pelos provedores. Estas diferenças dificultam a portabilidade e interoperabilidade na nuvem (HOFMANN; WOODS, 2010; ROCHWERGER *et al.*, 2009; RODERO-MERINO *et al.*, 2010). Como a portabilidade e interoperabilidade são extremamente importantes para a adoção da nuvem, diversas abordagens têm sido propostas para viabilizá-las. Além disso, existe um esforço significativo por parte dos profissionais, pesquisadores, órgãos competentes de normatização (FORUM, 2014) e até mesmo de governos (ROCHWERGER *et al.*, 2009) na formulação de políticas que permitam a ampla adoção da nuvem (VMWARE, 2014; BUYYA; RANJAN; CALHEIROS, 2010). Devido ao grande número de soluções resultantes, um dos maiores desafios tem sido compreender e identificar as principais contribuições e limitações de cada solução proposta.

Apesar de qualquer fase da atividade de teste funcional poder ser executada no ambiente de computação em nuvem, a fase mais utilizada pelos fornecedores de TaaS do mercado é a

última fase do processo de ciclo de vida do teste de software, conhecida como Teste de Aceitação do Usuário (UAT). Os testes UAT visam verificar as necessidades dos usuários e satisfazer os requisitos e processos de negócios (HAMBLING; GOATHEM, 2013). Devido ao grande número de diferentes combinações de plataformas, a execução dos testes UAT multiplataformas exige um maior custo com infraestrutura de servidores e, conseqüentemente tem despertando o interesse de pesquisa para a sua automatização na redução de custos, tanto por parte da academia quanto da indústria.

Jamshidi, Ahmad e Pahl (2013) realizaram uma revisão sistemática com objetivo de identificar, classificar e comparar sistematicamente as pesquisas existentes sobre migração para a nuvem. O estudo revelou que as pesquisas sobre migração no ambiente de computação em nuvem encontram-se em estágios iniciais, sendo necessários trabalhos envolvendo especialmente a migração com estudos de caso do mundo real. Enquanto o foco dessa revisão está na migração de sistemas legados para a nuvem, esta tese de doutorado revela lacunas no campo de migração entre fornecedores de TaaS no ambiente de nuvem, pois existem diversos outros trabalhos que apenas citam o problema de *vendor lock-in* entre diferentes contextos (SILVA; ROSE; CALINESCU, 2013a; SATZGER *et al.*, 2013; SILVA; ROSE; CALINESCU, 2013b; LOUTAS *et al.*, 2011), porém somente Gao *et al.* (2013) revelam o problema de *lock-in* no contexto de TaaS como uma questão de pesquisa em aberto.

Inçki, Ari e Sozer. (2012) apresentam na sua revisão sistemática da literatura uma classificação dos estudos de investigação em curso e identificaram na literatura uma lacuna em relação aos testes de aceitação do usuário como uma área de pesquisa em aberto para os testes na nuvem. Já Silva, Rose e Calinescu (2013a) classificam e relacionam as principais questões sobre o problema de *lock-in*.

Nguyen, Boehm e Danphitsanuphan (2009) descrevem um experimento controlado conduzido com estudantes para avaliar a produtividade e estimar o esforço de três diferentes tipos de manutenção em um programa C++ e projetou um modelo de estimativa que prevê o esforço necessário para as tarefas de manutenção. Os resultados do estudo sugerem que é mais caro modificar do que adicionar ou excluir um *Lines Of Code* (LOC). No entanto, os autores se concentraram apenas na manutenção de aplicações locais, enquanto a migração no ambiente de nuvem deve levar em consideração características específicas, como as interações entre o consumidor e fornecedor.

Miranda *et al.* (2013), Miranda *et al.* (2012) apresentam uma solução para o problema de *vendor lock-in* a partir da Engenharia Dirigida por Modelos, *Model-Driven Engineering* (MDE) baseada em técnicas de adaptação de software como uma abordagem alternativa para o desenvolvimento e manutenção de softwares na nuvem. O foco principal da abordagem é em aplicações na nuvem baseadas em PaaS, embora seu método de migração também possa ser estendido para aplicações baseadas em IaaS. Tran *et al.* (2011) descrevem os principais fatores que afetam a migração de uma aplicação local para a nuvem e propõem uma taxonomia de

tarefas de migração para auxiliar a tomada de decisão dos engenheiros de software na avaliação do uso da computação em nuvem. Embora esses estudos se concentrem nos modelos PaaS e IaaS, eles não levam em consideração o modelo SaaS, como observado anteriormente, todo TaaS é um SaaS com características específicas.

Kolb, Lenhard e Wirtz (2015) realizaram uma avaliação comparativa do processo de migração entre sete plataformas de nuvem, levando em consideração o esforço de implantação quanto à duração e ao número de etapas necessárias para implantar uma mesma aplicação em diferentes plataformas de nuvem. Ao contrário deste trabalho, os autores não levam em consideração aspectos humanos e ressaltam que esses fatores são dificilmente quantificáveis e exigem mais estudos empíricos (KOLB; LENHARD; WIRTZ, 2015).

Huang *et al.* (2011) apresentam como projetar e implementar uma plataforma de teste de regressão como serviço denominada *Regression Testing as a Service* (RTaaS). Candea, Bucur e Zamfir (2010) desenvolveram uma abordagem de teste de software automatizado como um serviço baseado em nuvem. Yu *et al.* (2009) mostram como construir uma plataforma de testes web para apoiar os Testes de Aceitação do Usuário.

Petcu (2011) lista as principais abordagens para solucionar o problema de *lock-in* com base na interoperabilidade e portabilidade da computação em nuvem, sendo elas: APIs abertas, protocolos abertos, normas ou padrões, camadas de abstrações, repositórios semânticos e Linguagens Específicas de Domínio ou *Domain Specific Languages* (DSL).

Além desses, existem diversos outros trabalhos que apenas citam o problema de *vendor lock-in* em diferentes contextos (PETCU, 2011; SILVA; ROSE; CALINESCU, 2013a; SATZGER *et al.*, 2013; SILVA; ROSE; CALINESCU, 2013b; RAZAVIAN *et al.*, 2013; BERMBACH *et al.*, 2011; LEWIS, 2012; CUNHA; MELO; SILVA, 2013), porém somente Gao *et al.* (2013) citam o problema de *lock-in* no contexto de Teste como Serviço, como uma questão de pesquisa em aberto. İnçki, Ari e Sozer. (2012) apresentam, na sua revisão sistemática da literatura, uma classificação dos estudos de investigação em curso. Eles identificaram na literatura uma lacuna em relação aos Testes de Aceitação do Usuário como uma área de pesquisa em aberto para os testes na nuvem.

Em todos os trabalhos relacionados ao problema de *vendor lock-in* observa-se, de modo geral, que as soluções propostas estão voltadas para a computação em nuvem. Em contraste, a abordagem deste trabalho de doutorado está voltada especificamente em reduzir os impactos ocasionados pelo problema de *vendor lock-in* no processo de teste na nuvem. Apesar de existirem diferentes modelos de computação em nuvem (por exemplo, SaaS, PaaS, IaaS, TaaS) foi proposto um projeto *open source* conhecido por *Compatible One*¹⁶, com padrões universais, para a computação em nuvem que está em fase de desenvolvimento. Porém são necessários padrões diferentes para cada modelo de serviço, em vez de um conjunto abrangente de padrões (PETCU,

¹⁶ *Compatible One*: <<http://www.compatibleone.org/bin/view/Discover/Overview>>

2011).

Nos últimos anos, tem havido um número crescente de pesquisas envolvendo diferentes abordagens, principalmente destinadas à aumentar a portabilidade e a interoperabilidade, a fim de resolver o problema de *vendor lock-in*, para proteger os investimentos dos consumidores de diferentes modelos de serviços e permitir uma ampla adoção da computação em nuvem (PETCU, 2011).

1.3 Definição do problema e justificativa para a pesquisa

Uma propriedade importante da computação em nuvem refere-se ao termo “*utility computing*”. Essa característica estabelece que a troca de um fornecedor de um determinado serviço não deve impactar o consumidor do serviço. Desse modo, um serviço deve funcionar independentemente do seu provedor e, caso haja alguma troca de fornecedor, essa mudança deve ocorrer de forma transparente para os seus consumidores. Fazendo uma analogia, a eletricidade é consumida de forma independente da fonte de energia, seja ela hidráulica, eólica ou solar (SILVA; ROSE; CALINESCU, 2013a).

O problema de *vendor lock-in* é um dos principais problemas que afetam a propriedade de “*utility computing*” da computação em nuvem, podendo ocorrer tanto nas etapas de desenvolvimento de software como na automatização de testes para ambientes de nuvem.

O *vendor lock-in*, conforme mencionado anteriormente, surge devido à falta de portabilidade, interoperabilidade e padronização entre os diferentes fornecedores de TaaS (ARMBRUST *et al.*, 2009; ARMBRUST *et al.*, 2010), sendo amplamente reconhecido como um dos maiores obstáculos para a adoção da nuvem (DILLON; WU; CHANG, 2010). O *vendor lock-in* é uma preocupação típica devido à dependência e ao aprisionamento do consumidor às tecnologias proprietárias do fornecedor. Esse aprisionamento pode impedir o consumidor TaaS de migrar ou trocar de um fornecedor TaaS para o outro, devido aos altos custos envolvidos (ARMBRUST *et al.*, 2009; ARMBRUST *et al.*, 2010).

Cada fornecedor impõe o seu padrão e adota requisitos de teste incompatíveis entre si, comprometendo a portabilidade dos testes automatizados e do gerenciamento (coleta e sumarização) dos resultados dos testes e da interoperabilidade entre os provedores de TaaS. O *vendor lock-in* representa uma séria ameaça para os consumidores do serviço de teste, podendo haver aumentos abusivos de preço ou problemas de disponibilidade técnica. A principal importância de pesquisar o *vendor lock-in* entre os fornecedores de TaaS ao invés do *vendor lock-in* que acontece em outros modelos de serviços, como PaaS e IaaS na Computação em Nuvem de uma forma geral, é devido às constantes evoluções das aplicações web. Elas precisam ser frequentemente testadas ao longo do ciclo de vida da aplicação, aumentando ainda mais a dependência do consumidor em relação ao fornecedor de TaaS específico com o passar do tempo.

Os usuários da nuvem são, muitas vezes, presos a um provedor de nuvem específico, pois não existem padrões abertos que garantam a portabilidade e a interoperabilidade do serviço de teste entre fornecedores de TaaS (SILVA; ROSE; CALINESCU, 2013a; ARMBRUST *et al.*, 2009). Essa é uma questão importante na construção de um ambiente que utilize testes baseados em nuvem, tanto para softwares SaaS e não SaaS. Uma vez que muitas tecnologias de nuvem existentes utilizam ferramentas, linguagens de programação e APIs proprietárias específicas, é importante para os engenheiros de software considerar como evitar tecnologias *lock-in* na adoção do TaaS (GAO *et al.*, 2013; LOUTAS *et al.*, 2010; ROCHWERGER *et al.*, 2009; RODERO-MERINO *et al.*, 2010; HOFMANN; WOODS, 2010);

Um aspecto relevante é que o modelo de serviço de IaaS oferece um maior grau de flexibilidade e controle para o usuário, ao passo que o SaaS fornece menor grau de flexibilidade e controle. Essa característica contribui para aumentar proporcionalmente o nível de *vendor lock-in* na camada de SaaS em relação às camadas inferiores, como a da camada IaaS, exemplificada pela Figura 3. Cabe ressaltar, que todo TaaS é um SaaS.

Figura 3 – Nível de flexibilidade e *vendor-lock-in* na computação em nuvem

Fonte: Karatas *et al.* (2012)

1.4 Objetivos do Trabalho

O objetivo deste trabalho de doutorado foi investigar a portabilidade referente à migração entre diferentes fornecedores de Teste como Serviço (TaaS) na Computação em Nuvem. Para alcançar o objetivo estabelecido, foram desenvolvidos dois protótipos que serviram como provas de conceito, sendo eles, a abordagem tradicional ou convencional e a abordagem proposta, intitulada Multi-TaaS.

A abordagem Multi-TaaS permite desacoplar o teste automatizado de qual plataforma TaaS ele será executado, bem como abstrair os aspectos da comunicação e integração entre as APIs REST proprietárias dos diferentes fornecedores de TaaS. Além disso, a abordagem Multi-TaaS possibilita sumarizar os resultados dos testes automatizados independentemente das tecnologias da plataforma TaaS subjacente.

É importante destacar também que a abordagem Multi-TaaS foi desenvolvida com base

nas classificações de Grozev e Buyya (2014a) e contribui para reduzir os impactos ocasionados pelo problema de *vendor lock-in* no processo de automatização de teste no ambiente de nuvem: na escrita de testes, configuração de capacidades e execução de testes nas plataformas de TaaS, bem como no gerenciamento, tais como a coleta e sumarização dos resultados dos testes por meio da API REST de cada fornecedor TaaS.

A abordagem Multi-TaaS desenvolvida neste doutorado foi avaliada e comparada com a abordagem convencional por meio de experimentos controlados. Por fim, foram realizadas as análises estatísticas para avaliar a qualidade da abordagem proposta em termos de efetividade, eficiência, dificuldade e esforço de migração entre os diferentes fornecedores de TaaS.

As duas principais questões de pesquisa investigadas neste doutorado foram:

1. Como solucionar ou reduzir os impactos ocasionados pelo problema de *vendor lock-in* no contexto de Teste como Serviço (TaaS)?
2. Quais as etapas do processo de automatização de teste na nuvem são afetadas pelo problema de *vendor lock-in*?

Para responder às questões de pesquisa, inicialmente foi realizada uma revisão sistemática, disponível no Apêndice A.1, com objetivo de encontrar e mapear as principais soluções para o problema de *vendor lock-in* no contexto de TaaS. O objetivo dessa revisão sistemática foi ampliar o campo de pesquisa e auxiliar os pesquisadores novatos.

As soluções encontradas na revisão sistemática giram em torno, por exemplo de, MDE, APIs ou padrão/protocolos abertos e camadas de abstração (solução adotada por este trabalho), entre outras. Após realizar todo o processo da revisão sistemática de planejamento, execução e análise dos resultados e, a partir dos principais artigos selecionados, foi possível vislumbrar além das vantagens e desvantagens de cada solução encontrada:

1. Categorizar e classificar as principais soluções adotadas para contornar o problema de *vendor lock-in* no contexto TaaS;
2. Analisar e entender melhor quais soluções do *vendor lock-in* poderiam ser reaproveitadas no contexto de TaaS a partir das soluções existentes de outros modelos de serviço;
3. Identificar as principais lacunas de pesquisas sobre o TaaS, levando em consideração as abordagens de Federação, *Multi-Cloud* e *Intercloud*.

Os principais objetivos alcançados a partir das questões de pesquisa, são descritos a seguir:

- Definição do problema de *vendor lock-in* no contexto de Teste como Serviço (TaaS) e discussão aprofundada do seu impacto nas diferentes etapas do processo de teste de software na computação em nuvem;
- Criação da abordagem *open source* Multi-TaaS por meio de uma arquitetura de teste e desenvolvida a partir de uma biblioteca Java. Cabe ressaltar que a arquitetura Multi-TaaS só será disponibilizada para toda a comunidade da indústria e pesquisadores interessados após a conclusão de todas as publicações derivadas deste trabalho;
- Avaliação comparativa da abordagem Multi-TaaS com a abordagem convencional por meio de experimentos controlados com objetivo de validar as novas descobertas. A pesquisa possibilita também contribuir para minimizar os impactos ocasionados pelo problema de *vendor lock-in* no processo de teste na nuvem, além de favorecer também a ampla adoção do modelo de serviço TaaS na computação em nuvem;
- Elaboração de uma revisão sistemática, disponível no Apêndice A.1, com objetivo de encontrar e mapear as principais soluções para o problema de *vendor lock-in* no contexto de TaaS.

Esta tese estabelece que a utilização da extensão do JUnit, padrões de projeto e o gerenciamento adequado de configurações permite reduzir consideravelmente os impactos ocasionados pelo problema de *vendor lock-in* de forma efetiva e transparente para os consumidores de TaaS.

A extensão do JUnit corresponde ao mecanismo criado para a utilização da abordagem Multi-TaaS com o objetivo de prover o desacoplamento adequado entre o *script* de teste da plataforma TaaS em que ele será executado, além de possibilitar a abstração da comunicação e integração entre consumidor e fornecedor TaaS na geração de relatórios dos resultados da execução dos testes na nuvem, facilitando o trabalho do engenheiro de software.

1.5 Principais Contribuições

A investigação da abordagem Multi-TaaS aplicada no processo de automatização de testes na computação em nuvem possibilita reduzir os impactos ocasionados pelo problema de *vendor lock-in* no contexto de TaaS. Os principais benefícios obtidos com a arquitetura de teste Multi-TaaS são aumentar a flexibilidade com o desacoplamento entre consumidor e fornecedor TaaS, reduzir os custos de manutenção ao reduzir o esforço de migração e consequentemente aumentar a reutilização do código de teste.

Além disso, a abordagem desenvolvida permite aumentar a portabilidade entre os serviços de teste para o Engenheiro de Software, além de facilitar a ampla adoção do modelo de serviço TaaS no ambiente de nuvem por meio das principais contribuições listadas a seguir:

- Apresentação de uma solução para o problema de *vendor lock-in* por meio de um modelo arquitetural de testes automatizados para aplicações web e *mobile* na computação em nuvem;
- Implementação do protótipo da abordagem adotada como prova de conceito, bem como demonstração da sua aplicabilidade e relevância no mundo real.
- Redução do impacto ocasionado pelo problema de *vendor lock-in* no contexto de TaaS por meio de uma abordagem portátil;
- Avaliação e comparação da efetividade, eficiência, dificuldade e esforço de migração entre as abordagens Multi-TaaS e convencional por meio de experimentos controlados;
- Criação de artefatos e empacotamento dos experimentos controlados para que outros pesquisadores interessados possam replicar os experimentos;

Espera-se que a adoção do modelo TaaS na computação em nuvem seja ampliado por meio da redução do esforço na migração de fornecedores de TaaS e no combate ao *vendor lock-in* e permita auxiliar os engenheiros de software e gerentes de controle de qualidade na tomada de decisão em relação aos riscos associados ao problema de *vendor lock-in* e com o menor custo de migração.

1.6 Estrutura da Tese

Este trabalho de doutorado está organizado em seis capítulos, conforme a listagem a seguir.

Este capítulo introduziu a área de pesquisa apresentando a contextualização, motivação, definição do problema de pesquisa, objetivos do trabalho e as principais contribuições deste doutorado.

No Capítulo 2 são apresentados os conceitos fundamentais para o desenvolvimento deste trabalho, sobre teste de software, automação de testes, padrões de projeto, fundamentos da computação em nuvem e Teste como Serviço.

O Capítulo 3 descreve o problema do *vendor lock-in* no contexto de Teste como Serviço e como ele afeta as etapas do processo de testes na nuvem, além de apresentar também a arquitetura da abordagem Multi-TaaS proposta neste trabalho.

No Capítulo 4 são descritas todas as etapas e atividades do processo de condução do experimento controlado necessárias para validar as descobertas deste trabalho.

Por fim, o Capítulo 5 apresenta as conclusões dos resultados obtidos a partir do trabalho desenvolvido, as contribuições da abordagem Multi-TaaS, produções científicas, dificuldades

e limitações encontradas durante o desenvolvimento do projeto de doutorado e finalmente os trabalhos futuros.

REVISÃO BIBLIOGRÁFICA

2.1 Considerações Iniciais

Neste capítulo são apresentados os fundamentos e os principais termos relacionados ao teste de software e automatização de testes no ambiente de nuvem.

Organização do capítulo: na Seção 2.2, são apresentados os principais conceitos, termos e fases relacionadas ao teste de software. A Seção 2.3 descreve os principais conceitos de Padrões de Projeto. Na Seção 2.4 são apresentados os principais conceitos relacionados com a computação em nuvem. Na Seção 2.5 são apresentados os principais conceitos e termos de teste de software na computação em nuvem. A Seção 2.6 descreve o desenvolvimento da abordagem convencional de migração de testes automatizados de aceitação do usuário (UAT) e do mecanismo de sumarização dos resultados dos testes na nuvem. Na Seção 2.7 é apresentada a taxonomia de desenvolvimento de aplicações *intercloud*, federação e *multi-cloud* na computação em nuvem proposta por Grozev e Buyya (2014a). Na Seção 2.8 são apresentadas as considerações finais.

2.2 Teste de Software

2.2.1 Verificação e Validação

No desenvolvimento de um sistema integrado, é importante ser capaz de determinar se o sistema satisfaz as especificações e se as suas saídas estão corretas. Este é o processo de verificação e validação (V & V) e o seu planejamento deve começar no início do ciclo de vida de desenvolvimento (DELAMARO; MALDONADO; JINO, 2016). Existem diversas técnicas diferentes de V & V que são aplicáveis em diferentes estágios do ciclo de vida do software. O desenvolvimento de qualquer sistema de qualidade não é completo sem testes rigorosos e verificação de que a aplicação esteja de acordo com as especificações. De acordo com o glossário

padrão de terminologias de engenharia de software da IEEE (IEEE, 1990), a verificação é definida como “o processo de avaliação de um sistema ou componente para determinar se os produtos de uma determinada fase do desenvolvimento satisfazem as condições impostas no início dessa fase”. A validação, por outro lado, é definida como o processo de avaliação de um sistema ou componente durante ou no final do processo de desenvolvimento, para determinar se ele satisfaz os requisitos especificados. O processo de verificação em si não irá detectar defeitos resultantes da especificação e muito menos da entrada incorreta. Além disso, esses defeitos podem propagar-se sem detecção por fases posteriores do ciclo de desenvolvimento. Finalmente, a certificação é um documento que possibilita garantir que um sistema ou componente está em conformidade com os requisitos especificados e se é aceitável para uso operacional.

Dentre as técnicas de verificação e validação utilizadas para minimizar a ocorrência de defeitos e riscos associados, a atividade de teste é o processo sistemático de avaliar o sistema pela observação de sua execução, sendo também uma das mais utilizadas (AMMANN; OFFUTT, 2008). Os resultados obtidos na atividade de teste podem ser usados nas atividades de manutenção, depuração e estimativa de confiabilidade de software (PRESSMAN, 2009), já que o principal objetivo de um teste é revelar a presença de defeitos no produto de software (SOMMERVILLE, 2010; MYERS; SANDLER; BADGETT, 2004). Desse modo, pode-se afirmar que o teste bem sucedido é aquele que executa casos de teste que fazem o produto falhar.

2.2.2 Fundamentos do Teste de Software

Segundo Sommerville (2010), o teste é destinado a mostrar que um programa faz o que é proposto a fazer. Quando se testa o software, o mesmo é executado com dados reais ou fictícios para revelar defeitos e anomalias em requisitos funcionais, bem como nos requisitos não funcionais do software, antes do uso.

O processo de desenvolvimento de software envolve uma série de atividades nas quais, apesar das técnicas, métodos e ferramentas empregados, erros no produto ainda podem ocorrer. O teste de software é a atividade que visa avaliar um atributo ou capacidade de um programa ou sistema e determinar se ele atende aos seus resultados esperados (DELAMARO; MALDONADO; JINO, 2016). Embora fundamental para qualidade de software e amplamente utilizados por programadores e testadores, o teste de software continua a ser uma arte, devido à limitada compreensão dos princípios do software (DELAMARO; MALDONADO; JINO, 2016).

No contexto de teste de software, o glossário padrão IEEE de terminologias de engenharia de software (DELAMARO; MALDONADO; JINO, 2016; IEEE, 1990) diferencia os seguintes termos:

- **Engano (*mistake*):** ação humana que produz um resultado diferente do esperado, como por exemplo, o ato de incompreender ou não interpretar adequadamente o requisito de software;

- **Falha (*failure*):** quando o sistema não atende aos requisitos e é incapaz de produzir a resposta correta a partir da ação do usuário final;
- **Defeito (*fault*):** passo, processo ou definição de dados de forma inadequada fazendo com que o software se comporte de forma inesperada;
- **Erro (*error*):** discrepância entre o valor obtido e o valor esperado, ou seja, qualquer estado inconsistente ou resultado inesperado que ocorra em tempo de execução do software;

Conforme mostrado na Figura 4, o erro é gerado a partir de um defeito, ou seja, é um estado inconsistente ou inesperado durante a execução do sistema. Por exemplo, um engano cometido pelo desenvolvedor no comando *IF* que gera um desvio de fluxo incorreto ou *loop* infinito. É importante destacar que o estado inconsistente pode levar a uma falha, que é caracterizada por um comportamento inesperado do programa percebido pelo usuário que o executa (DELAMARO; MALDONADO; JINO, 2016). Além dos termos usados para o processo de teste, existe o processo de depuração, que é diferente do processo de teste e tem como principal objetivo encontrar defeitos que causam as falhas. Portanto, a manifestação de uma falha no programa indica que existe um defeito ou mais e consequentemente possibilita dar início a atividade de depuração do código fonte da aplicação.

Figura 4 – Conceitos de teste de software

Fonte: Autoria própria

Como o teste não garante a inexistência de defeitos, é muito importante a criação de casos de teste efetivos, que sejam capazes ou com alta probabilidade de revelar defeitos (PRESSMAN, 2009).

2.2.3 Fases de Teste de Software

As fases de teste são utilizadas para revelar a presença de defeitos em diferentes momentos do processo de desenvolvimento do software, com o objetivo de diminuir a complexidade de teste (DELAMARO; MALDONADO; JINO, 2016). Por exemplo, um defeito pode ser revelado durante a execução do teste em uma unidade do programa ou então nos testes de comunicação entre unidades. Fases do teste de software são descritas a seguir:

- **Teste de Unidade:** é uma atividade que concentra esforços na menor unidade ou componente individual do código fonte do software. No paradigma procedimental essa unidade

pode ser, por exemplo, uma função. Porém, no paradigma de Orientação a Objetos (OO), essa unidade pode ser representada por um método ou uma classe (BINDER, 1999). O teste de unidade busca revelar os defeitos de lógica e de implementação em cada unidade do software de forma separada. Na maioria das vezes é criado pelo próprio desenvolvedor (PRESSMAN, 2009);

- **Teste de Integração:** é uma atividade sistemática aplicada durante a integração da estrutura do software com a finalidade de expor erros nas interfaces entre os módulos e nas interações ou comunicação entre os componentes ou sistemas integrados (PRESSMAN, 2009);
- **Teste de Sistema:** é uma atividade realizada após a integração do sistema, que verifica se ele atende à especificação dos requisitos. Visa exercitar o sistema como um todo por meio de vários tipos de testes: de segurança, de estresse e de desempenho (PRESSMAN, 2009);
- **Testes de Aceitação do Usuário (UAT - *User Acceptance Tests*):** esta atividade simula as operações de rotina do sistema de modo a verificar as necessidades dos usuários, requisitos e processos de negócios. Geralmente, são realizados pelos usuários/*stakeholders* permitindo ao cliente determinar se aceita ou não o sistema. Além disso, permitem estabelecer confiança no sistema e verificar se o comportamento do sistema está de acordo com o solicitado é considerado o estágio final de testes no desenvolvimento de software. Quando os resultados de testes atendem aos critérios de aceitação, o software está pronto para ser liberado (LEUNG; WONG, 1997).

Além das quatro fases explicitadas anteriormente existe, ainda, o teste de regressão, que é uma atividade ou processo de validação do software modificado, para fornecer a confiança de que as partes alteradas se comportarão conforme o esperado e que as partes inalteradas do software não serão prejudicadas por modificações, sendo essencial para a qualidade do software em todo o seu ciclo de vida (HARROLD *et al.*, 2001). É importante ressaltar que o teste de regressão pode ser aplicado em qualquer uma das fases do teste de software.

Como complemento às fases de teste, é possível encontrar modelos de processo que relacionam as fases de desenvolvimento com as fases de teste de software, como por exemplo o processo V-Model (BEAUFTRAGTEN, 1997). Cada fase de teste pode ser aplicada a uma fase do desenvolvimento de software (BEAUFTRAGTEN, 1997). A Figura 5 ilustra o modelo V-Model, na qual, no lado esquerdo, as fases do desenvolvimento de software possuem uma abordagem *top-down*, enquanto que no lado direito as fases de teste têm uma abordagem *bottom-up*.

Cabe ressaltar que o Teste como Serviço (TaaS) será melhor explorado na Seção 2.5, mas em termos gerais, o TaaS pode ser definido como uma ferramenta de teste que é oferecida como um serviço (GAO; BAI; TSAI, 2011), podendo ser utilizada em qualquer metodologia

Figura 5 – O modelo de processo de desenvolvimento V-Model

Fonte: Beauftragten (1997)

de desenvolvimento e aplicada em qualquer parte do ciclo de vida do software, ou seja, é independente da metodologia utilizada no projeto de software.

De acordo com Maldonado (1991) e IEEE (2008) para cada fase de teste é definido um processo de teste que envolve basicamente quatro etapas:

1. **Planejamento de Testes:** etapa de elaboração de um plano de teste fundamental para o gerenciamento do processo de teste.
2. **Projeto dos Casos de Teste:** etapa de desenvolvimento de casos de teste usando a técnica de teste funcional complementada com a técnica de teste estrutural (MYERS; SANDLER; BADGETT, 2004).
3. **Execução do Teste:** etapa de execução dos procedimentos especificados no plano de teste.
4. **Coleta e avaliação dos resultados dos testes:** etapa de sumarização dos resultados de teste realizados em forma de relatórios. Os resultados obtidos são comparados com os resultados esperados por meio do oráculo, que é o testador, ou outro mecanismo, que decide se os valores de saída estão corretos.

2.2.4 Caso de Teste, Conjunto de Teste e Oráculo

O padrão IEEE (1990) define o conceito de caso de teste como um conjunto de entradas de teste, condições de execução e o resultado de saída esperada para aquele teste. Um conjunto de teste é formado pelos casos de teste com o objetivo particular de exercitar determinados caminhos no programa ou validar determinados requisitos.

As falhas de um programa podem ser reveladas com base nos resultados da execução de um conjunto de teste. Por meio da saída esperada é possível determinar se um caso de teste foi executado corretamente de acordo com o resultado esperado. Se o caso de teste falhar, significa que o resultado é diferente do esperado. Cada caso de teste pode executar diferentes caminhos do programa e o resultado pode ser correto ou incorreto. Essas informações determinam quais os trechos do código fonte possuem maior possibilidade de conter defeitos.

A identificação da falha geralmente é determinada pela intervenção humana e depende do conhecimento sobre o comportamento esperado do programa. O oráculo no contexto de teste de software é a entidade responsável pela comparação do resultado esperado com a saída dos casos de teste (DELAMARO M.; CHAIN; VINCENZI, 2010). O oráculo determina quando o caso de teste é executado com sucesso ou falha, exercendo um papel fundamental para as técnicas de localização das informações da execução dos testes. Portanto a cobertura de teste pode ser avaliada como uma porcentagem dos requisitos satisfeitos em relação ao total de requisitos estabelecidos pelo critério de teste. Essa porcentagem pode indicar a necessidade de criação de novos casos de teste para aumentar a cobertura e, conseqüentemente, testar mais trechos do programa (ZHANG; CHENG; BOUTABA, 2010).

Existe também um outro termo, conhecido como cobertura de teste, que avalia a adequação de um conjunto de teste aos requisitos de teste, ou seja, se os requisitos foram satisfeitos por um dado caso de teste ou conjunto de teste (ZHANG; CHENG; BOUTABA, 2010). A cobertura de teste pode ser aplicada aos critérios de diferentes técnicas de teste como o teste estrutural, teste funcional ou teste baseado em defeitos.

2.2.5 Critérios e Técnicas de Teste de Software

Para decidir se o programa foi suficientemente testado, é imprescindível um alto nível de confiança na atividade de teste para a correção do programa, levando-se em conta os custos de teste. É importante destacar que testar todas as entradas possíveis de um software é impraticável. Portanto os critérios de teste são usados para determinar os casos de teste que serão executados (DELAMARO; MALDONADO; JINO, 2016).

De acordo com Delamaro, Maldonado e Jino (2016), um critério de teste ou *critério de adequação de casos de teste*, estabelece que um conjunto de teste T é adequado para testar um programa P , se dado um critério de adequação C , existe pelo menos um método de seleção de casos de teste MC para C que selecione os casos de teste com o objetivo de testar P . Dados P , T

e um critério C , diz-se que o conjunto de casos de teste T é C – *adequado* para o teste de P se T atender os requisitos de teste estabelecidos pelo critério C .

Dentre as diferentes abordagens de integração de testes no desenvolvimento de sistemas, destacam-se as três principais abordagens existentes: a convencional, na qual o sistema é desenvolvido a partir dos requisitos e é testado após seu desenvolvimento; a abordagem concorrente, ou seja, que utiliza os requisitos para desenvolver o sistema e realizar os testes de forma concorrente, por considerar que defeitos encontrados no início do desenvolvimento não têm um alto custo de correção; e a abordagem *test-first* que cria os testes antes do desenvolvimento do sistema, sendo utilizada nas metodologias ágeis. A Figura 6 representa os três tipos de abordagem acima explicitados, que estão representados pelas abordagens da esquerda, centro e direita respectivamente.

Figura 6 – Integração de teste ao desenvolvimento de sistemas

Fonte: Weißleder (2010)

A partir dessas três abordagens, podem ser aplicados os critérios na atividade de teste de software estabelecidos a partir das técnicas de testes, divididas em funcional, estrutural e baseada em defeitos. As técnicas de teste são usadas para verificar diferentes aspectos do programa com o objetivo de revelar os defeitos no comportamento de um programa além de otimizar a quantidade de testes a serem executados. Portanto as técnicas de teste permitem sistematizar todo o processo de teste de software (MALDONADO, 1991).

Na técnica funcional, os critérios e requisitos de teste são estabelecidos por meio da especificação do software; na técnica estrutural, os critérios e requisitos são constituídos a partir das características de uma implementação específica em teste; e na técnica baseada em defeitos, os critérios e requisitos de teste são provenientes do conhecimento sobre defeitos típicos cometidos no processo de desenvolvimento de software.

O teste funcional é uma técnica utilizada para projetar casos de teste que permite analisar as funcionalidades de um software (por exemplo, o que o software faz) sem se ter o conhecimento sobre as estruturas internas do programa (MYERS; SANDLER; BADGETT, 2004). Por este motivo o código fonte da aplicação é considerado uma “caixa preta”, pois para executar os testes, são fornecidas as entradas e avaliadas as saídas geradas para verificar se estão em conformidade com os objetivos especificados. Nessa técnica, os detalhes de implementação não são considerados e o software é avaliado segundo o ponto de vista do usuário (DELAMARO;

MALDONADO; JINO, 2016). Este método de teste pode ser aplicado a quase todas as fases de teste de software: unidade, integração, sistema e aceitação.

Já o teste estrutural, também conhecido como teste “caixa branca”, é um método de teste de software que testa estruturas internas de uma aplicação. No teste estrutural, o testador escolhe as entradas para exercitar os caminhos por meio do código e determinar as saídas apropriadas.

Por fim, a técnica de teste baseada em defeitos utiliza informações sobre os tipos de defeitos mais frequentes no processo de desenvolvimento de software para derivar os requisitos de teste (DEMILLO, 1978; BUDD, 1981).

2.2.6 Automatização de Testes de Aceitação do Usuário (UAT)

Atualmente, as empresas possuem projetos de softwares grandes e complexos que exigem um considerável investimento inicial com teste de software para garantir a confiabilidade e, principalmente, a qualidade do software no geral. Os investimentos na área de teste exigem muito esforço e tempo, impactando, assim, diretamente no custo global do desenvolvimento do software. A automatização de testes permite reduzir o tempo gasto e contribui para reduzir os custos associados com o desenvolvimento do software.

O teste funcional verifica se o sistema satisfaz os seus requisitos funcionais. O Teste de Aceitação do Usuário ou *User acceptance testing* (UAT) é essencial para verificar se as funcionalidades do sistema estão de acordo com a especificação do cliente, sem considerar os detalhes da implementação do sistema (LEUNG; WONG, 1997).

Segundo Sommerville (2010), Graham *et al.* (2008), o Teste de Aceitação do Usuário é um estágio no processo de teste em que os usuários ou clientes fornecem entradas e acompanham o teste do sistema, podendo envolver um processo formal ou informal. O teste de usuário é essencial e muito importante para a confiabilidade, o desempenho, a usabilidade e a robustez de um sistema (SOMMERVILLE, 2010; GRAHAM *et al.*, 2008).

Já o padrão IEEE (1986), define o teste de aceitação como sendo um “teste formal conduzido para determinar se um sistema satisfaz ou não os seus critérios de aceitação, permitindo dessa forma que o cliente determine se deve ou não aceitar o sistema”. Ele é geralmente realizado em todo o sistema ou uma grande parte dele (RICCA *et al.*, 2009). Dentro de processos de desenvolvimento de software iterativos curtos (semanal, bissemanal, etc), se os testes de aceitação são realizados manualmente, haverá muito tempo alocado para o teste em cada iteração. Além disso, os testes de aceitação manual, na maioria dos casos, são entediantes, caros e demorados (HANSEN; HAUGSET, 2009; KROPP; SCHWAIGER, 2009).

Os testes UAT são muito utilizados para testar se o aplicativo está funcionando corretamente, podendo ser realizado manualmente por um testador ou de forma automática com o uso de um software. No entanto, os testes UAT visam automatizar as tarefas de testes manuais, pois as tarefas de testes manuais podem ser demoradas, tediosas, caras e propensas a erros. Em

contrapartida a automatização dos testes permitem realizar um método mais preciso, eficiente e confiável (URBONAS, 2013; HAUGSET; HANSSEN, 2008).

A automatização dos testes de interface do usuário podem ser executados simultaneamente em diferentes plataformas e navegadores. No entanto os testes UAT não detectam erros de renderização de *layout* de interfaces web. A automatização dos testes UAT torna-se importante a partir do momento em que uma equipe de testadores trabalha no mesmo trecho de código para garantir uma boa cobertura da automatização dos testes, sem que um determinado código complexo torne-se domínio de um único testador (URBONAS, 2013).

Os testes de aceitação melhoram o processo global de desenvolvimento (NOLL *et al.*, 2011), além de oferecer um bom indicador do *status* do projeto em relação às expectativas dos clientes (MILLER; COLLINS, 2001; ROGERS, 2004). Eles também possibilitam aumentar a confiança geral das funcionalidades que estão sendo desenvolvidas pela equipe de desenvolvimento (GANDHI *et al.*, 2005). Diversos autores relatam que a automatização de testes de aceitação melhoram a qualidade dos produtos de software (NOLL *et al.*, 2011). Senior e Crispin (2001) afirmam que a automatização dos testes de aceitação ajudam a encontrar defeitos nas fases iniciais e, portanto, permitem evitar o excesso de trabalho no final da iteração. Hanssen e Haugset (2009) também argumentam a favor da automatização dos testes de aceitação com base no fato de que ela contribui para a melhoria da qualidade do produto, permitindo que mais defeitos possam ser detectados antes do lançamento da produção.

Um dos principais objetivos na redução de custos com os testes de aceitação do usuário é automatizar ao máximo possível as tarefas de teste de software, e com isso, reduzir também o custo global do desenvolvimento de software, além de minimizar os erros humanos e facilitar o teste de regressão (AMMANN; OFFUTT, 2008). Durante a geração de casos de teste, é possível também implementar critérios de seleção para automatizar o processo. Porém existem limitações próprias da atividade de teste que dificultam a geração de casos de teste como, por exemplo, decidir se um determinado caminho de um programa é executável ou não e se dois programas são equivalentes. Uma forma de abordar o problema da geração automática é, dado um conjunto de critérios de teste, identificar quais as entradas de teste são necessárias para satisfazer esses critérios (MICHAEL; MCGRAW; SCHATZ, 2001).

Os Testes de Aceitação do Usuário permitem verificar os critérios de aceitação funcionais e não funcionais com o objetivo de garantir que a aplicação entrega valor de negócio, como por exemplo, funcionalidades que os seus usuários finais realmente necessitam (HUMBLE; FARLEY, 2010). Existem diversas vantagens na utilização da automatização dos testes. A maioria delas está relacionada com a repetitividade dos testes e com a velocidade em que os testes podem ser executados. Há uma série de ferramentas *open source* e comerciais disponíveis para ajudar com o desenvolvimento de automação de teste. A Selenium é uma delas e foi escolhida por ser utilizada pela maioria dos fornecedores de TaaS funcionais na automatização dos testes UAT e será descrita melhor a seguir (SELENIUMHQ, 2017).

2.2.6.1 Ferramentas da Selenium

Automatizar testes funcionais é um dos problemas tradicionais dos projetos de desenvolvimento de software. A ferramenta de código aberto conhecida como Selenium, originalmente desenvolvida pela ThoughtWorks¹, possibilita solucionar os problemas de automatização dos testes para aplicações web (HOLMES; KELLOGG, 2006).

Atualmente, os processos de software são altamente interativos e responsivos e a automação de teste está cada vez mais frequente nos projetos de software. A automação de teste envolve, entre outros fatores, o processo de utilização de uma ferramenta de software para executar testes de forma repetitiva sobre a aplicação a ser testada (SELENIUMHQ, 2017).

Além disso, o Selenium é um conjunto de ferramentas de software, cada qual com abordagens diferentes para apoiar a automação de testes. A maioria dos engenheiros que utilizam o Selenium concentram-se na utilização de apenas uma ou duas ferramentas que mais atendem às necessidades dos seus projetos. No entanto todas as ferramentas fornecem diversas opções para abordar os diferentes problemas de automatização de teste, resultando em um conjunto de funcionalidades especificamente orientadas às necessidades de testes de aplicações web de todos os tipos. Sendo assim, estas operações são altamente flexíveis, permitindo várias opções para a localização de elementos de interface do usuário, ao mesmo tempo em que comparam os resultados dos testes previstos com o comportamento real da aplicação (SELENIUMHQ, 2017).

Dentre as duas ferramentas mais utilizadas, pode-se citar a Selenium Webdriver², que é voltada para automação de navegadores e a Selenium IDE³, que é um *plugin* para o navegador Mozilla Firefox, sendo que ambas as ferramentas possuem o núcleo Selenium. É importante ressaltar que neste projeto de doutorado foi utilizado o Selenium Webdriver, pois além de ser a solução de código aberto mais utilizada (SELENIUMHQ, 2017), atualmente é uma ferramenta completa e usada pela maioria dos fornecedores de Teste como Serviço para a automatização dos Testes de Aceitação do Usuário (SELENIUMHQ, 2017). As demais ferramentas do Selenium Grid são exemplificadas na Figura 7.

O Selenium permite fazer testes web com *scripts* que são executados a partir de um navegador por meio de JavaScript e *iframe* para incorporar o mecanismo de automação de teste ao *browser*. Isso permite que os mesmos *scripts* de teste possam ser usados em diversos navegadores e em várias plataformas. Uma das principais características do Selenium é que a ferramenta oferece ao usuário um conjunto padrão de comandos de verificação para permitir que o usuário especifique os valores ou o comportamento esperado (HOLMES; KELLOGG, 2006).

Como o Selenium é baseado em JavaScript e executado a partir do navegador, o usuário pode acompanhar a execução dos testes e conferir os resultados esperados. Por exemplo, se o teste foi executado corretamente com sucesso é sinalizado por uma indicação verde, caso contrário, se

¹ ThoughtWorks: <http://www.thoughtworks.com/pt/>

² Selenium Webdriver: <http://www.seleniumhq.org/projects/webdriver/>

³ Selenium IDE: <http://www.seleniumhq.org/projects/ide/>

Figura 7 – Componentes do Selenium

Fonte: Adaptado de Goyal (2013)

o caso de teste falhar é sinalizado por uma indicação vermelha (HOLMES; KELLOGG, 2006).

O funcionamento da ferramenta Selenium está representado na Figura 8, na qual é possível perceber claramente que a partir de um mesmo teste automatizado é possível executá-lo em diferentes navegadores por meio do drive do navegador desejado.

Figura 8 – Funcionamento da ferramenta Selenium

Fonte: Krishnamoorthy (2012)

Portanto a ferramenta Selenium permite que os testadores possam submeter os seus *scripts* de testes para um navegador web executá-los de forma automatizada, sem a necessidade de intervenção humana. Neste sentido possibilita executar os testes ao mesmo tempo em que instrui os navegadores para testar as aplicações web. Porém, ao instalar o Selenium localmente podem ocorrer falhas de hardware, sendo necessários gastos com manutenção de infraestrutura local (KRISHNAMOORTHY, 2012).

2.3 Padrões de Projeto

Em engenharia de software, os padrões são descrições formais ou modelos de como resolver um problema específico (GAMMA *et al.*, 1995). Além disso, os padrões de software podem ser classificados ou categorizados em diferentes níveis de abstração, por exemplo, os padrões de projeto de desenvolvimento e teste de software (GAMMA *et al.*, 1995; YI-CHEN; YI-KUN, 2011).

Os padrões de projeto são soluções reutilizáveis em geral, para problemas específicos recorrentes encontrados no processo de desenvolvimento de software orientado a objetos. As principais vantagens dos padrões de projeto são: reúso de soluções de forma eficiente, baixo acoplamento e alta coesão, aumento da manutenibilidade do código fonte da aplicação e, por fim, contribuem também para melhorar a legibilidade e a qualidade do software no geral (GAMMA *et al.*, 1995).

Além dos padrões de projeto de desenvolvimento, existem também os padrões de projeto de teste de software que descrevem as soluções dos problemas que ocorrem repetidamente durante o teste de software, na criação de cenários de testes específicos, porém não estão relacionados aos testadores e nem ao sistema sob teste ou *System Under Test* (SUT). Esses padrões possibilitam melhorar a eficiência e a qualidade do software com base na experiência de processos de teste anteriores. Além disso, os padrões de projeto de teste permitem reduzir também as incertezas dos testes, devido à limitada capacidade humana em prever todos os cenários de teste possíveis e necessários (YI-CHEN; YI-KUN, 2011).

Em termos gerais, os padrões de projeto costumam ser divididos em três categorias: padrões de criação, padrões estruturais e padrões de comportamento. Padrões de criação descrevem mecanismos de criação de objetos. Padrões estruturais descrevem uma maneira simples de compreender as relações entre as entidades. Por fim, padrões comportamentais que descrevem o comportamento de como os objetos interagem por meio da troca de mensagens (GAMMA *et al.*, 1995).

Os padrões de projeto utilizados neste trabalho são uma combinação dos padrões de criação *Abstract factory* com o padrão comportamental *Strategy*, descritos por *Gang of Four* (GoF) (GAMMA *et al.*, 1995). O padrão de projeto *Strategy* foi utilizado para encapsular comportamentos, com o objetivo de torná-los intercambiáveis e selecioná-los em tempo de execução conforme as necessidades (GAMMA *et al.*, 1995). O padrão de projeto *Singleton* foi utilizado para garantir a existência de uma única instância de uma classe, centralizando o consumo de recursos computacionais (GAMMA *et al.*, 1995). Por último, o padrão de projeto *PageObject* foi utilizado para melhorar a legibilidade e a manutenibilidade do código de teste automatizado (SELENIUM, 2017).

O padrão de projeto *Abstract factory* é um padrão de projeto que permite a criação de diferentes tipos de objetos, sem a exigência de se especificar as classes concretas exatas que

serão utilizadas. Os objetos criados pela fábrica são determinados em tempo de execução de acordo com a seleção de uma classe. Sendo assim, usando interface genéricas, o cliente não sabe qual objeto concreto vai receber da fábrica. Conforme afirma Gamma *et al.* (1995), a utilização desse tipo de padrão é particularmente interessante quando o objetivo é separar os detalhes de instanciação dos objetos.

O padrão de projeto Singleton é um padrão de criação, utilizado quando é necessário garantir que a aplicação contenha apenas uma única instância de uma classe. Este padrão é muito utilizado para evitar o desperdício de recursos computacionais, centralizando o gerenciamento de recursos e o acesso de forma global do sistema (GAMMA *et al.*, 1995).

Um outro padrão de projeto utilizado neste trabalho foi PageObject (SELENIUM, 2017). Este padrão é utilizado na automatização de teste para evitar a duplicação de código. Ele organiza, simplifica e aumenta a reutilização do código de teste automatizado UAT. Este padrão estabelece também que toda a complexidade referente ao código do Selenium correspondente à localização de elementos em uma página web e ações, como por exemplo, clicar no botão submeter, devem ser encapsuladas dentro de uma classe que representa a página do sistema web, permitindo melhorar a legibilidade e aumentar a manutenibilidade do código de teste automatizado (SELENIUM, 2017; SAMS, 2015).

2.4 Fundamentos da Computação em Nuvem

A computação em nuvem, em meio à centralização de recursos e gerência da entrega e consumo de serviços de tecnologia, ressurgiu como um “novo” paradigma na indústria da Tecnologia da Informação, ou seja, as ideias de centralização ou distribuição de recursos computacionais não devem mais ser vistas de forma isolada, mas como uma combinação de recursos centralizados com a distribuição por meio de dispositivos portáteis e de baixo poder computacional (TAURION, 2009).

Essa transformação traz um grande impacto, não só para a indústria da informação como também para a sociedade em geral, fazendo com que os recursos sejam otimizados e, desta forma, acelerem o desenvolvimento científico e tragam benefícios capazes de aumentar o acesso às novas tecnologias (TAURION, 2009). Um exemplo disso é a redução de custos com máquinas extremamente simples e com baixo poder de processamento, que acessam um servidor central⁴. Isso representa uma economia de até 35% nos custos de hardware, além das consequentes economias com gastos de manutenção, pois os terminais não exigem atualizações individuais de software. Tudo isso sem incorrer nos riscos, por exemplo, de infecção por vírus (FUSCO, 2009).

O conceito de nuvem não surgiu como um novo modelo de tecnologia, mas como a integração de tecnologias anteriores (CHEN; LI; CHEN, 2011), o que resultou em uma nova maneira de usar e fornecer recursos computacionais como um serviço, por meio da internet. O

⁴ Chromebook: <http://www.google.com/intl/en/chrome/devices/chromebooks.html>

que torna essa ideia interessante é o fato de ela propor uma transformação da computação em um serviço sob demanda, como outro qualquer, como por exemplo: água e energia elétrica. As principais ideias por trás deste conceito são:

- (i) Permitir que as empresas possam adquirir recursos de computação sob demanda; e
- (ii) Permitir o pagamento de acordo com o volume de utilização, possibilitando que as empresas ignorem completamente o conhecimento sobre de onde esses recursos vêm (ARM-BRUST *et al.*, 2009).

As aplicações da nuvem apresentam, além de todas as vantagens de um sistema web tradicional, uma que se coloca entre suas principais características: a escalabilidade. Tal aspecto permite que as empresas possam contratar recursos computacionais sob demanda. Assim, é possível alocar e desalocar recursos de forma automática e conforme a necessidade de cada consumidor e da disponibilidade do provedor em ofertar mais recursos, tudo isso por meio da forma de pagamento por uso (RIMAL; CHOI; LUMB, 2009).

A escalabilidade é um requisito não funcional do sistema que, dependendo da necessidade do cliente, e a quantidade de acesso de usuários, pode resultar em uma aplicação escalável com alta ou baixa disponibilidade (BEZEMER; ZAIDMAN, 2010). Além disso, a escalabilidade de servidores pode ser horizontal ou vertical.

A escalabilidade horizontal (*scale out*) permite adicionar novos nós ou servidores a uma arquitetura de sistema. Para isso, é necessário que o sistema de software gerencie e suporte a distribuição do trabalho entre múltiplas máquinas. Portanto escalar horizontalmente é acoplar várias máquinas individuais (virtuais ou físicas) com o objetivo principal de compartilhar o mesmo recurso, servindo a mesma aplicação (GOGRID, 2010). A Figura 9 (a) corresponde a um exemplo de escalabilidade horizontal.

Já o termo escalabilidade vertical (*scale up*) permite aumentar o processamento do hardware de um único servidor, adicionando-se novos recursos computacionais, como memória, processador ou até mesmo um disco rígido mais eficiente para o armazenamento de uma determinada aplicação e, principalmente, para atender a uma demanda de um número crescente de requisições. A Figura 9 (b) corresponde a um exemplo de escalabilidade vertical.

É importante destacar a diferença entre os termos escalabilidade e elasticidade, pois nem sempre o que é escalável é elástico. A escalabilidade está relacionada em atender uma grande quantidade de demanda, como por exemplo adicionar ou remover infraestrutura de servidores. Porém a escalabilidade é um conceito amplo, podendo inclusive ser gerenciada por meio de um software e não estabelece a forma como deve ocorrer a alocação e desalocação de recursos. Uma das possíveis formas de escalar recursos computacionais é por meio da elasticidade. A elasticidade está diretamente relacionada com a otimização de recursos, ou seja, o processo de alocar e desalocar recursos conforme a demanda e de forma automática, simples

Figura 9 – Exemplos de Escalabilidade na Computação em Nuvem

Fonte: GoGrid (2010)

e rápida. Todas essas características determinam se o sistema é elástico ou não. A elasticidade é uma das principais características da computação em nuvem, porém ser elástica não implica necessariamente em reduzir custos e nem sempre é a melhor solução para todos os problemas (HERBST; KOUNEV; REUSSNER, 2013).

Os ambientes de computação em nuvem são diferentes de um ambiente tradicional em termos de implantação, configuração, execução e gerenciamento das aplicações. A principal diferença entre os aplicativos tradicionais e aplicativos baseados em nuvem está na comparação entre os tipos de usuários, arquitetura *multi-tenant*, segurança, etc (CHANA; CHAWLA, 2013).

Assim como as arquiteturas *Service-Oriented Architecture* (SOA) e cliente-servidor, a arquitetura *multi-tenant* ou multi-inquilino está diretamente relacionada ao conceito de virtualização, que suporta o compartilhamento de computação, armazenamento e recursos de rede entre vários clientes. Em uma nuvem, um cliente (inquilino) poderia ser um usuário, um grupo de usuários, ou até mesmo uma organização/empresa (CHANA; CHAWLA, 2013). Portanto os termos cliente, inquilino ou *tenant* são sinônimos e podem ser definidos também como a entidade organizacional que aluga uma solução SaaS *multi-tenant*. Normalmente, um inquilino é considerado como um grupo de vários usuários, que são as partes interessadas (*stakeholders*) na organização (BEZEMER; ZAIDMAN, 2010).

A arquitetura *multi-tenant* permite que os clientes (*tenants*) compartilhem os mesmos recursos de hardware, oferecendo-lhes um aplicativo e instância de banco de dados de forma compartilhada, ao mesmo tempo que lhes permite configurar o aplicativo para atender suas necessidades, como se fosse executado em um ambiente dedicado (BEZEMER; ZAIDMAN, 2010).

A computação em nuvem é uma das mais promissoras evoluções tecnológicas em desenvolvimento. Na literatura foram encontrados diversos trabalhos relacionados ao problema da falta de portabilidade e interoperabilidade em aplicações da computação em nuvem. O tema tem sido bastante abordado pela comunidade acadêmica (ARMBRUST *et al.*, 2009), o que demonstra sua relevância.

Um dos principais problemas relacionados não só ao desenvolvimento, mas também ao teste de software na computação em nuvem é denominado *vendor lock-in* (ARMBRUST *et al.*, 2009; ARMBRUST *et al.*, 2010). O *vendor lock-in* é ocasionado devido à falta de portabilidade e interoperabilidade entre os provedores (ARMBRUST *et al.*, 2009; ARMBRUST *et al.*, 2010), sendo amplamente reconhecido como um dos maiores obstáculos para a adoção da nuvem (DILLON; WU; CHANG, 2010). Usuários da nuvem são muitas vezes presos a um provedor de nuvem específico, devido às diferenças significativas nas tecnologias (ROCHWERGER *et al.*, 2009) e interfaces (RODERO-MERINO *et al.*, 2010) adotadas pelos provedores de serviço na nuvem. Como consequência, as soluções são proprietárias e dificultam a portabilidade e a interoperabilidade na nuvem (HOFMANN; WOODS, 2010).

Embora a definição dos conceitos de portabilidade e interoperabilidade possam variar na literatura e até causar confusão (GOVINDARAJAN; LAKSHMANAN, 2010), (PETCU, 2011), (SNELL; GLOVER, 2003) considera-se a portabilidade como a capacidade de transportar uma aplicação que executa, por exemplo no Glassfish⁵ do fornecedor Oracle⁶ e ao implantá-la em outro *Java EE container*, por exemplo, o *JBoss Application Server*⁷ ou *Weblogic*⁸ dos fornecedores JBoss⁹ e Oracle respectivamente, a aplicação executará com sucesso, pois ambos implementam a mesma especificação *Java 2 Platform, Enterprise Edition* (J2EE) (SNELL; GLOVER, 2003; SURHONE; TENNOE; HENSSONOW, 2011). A portabilidade pode ser resumida no *slogan* da empresa Sun Microsystems para a tecnologia Java: “*Escreva uma vez, execute em qualquer lugar*”. Por sua vez, a interoperabilidade é a capacidade da implementação de um determinado fornecedor comunicar-se diretamente com outros fornecedores. Portanto a interoperabilidade pode ser alcançada por meio de: **a)** adoção de protocolos e padrões abertos, como *Hypertext Transfer Protocol* (HTTP), *Transmission Control Protocol* (TCP)/*Internet Protocol* (IP) e *HyperText Markup Language* (HTML) da *World Wide Web* (WWW) e **b)** o uso de um *broker* ou camada intermediária para integração de aplicações de sistemas de tecnologias heterogêneas, como *Simple Object Access Protocol* (SOAP) e *Web Services Description Language* (WSDL) ou SOAP-WSDL, *Common Object Request Broker Architecture* (CORBA), *Distributed Component Object Model* (DCOM) e *Java Remote Method Invocation* (RMI) (TECHTARGET, 2006).

De acordo com Snell e Glover (2003), uma aplicação portátil não é necessariamente interoperável, e vice-versa. A diferença sutil entre esses dois atributos de qualidade de software reside no aspecto de comunicação considerado apenas pela interoperabilidade (SNELL; GLOVER, 2003).

No escopo deste trabalho é adotada a definição dos autores Snell e Glover (2003), na qual a portabilidade é um conceito vertical, ou seja, relativo à abstração das interfaces que um

⁵ *Glassfish*: <<https://glassfish.java.net/>>

⁶ *Oracle*: <<http://www.oracle.com/index.html>>

⁷ *JBoss Application Server*: <<http://jbossas.jboss.org/downloads/>>

⁸ *Weblogic*: <<http://www.oracle.com/technetwork/middleware/weblogic/overview/index.html>>

⁹ *JBoss*: <<http://www.jboss.org/>>

aplicativo utiliza para se comunicar com outros sistemas da mesma plataforma sobre a qual ele está sendo executado, conforme ilustrado na Figura 10. A interoperabilidade é um conceito horizontal, quando um determinado sistema consegue se comunicar com um ou mais sistemas de outras plataformas, conforme ilustrado na Figura 11.

Figura 10 – Modelo de componente portátil vertical

Fonte: Snell e Glover (2003)

Figura 11 – Modelo de serviço interoperável horizontal

Fonte: Snell e Glover (2003)

Sem essas abstrações horizontais, aplicações executadas em múltiplas plataformas não seriam capazes de se comunicar umas com as outras, devido às diferentes tecnologias existentes entre elas. Portanto assim como tecnologia Java e J2EE permitem mover uma aplicação de negócio a partir de uma tecnologia de plataforma para a outra, os *web services* permitem integrar aplicativos de negócios que funcionem em diferentes plataformas.

2.5 Teste de Software na Computação em Nuvem

O teste de software é uma área que possibilita a migração para o ambiente de computação em nuvem. A virtualização, que é uma tecnologia capacitadora da computação em nuvem é usada para fornecer recursos computacionais virtuais para testar aplicações web em diferentes plataformas (INÇKI; ARI; SOZER., 2012).

Testar um novo software muitas vezes requer dispositivos de alto custo, tais como: servidores, armazenamento e rede, apenas por um tempo limitado (KIM *et al.*, 2009). Estes recursos de computação são subutilizados após o teste, incorrendo, assim, em um custo extra no orçamento. Esse é o cenário no qual o teste na nuvem surgiu como uma nova abordagem para o teste de software, na qual os ambientes de computação em nuvem são aproveitados para simular o tráfego de usuários no mundo real, diminuindo significativamente os custos (PRAKASH, 2010). Essa característica também pode ser estendida para os clássicos testes funcional, estrutural,

regressão e entre outros testes de produtos regulares de um ciclo de desenvolvimento de software, a partir de uma perspectiva de custos (KATHERINE; ALAGARSAMY, 2012).

Em essência, o teste na nuvem é uma forma de teste de software, no qual o teste é executado utilizando os recursos, máquinas ou servidores da infraestrutura de nuvem para apoiar os testes funcionais multiplataformas de aplicações web. Com esse tipo de teste é possível combinar diferentes dispositivos, linguagens de programação, sistemas operacionais e até mesmo diferentes navegadores (*cross browser*), sendo possível também prever restrições do tipo, como por exemplo, o navegador internet explorer não executa no sistema operacional linux. Além dos testes funcionais, o teste na nuvem permite realizar também os testes não funcionais por meio de simulação do tráfego de usuários no mundo real, para auxiliar os testes de carga, estresse e desempenho de aplicações web.

Comparado com o teste de software convencional, o teste na nuvem possui diversas vantagens, conforme mencionado por (GAISBAUER *et al.*, 2008), (CIORTEA *et al.*, 2010) e (AKERELE; RAMACHANDRAN; DIXON, 2013), com destaque para:

- Aproveitamento da infraestrutura de computação em nuvem, o que reduz o custo unitário de computação. Refere-se ao uso de forma eficaz, dos recursos virtualizados e da infraestrutura de nuvem compartilhada. Serão suprimidos recursos computacionais desnecessários e os custos de software licenciados em um laboratório de teste. Consequentemente, a eficiência do teste, ou seja, a velocidade referente ao tempo de execução será aumentada;
- Oferecimento por parte dos prestadores de serviços de testes baseados em nuvem de uma infraestrutura padronizada e imagens de software pré-configuradas, capazes de reduzir consideravelmente os erros;
- Oferecimento de serviços de teste sob demanda (conceito de terceirização) para realizar testes em grande escala e validação *on-line* em tempo real e efetiva para os sistemas web na nuvem;
- Execução e avaliação de testes (SaaS/Nuvem/Aplicação) sobre a infraestrutura do sistema de nuvem de forma escalável.

2.5.1 Teste como Serviço (TaaS)

O Teste como Serviço (TaaS) pode ser definido como uma ferramenta de teste oferecida como um serviço para verificar e validar os requisitos funcionais e não funcionais de um sistema web e *mobile*, no ambiente de computação em nuvem. Um serviço de teste funcional pode ser usado para testar não só a interface mas também as funcionalidades de um sistema por meio da internet. Esse tipo de ferramenta utiliza o Selenium Grid¹⁰ como principal *engine* (motor) e

¹⁰ Selenium Grid: <<http://www.seleniumhq.org/projects/grid/>>

representa o principal foco desta pesquisa. Exemplos de fornecedores desse tipo de serviço são o Saucelabs¹¹, Browserstack¹², Testingbot¹³ e Experitest¹⁴ (*Mobile*) (GAO; BAI; TSAI, 2011; YU *et al.*, 2010; GAO *et al.*, 2014).

Em contrapartida, os serviços de teste não funcionais utilizam o Jmeter¹⁵ para simular o tráfego intenso de usuários de diferentes localizações geográficas. Além disso eles possibilitam gerar uma grande quantidade de carga por meio de requisições HTTP que serão utilizadas nos testes de desempenho, carga ou stress. Pode-se citar, os serviços oferecidos pela Loadmanager¹⁶, Blazemeter^{17 18} e Loadimpact¹⁹ (GAO; BAI; TSAI, 2011; YU *et al.*, 2010; GAO *et al.*, 2014).

O Teste como Serviço está revolucionando a área de teste de software ao inovar a forma de gerenciar e, principalmente, entregar serviços de teste por meio da internet. As principais vantagens do modelo TaaS são a redução de custos, pagamento por uso ou conforme a demanda de teste e eliminação de despesas de capitais iniciais com infraestrutura, licenças de ferramentas, configuração e manutenção de ambientes de teste (SHAWISH; SALAMA, 2014).

Nesse contexto, o teste na nuvem proporciona uma maneira eficiente e eficaz de garantir a qualidade do teste de software de forma escalável, permitindo que as organizações reduzam os custos com os gastos referentes à infraestrutura de teste.

As principais entidades básicas envolvidas no TaaS são:

- **Consumidor TaaS:** Cada TaaS é um subconjunto de SaaS usado por um engenheiro de software — desenvolvedor ou testador — como uma ferramenta de apoio para suas atividades, e
- **Fornecedor TaaS:** Um provedor TaaS é uma empresa que fornece o serviço de teste para os consumidores de TaaS por meio da Internet.

2.5.1.1 Consumidores de Teste como Serviço

Todo TaaS é um Software as a Service (SaaS) consumido por um engenheiro de software que pode ser tanto um desenvolvedor quanto um testador, servindo como uma ferramenta de apoio para as suas atividades. Na Figura 12 ilustram-se as relações entre as diferentes camadas de abstração na computação em nuvem. No entanto, cabe ressaltar, que nem todo SaaS é um

¹¹ Saucelabs: <<https://Saucelabs.com/>>

¹² Browserstack: <<https://www.browserstack.com/>>

¹³ Testingbot: <<https://Testingbot.com/>>

¹⁴ Experitest: <<https://experitest.com/>>

¹⁵ Apache JMeter: <<http://jmeter.apache.org/>>

¹⁶ Loadmanager: <<http://loadmanager.com.br/>>

¹⁷ Blazemeter: <<https://www.blazemeter.com/>>

¹⁸ Blazemeter mobile: <<https://guide.blazemeter.com/hc/en-us/articles/207420335-Testing-Mobile-sites-and-Apps>>

¹⁹ Loadimpact: <<https://loadimpact.com/>>

TaaS. A aplicação do Gmail da empresa Google é um SaaS que é oferecido como um serviço para o usuário final.

Figura 12 – Exemplo de interação entre atores e modelos de serviços na computação em nuvem

Fonte: Adaptado de Marinos e Briscoe (2009)

2.5.1.2 Fornecedores de Teste como Serviço

Conforme explicado anteriormente, o Selenium é um conjunto de ferramentas utilizado pela maioria dos fornecedores TaaS funcionais na automatização dos Testes de Aceitação do Usuário (UAT), permitindo localizar elementos de interface do usuário e possibilitando executar os testes ao mesmo tempo em que instrui os navegadores na execução dos *scripts* de testes de aplicações web, sem a necessidade de intervenção humana (SELENIUMHQ, 2017).

Saucelabs²⁰, Browserstack²¹ e TestingBot²² são exemplos de plataformas de TaaS que oferecem o serviço de testes automatizados de múltiplos navegadores (*cross-browser*). Permitem executar os testes no ambiente de computação em nuvem, além de oferecer um ambiente de teste totalmente configurado em seus servidores. Cabe ressaltar que os fornecedores de TaaS possuem um conjunto de ferramentas de desenvolvimento pré-instalado para auxiliar na depuração de problemas em tempo real, além de permitir navegar através da aplicação web para verificar questões não funcionais e funcionais, como por exemplo: capturar *screenshots* das páginas com erros, vídeos da execução dos testes e até mesmo um sistema de *log* semelhante ao da ferramenta Selenium (SAUCELABS, 2014a; BROWSERSTACK, 2014; TESTINGBOT, 2015).

Os fornecedores de TaaS Saucelabs, Browserstack e Testingbot foram escolhidos para essa pesquisa, justamente por serem considerados os maiores fornecedores de Teste como Serviço

²⁰ Saucelabs: <<https://saucelabs.com/>>

²¹ Browserstack: <<https://www.browserstack.com/>>

²² TestingBot: <<https://testingbot.com/>>

do mercado atual. Além disso, as três ferramentas de teste são utilizadas como referência no próprio site do Selenium²³.

Uma característica importante das plataformas de TaaS é a possibilidade de fornecerem o serviço de teste em tempo real para testar qualquer tipo de aplicações web, além de possuir um conjunto de máquinas virtuais pré-configuradas para cada plataforma e combinação de *browsers*, desde dispositivos móveis aos computadores *desktop* por meio da ferramenta Selenium (DEGRAVE, 2013).

Uma das principais vantagens das plataformas de TaaS está na paralelização da execução dos testes. Além disso, os três fornecedores dessa pesquisa utilizam a ferramenta Selenium para a automatização dos testes UAT multiplataforma. Entre os principais problemas a serem apontados na sua utilização, destaca-se o fato de possuir API's proprietárias, como por exemplo as suas APIs REST para testes automatizados: Saucelabs²⁴, Browserstack²⁵ e Testingbot²⁶, que ocasionam o problema de *lock-in* para os consumidores dos serviços de teste.

Por meio dos exemplos de código do Saucelabs, Browserstack e Testingbot, retirados de Sams (2015) e conforme os códigos 2, 3 e 4, respectivamente, do Apêndice D.1, serão apresentados nas próximas subseções, no qual é possível perceber claramente que o problema de *lock-in* está na escrita das capacidades e não no código referente ao Selenium, que está diretamente relacionado com a execução dos testes. Nesse contexto, é importante destacar que cada conjunto de teste pode possuir diversas combinações e restrições de capacidades entre diferentes sistemas operacionais, linguagens de programação, dispositivos e até mesmo diferentes navegadores com suas respectivas versões.

2.5.1.2.1 Fornecedor de TaaS - Saucelabs

O Saucelabs é uma ferramenta de automatização de múltiplos navegadores que permite executar os testes no ambiente de computação em nuvem, além de oferecer um ambiente de teste totalmente configurado em seus servidores. Possui também um conjunto de ferramentas de desenvolvimento pré-instalado para auxiliar na depuração de problemas em tempo real (SAUCELABS, 2014a).

O único pré-requisito necessário para os usuários executarem os testes é ter conexão com a internet. Sendo assim, basta aos usuários selecionar as configurações de requisitos de teste desejadas através da interface do navegador ou por meio das API's fornecidas pelos prestadores de serviço (AKERELE; RAMACHANDRAN; DIXON, 2013). Dessa forma o consumidor se preocupará apenas com as principais questões de projeto (KRISHNAMOORTHY, 2012).

²³ Selenium: <<http://www.seleniumhq.org/ecosystem/>>

²⁴ API REST Saucelabs: <<https://docs.Saucelabs.com/reference/rest-api/>>

²⁵ API REST Browserstack: <<https://www.browserstack.com/automate/rest-api/>>

²⁶ API REST Testingbot: <<https://Testingbot.com/support/api>>

É importante destacar ainda que uma das suas principais vantagens está na paralelização da execução dos testes. Além disso, o Sauce Labs utiliza a ferramenta Selenium para a automatização dos testes UAT multiplataforma.

O Código fonte 2 do Apêndice D apresenta um exemplo de utilização das capacidades do Sauce Labs.

2.5.1.2.2 Fornecedor de TaaS - Browserstack

O Browserstack é uma plataforma de teste de múltiplos navegadores, semelhante ao Sauce Labs, que também é executada a partir de um ambiente de nuvem (BROWSERSTACK, 2014). A plataforma do Browserstack oferece também API's proprietárias, como a sua API REST para testes automatizados²⁷, que também ocasionam o problema de *vendor lock-in* para os consumidores dos serviços de teste.

É importante enfatizar que a plataforma Browserstack fornece serviço de teste em tempo real para testar qualquer tipo de aplicações web. Possui um conjunto de máquinas virtuais pré configuradas para cada plataforma e combinação de *browsers* desde dispositivos móveis aos computadores *desktop* por meio da ferramenta Selenium (DEGRAVE, 2013).

O Código fonte 3 do Apêndice D apresenta um exemplo de utilização das capacidades do Browserstack.

2.5.1.2.3 Fornecedor de TaaS - Testingbot

O Testingbot é uma plataforma de teste de múltiplos navegadores por meio do Selenium e executada a partir de um ambiente de nuvem (TESTINGBOT, 2015), semelhante ao Browserstack e ao Sauce Labs. Além disso, o Testingbot utiliza toda a infraestrutura de nuvem da Amazon em *background* para executar *scripts* de Selenium em vários *browsers* (SAMS, 2015).

O Código fonte 4 do Apêndice D apresenta um exemplo de utilização das capacidades do Testingbot.

2.5.1.3 **Funcionamento do Teste como Serviço (TaaS)**

O teste na nuvem envolve as etapas de configuração das capacidades, escrita e execução de testes automatizados e gerenciamento dos resultados de testes. A redução nos custos por meio do TaaS ocorre principalmente devido ao modelo de pagamento por uso. No entanto a falta de padronização nas atividades de teste causa o aprisionamento do consumidor a um fornecedor específico. Essa dependência ocasionada pelo *vendor lock-in* dificulta a troca de provedores devido aos altos custos de manutenção para a migração dos artefatos envolvidos em cada etapa do processo de teste na nuvem (GAO; BAI; TSAI, 2011; GAO *et al.*, 2013).

²⁷ API REST Browserstack: <<https://www.browserstack.com/automate/rest-api/>>

A Figura 13 mostra uma visão geral do uso de TaaS. Três entidades, nomeadamente o Cliente TaaS, o Fornecedor TaaS e a Aplicação Web (SUT) estão envolvidos. Os *scripts* de teste são escritos localmente por um cliente TaaS. Um *socket* é aberto através do protocolo TCP e os comandos do Selenium são enviados ao provedor (Saucelabs, por exemplo) (SAUCELABS, 2014a). O provedor executa os testes em uma infraestrutura de nuvem sobre a aplicação Web e retorna os resultados ao consumidor por meio da API REST proprietária, a partir da qual as informações, como a versão do navegador, sistema operacional, *logs*, *screenshots* e vídeos dos testes podem ser obtidos. Toda a interação e comunicação entre o cliente e o fornecedor ocorre através do Selenium Remote WebDriver e API REST (SELENIUMHQ, 2017).

Figura 13 – Visão geral do uso de TaaS por meio de testes automatizados

Fonte: Autoria própria

Entre as principais causas do *vendor lock-in* no TaaS, estão a falta de padronização de capacidades necessárias para a execução do teste na nuvem e também das APIs REST^{28 29 30} proprietárias dos fornecedores de TaaS, que prejudicam a portabilidade e a interoperabilidade das aplicações que gerenciam os resultados dos testes automatizados e consomem o serviço de teste na nuvem. Além disso, a forte dependência do consumidor a um determinado fornecedor ou serviço de teste impede que o TaaS seja amplamente adotado.

A portabilidade e interoperabilidade no contexto de Teste como Serviço são dois conceitos importantes para contornar o problema de *vendor lock-in* (LOUTAS *et al.*, 2011; ZHANG; WU; CHEUNG, 2013), no qual a portabilidade do TaaS é o conceito que se refere à habilidade do engenheiro de software escrever o teste automatizado uma única vez e executá-lo em

²⁸ API REST Saucelabs: <<https://docs.Saucelabs.com/reference/rest-api/>>

²⁹ API REST Browserstack: <<https://www.browserstack.com/automate/rest-api/>>

³⁰ API REST Testingbot: <<https://Testingbot.com/support/api>>

múltiplas plataformas TaaS, sem a necessidade de modificar o código de teste. A portabilidade refere-se também a portar o software que sumariza os resultados dos testes sem a necessidade de modificar o código fonte da aplicação. Já interoperabilidade, no contexto de TaaS, refere-se à ter um mecanismo (protocolo padrão ou camada de abstração) que permita que os fornecedores TaaS heterogêneos colaborem de forma conjunta e se comuniquem uns com os outros de forma transparente para os consumidores. Isso só seria possível se as APIs REST dos fornecedores TaaS seguissem uma especificação em comum.

2.5.2 Mapeamento das capacidades dos Fornecedores de TaaS

Atualmente, o número empresas e a demanda pelo serviço de teste na nuvem vem crescendo (MARKETING, 2014). Além disso, o número de fornecedores TaaS no mercado vem aumentando consideravelmente, conforme sugere a listagem (ALTERNATIVETO, 2017) e, conseqüentemente, houve também um aumento do número de capacidades específicas oferecidas por diferentes fornecedores de TaaS. As capacidades são formadas por um par de chave e valor que fazem parte da escrita do teste automatizado e são responsáveis pela execução dos testes automatizados na nuvem.

Sabendo que cada conjunto de testes pode possuir diferentes capacidades com diversas combinações de sistemas operacionais, dispositivos e navegadores com as suas respectivas versões, uma empresa que possui entre centenas ou milhares de testes, terá um custo exorbitante com a manutenção do código ao realizar a troca de fornecedores, devido a escrita das capacidades, execução, configuração e gerenciamento dos resultados de teste.

Além disso, ao criar um novo conjunto de casos de teste, seja por meio de um *bug report* do usuário final ou pela própria manutenção evolutiva do sistema e considerando o número de projetos envolvidos, torna-se inviável alterar manualmente todo o código das capacidades para cada nova combinação de configurações de sistema operacional, navegador, dispositivos, resolução de tela entre outras.

A Tabela 1 apresenta uma lista completa comparando as capacidades dos fornecedores Sauce Labs, Browserstack e Testingbot. Todas as informações foram extraídas a partir dos sites dos respectivos fornecedores de TaaS, onde é possível perceber claramente que existem apenas onze capacidades em comum entre os fornecedores, como por exemplo: nome do navegador, versão do navegador, capturas de *screenshots* e vídeo, conforme ilustrado na Tabela 2.

Nesse contexto, tornando a situação ainda mais complexa, não existe nenhuma padronização da nomenclatura das capacidades, o que dificulta a manutenção adaptativa do código, referente a capacidade do sistema de se adaptar a um novo ambiente (SOMMERVILLE, 2007), caso o cliente queira mudar de fornecedor futuramente. Assim, é necessário ainda utilizar um

³¹ *Saucelabs*: <<https://wiki.saucelabs.com/display/DOCS/Test+Configuration+Options/>>

³² *Browserstack*: <<https://www.browserstack.com/automate/capabilities/>>

³³ *Testingbot*: <<https://testingbot.com/support/other/test-options/>>

Tabela 1 – Capacidades dos fornecedores de TaaS

Nº	Capacidades / TaaS	Saucelabs ³¹	Browserstack ³²	TestingBot ³³
1	Nome do Navegador	✓	✓	✓
2	Versão do Navegador	✓	✓	✓
3	Nome do Sistema Operacional	✓	✓	✓
4	Versão do Sistema Operacional	✓	✓	
5	Nome do dispositivo	✓	✓	
6	Raiz da aplicação	✓		
7	Versão do Selenium	✓	✓	✓
8	Orientação do dispositivo	✓	✓	✓
9	Motor de automação	✓		
10	Desativar flash no Internet Explorer (IE)		✓	
11	Alertas	✓		✓
12	<i>AutoClicker</i>			✓
13	Pacote de Aplicação	✓		
14	Atividade do Android	✓		
15	Nome do teste	✓	✓	✓
16	Nome do Projeto		✓	
17	Número da <i>build</i>	✓	✓	✓
18	Teste interno/servidores locais		✓	
19	Marcação/ <i>Tagging</i>	✓		
20	<i>Status</i> Passou/Falhou	✓		
21	Dados personalizados	✓		✓
22	Versão do <i>driver</i> do Chrome	✓		✓
23	Duração máxima do teste	✓		✓
24	<i>Command Timeout</i>	✓		
25	<i>Idle Test Timeout</i>	✓		✓
26	Privacidade do teste			✓
27	Extensão do usuário			✓
28	Extensão do Chrome/Firefox			✓
29	Agrupamento			✓
30	<i>Pre-run Executables</i>	✓		✓
31	<i>Tunnels</i> identificados	✓		
32	Resolução de tela	✓	✓	✓
33	<i>Time Zones</i> personalizados	✓		
34	Versão do driver do IE	✓	✓	✓
35	<i>Upload</i> de arquivos para VM			✓
36	Evitar o proxy do Selenium	✓		✓
37	Visibilidade do <i>Job</i>	✓		
38	Captura de vídeo	✓	✓	✓
39	Captura de <i>screenshots</i>	✓	✓	✓
40	Desativar gravação de <i>log</i>	✓		
41	Lista negra de <i>hostnames</i>			✓
42	Ativar <i>popups</i> no IE		✓	
43	Ativar <i>popups</i> no Safari		✓	
44	Ativar captura de fonte HTML	✓		
45	Ativar <i>captura automática de tela do WebDriver</i>	✓		
Total de Capacidades		33	18	25

processo de reengenharia, ou seja, modificar um sistema sem alterar as suas funcionalidades (SOMMERVILLE, 2007), o que envolve inclusive todo o processo de reescrita e migração do código fonte para as diferentes combinações de capacidades.

Por fim, para agravar ainda mais o problema da falta de padronização entre os fornecedores de TaaS, os formatos de dados para consumir os serviços de teste podem ser diferentes, como por exemplo o Saucelabs que utiliza os serviços RESTful³⁴ sobre o HTTPS e codificação de dados de requisição e resposta no formato JSON ao passo que o Browsershots utiliza chamadas

³⁴ *Saucelabs REST API*: <<https://wiki.Saucelabs.com/display/DOCS/The+Sauce+Labs+REST+API>>

Tabela 2 – Capacidades em comum dos fornecedores de TaaS

Nº	Capacidades
1	Nome do Navegador
2	Versão do Navegador
3	Número da <i>Build</i>
4	Captura de <i>screenshots</i>
5	Captura de vídeo
6	Orientação do dispositivo
7	Versão do driver do Internet Explorer
8	Nome do Sistema Operacional
9	Versão do Selenium
10	Resolução de tela
11	Nome do teste

em *Remote Procedure Call* (RPC)³⁵ e formato de dados XML.

2.5.3 Mapeamento das API REST dos Fornecedores de TaaS

A análise da API REST dos fornecedores de TaaS descritas nesta seção foi realizada com base na documentação disponibilizada pelos três fornecedores de TaaS. A análise comparativa entre as APIs REST foi estabelecida a partir de programas intermediários que consomem os serviços de teste, utilizados como objetos de estudo, para cada uma das diferentes APIs REST com as mesmas informações do relatório final dos resultados.

Durante o desenvolvimento deste trabalho, observou-se que uma das principais desvantagens do fornecedor Browserstack é não ser possível obter os *screenshots* por meio da sua API REST, sendo necessário recuperar, analisar e extrair por meio de expressões regulares as URL's das imagens dos *screenshots*, contidas nos arquivos de *logs* no formato de texto de cada uma das *builds*. É evidente que essa abordagem impacta negativamente o desempenho das aplicações que consomem o serviço de teste, devido ao considerável tempo de processamento da aplicação intermediária e o fornecedor de TaaS.

Em comparação com o Browserstack, o fornecedor Sauce Labs disponibiliza as URL's das imagens do *screenshots* por meio da sua API, o que facilita o trabalho dos engenheiros de software no gerenciamento dos resultados da execução dos testes. Um outro detalhe importante do Sauce Labs é que o campo de tempo “*Duration*” do relatório foi calculado a partir das devidas conversões dos campos “*start time*” e “*end time*” para o *time zone* adequado.

Durante o desenvolvimento do projeto, o fornecedor Testingbot não fornecia a versão do *browser* de forma simples, sendo necessário criar métodos adicionais, como por exemplo, dividir o valor do campo “*browser*” em duas strings, separando as informações referentes ao navegador utilizado no teste (por exemplo, Internet Explorer, Firefox, Google Chrome) das informações da versão do navegador (por exemplo, 42.0, 47.0, ...). Além disso, foi necessário adicionar um outro método para extrair as informações do nome da *build* a partir do arquivo de *log* no formato “.txt”, também por meio do uso de expressões regulares. Porém, quando o pesquisador contactou o

³⁵ *Browsershots XML-RPC*: <<http://api.browsershots.org/xmlrpc/>>

fornecedor Testingbot com as principais dúvidas da sua API REST, o próprio fornecedor evoluiu a sua API REST, passando a facilitar a forma de obtenção do nome da *build* de teste, além de disponibilizar os valores do *browser* e versão de forma independente.

Apesar dos TaaS's oferecerem o mesmo tipo de serviço como, por exemplo, *screenshot*, a maneira como eles oferecem esses serviços são diferentes, sendo necessário gerar programas intermediários diferentes para sumarizar os resultados de cada TaaS, ocasionando o problema de *vendor lock-in* no gerenciamento dos resultados da execução dos testes, devido as diferentes informações obtidas entre os fornecedores de TaaS.

Portanto com base nas análises descritas anteriormente e de acordo com a Tabela 3, pode-se afirmar que as API's proprietárias dos fornecedores de TaaS são totalmente diferentes e que o Saucelabs possui um número maior de funcionalidades que os demais fornecedores. Diferentemente das 11 capacidades em comum descritas anteriormente, as operações em comum das API REST dos fornecedores Saucelabs, Browserstack e Testingbot são apenas 4: operações básicas de obter *status*, atualizar, listar e remover testes. Este cenário ilustra perfeitamente o alto impacto do *vendor lock-in* de maior grau na API REST de cada fornecedor de TaaS.

2.6 Exemplo de Abordagem Convencional de Migração entre Fornecedores de TaaS

Nesta seção será apresentado um exemplo de implementação da abordagem convencional de migração entre fornecedores de TaaS. Conforme o próprio nome sugere, a abordagem convencional para automatização de testes na computação em nuvem foi desenvolvida conforme as especificações e convenções sugeridas nas documentações dos fornecedores de TaaS Saucelabs (GOCHENOUR, 2016), Browserstack (BROWSERSTACK, 2017) e Testingbot (TESTINGBOT, 2017).

A abordagem convencional representa uma abordagem de migração entre fornecedores de TaaS que ocorre de forma programática (*hard coded*), sem o devido planejamento ou preocupação em evitar o problema e os riscos associados ao *vendor lock-in* do TaaS, contribuindo consequentemente para aumentar drasticamente os custos de manutenção da migração.

A abordagem convencional foi desenvolvida na linguagem de programação Java e de forma independente para cada um dos três fornecedores de TaaS sem uma camada de abstração. Todo o processo de teste na nuvem é realizado de forma convencional. Uma outra característica importante desta abordagem é que ela dificulta muito a troca de fornecedores de TaaS, encarecendo todo o processo de manutenção e migração tanto dos testes automatizados UAT quanto do mecanismo de sumarização dos resultados, pois exige que o engenheiro de software realize todo o processo de reengenharia e reescrita dos testes automatizados UAT.

Na abordagem convencional foram utilizados padrões de projeto na automatização dos

Tabela 3 – Mapeamento das APIs REST dos fornecedores de TaaS

ID	Operação da API/REST	Saucelabs	Browserstack	Testingbot
1	Autenticação de usuários (<i>Request URL e Authorization Header</i>)	✓		
2	Criar um usuário	✓		
3	Obter usuários simultâneos	✓		
4	Listar subcontas	✓		
5	Obter lista de subcontas irmãs	✓		
6	Obter informações de subconta	✓		
7	Listar informações de um usuário	✓		✓
8	Editar informações do usuário			✓
9	Obter histórico de acesso dos usuários	✓		
10	Modificar chave de acesso	✓	✓	
11	Limites de taxa de requisições por segundo	✓		
12	Obter <i>status</i> dos testes (por exemplo, <i>Running</i> e <i>Finished</i>)	✓	✓	✓
13	Obter <i>status</i> do serviço de teste	✓		
14	Obter <i>status</i> da API REST e planos de testes automatizados		✓	
15	Atualizar <i>status</i> dos testes	✓	✓	✓
16	Obter quantidade de <i>jobs</i> em execução e <i>status</i> do <i>job</i>	✓		
17	Modificar o <i>status</i> da sessão		✓	
18	Filtro de resultados (por exemplo, <i>Running</i> e <i>failed</i>)		✓	
19	Listar (<i>jobs</i> , <i>builds</i> , <i>tests</i>)	✓	✓	✓
20	Listagem completa dos <i>jobs</i> , <i>builds</i> e <i>tests</i>	✓		✓
21	Atualizar (<i>job</i> , <i>build</i> , <i>test</i>)	✓		✓
22	Remover (<i>job</i> , <i>build</i> , <i>test</i>)	✓	✓	✓
23	Listar <i>build</i> Específica			✓
24	Listar informações de um teste específico			✓
25	Listar todas as Sessões de uma <i>build</i>		✓	
26	Listar Seção		✓	
27	Remover Seção		✓	
28	Listar plataformas suportadas	✓		
29	Listar <i>browsers</i>		✓	✓
30	Limite de Resultados a serem exibidos	✓	✓	
31	Obter <i>jobs</i> a partir de um intervalo de tempo	✓		
32	Obter atividades do <i>job</i> em tempo real (concorrente)	✓		
33	Obter <i>jobs</i> em diferentes formatos (JSON e CSV)	✓		
34	Obter os <i>assets</i> do <i>job</i>	✓		
35	Download de arquivo <i>assets</i> do <i>job</i>	✓		
36	Remover <i>assets</i> do <i>job</i>	✓		
37	Listar <i>logs</i>	✓	✓	
38	Upload de arquivos temporários	✓		
39	Verificar arquivos temporários	✓		
40	Listar Projetos		✓	
41	Detalhar Projeto		✓	
42	Remover Projeto		✓	✓
43	Listar testes do TestLab			✓
44	Lista TestLab específico			✓
45	Atualizar TestLab			✓
46	Executar TestLab específico			✓
47	Interromper (<i>stop</i>) a execução de TestLab específico			✓
48	Executar todos os testes do TestLab			✓
49	Executar conjunto de TestLab específico			✓
50	Ignorar (<i>skip</i>) um <i>job</i> específico	✓		
51	Interromper (<i>stop</i>) a execução do <i>job</i> , <i>build</i> e <i>test</i>	✓		✓
52	Compartilhar resultado do teste			✓
53	Compartilhar <i>build</i> de testes			✓
54	Obter <i>Tunnel</i>	✓		✓
55	Listar <i>Tunnels</i>	✓		✓
56	Remover <i>Tunnel</i>	✓		✓
57	Obter data EOL Appium	✓		
58	Inicializar teste unitário em Javascript com diferentes navegadores	✓		
59	Obter <i>status</i> de testes unitários em Javascript	✓		
Total de Operações das APIs REST		37	16	26

testes UAT, sendo eles os padrões Singleton e PageObject. O padrão de projeto Singleton garante o controle e a existência de apenas uma única instância do RemoteWebDriver que é responsável pela comunicação com o fornecedor de TaaS. O padrão de projeto PageObject estabelece a criação de uma classe para cada página do sistema web (SUT), facilitando a manutenção do código e reduzindo a duplicação do código de teste.

Por fim, apesar de cada um dos fornecedores de TaaS utilizarem o Selenium, não existe um padrão de nomenclatura das capacidades comumente aceito entre eles. Os exemplos de código em Java necessários para executar os testes automatizados nos fornecedores de TaaS com as suas respectivas configurações de plataformas utilizados na abordagem convencional estão disponíveis na web³⁶.

2.6.1 Implementação da Abordagem Convencional - 1ª Parte

A abordagem convencional foi dividida em duas partes para facilitar a sua explicação. A primeira parte refere-se à escrita dos testes automatizados, configuração das capacidades e execução dos testes na nuvem. A segunda parte refere-se ao gerenciamento, coleta de informações e sumarização dos resultados de testes. A seguir, são descritas de forma detalhada a implementação da primeira parte da abordagem convencional apresentada no diagrama UML da Figura 14.

Figura 14 – Diagrama UML dos testes e carga de dados da abordagem convencional

Fonte: Autoria própria

- **UniversidadeSystemTest** é a classe de testes automatizados da aplicação SinAU (SUT) referentes ao CRUD (*Create, Update, Read e Delete*) da universidade conforme disponível

³⁶ Code examples: <http://www.ricardoramosdeoliveira.com.br/experiment/code_examples.pdf>

no Código fonte 6 do Apêndice E.1;

- **UniversidadePage** é a classe responsável por implementar o padrão de projeto `PageObject` com o objetivo principal de encapsular os atributos, métodos, campos e ações correspondentes a página web da universidade para serem utilizados pela classe de teste `UniversidadeSystemTest`;
- **Constantes** é a classe responsável por implementar as constantes que serão utilizadas pela classe `UniversidadePage`;
- **AbstractCarga** é uma classe abstrata responsável por realizar a conexão com o banco de dados remoto, executar comandos em SQL e disponibilizar os métodos abstratos `realizaCarga()` e `realizaDescarga()` que serão implementados pelas classes que estendê-la, como por exemplo, `UniversidadeCarga`;
- **UniversidadeCarga** é a classe responsável por realizar a carga e a descarga dos dados de teste por meio de comandos SQL da entidade `Universidade` permitindo que os testes sejam independentes entre si;
- **WebDriverSingleton** é a classe responsável pelas diferentes combinações e configurações das capacidades necessárias para a execução de teste na plataforma de nuvem. Essa classe é responsável por estabelecer a comunicação com o fornecedor de TaaS por meio do `RemoteWebDriver` que se encarrega de realizar toda a comunicação e interação com o fornecedor TaaS;
- **NovaUniversidadePage** é a classe responsável por localizar os elementos da página de cadastro de uma universidade e preencher os campos com dados enviados por parâmetros pelo teste `deveAdicionarUmaUniversidade()`. Cabe ressaltar, no entanto, que toda essa interação ocorre por meio do `Selenium WebDriver`;
- **WebDriver** é a interface do `Selenium WebDriver` responsável pela comunicação e interação com o navegador na execução dos testes automatizados sob a aplicação web;
- **PropriedadeUtils** é a classe responsável por criar e manipular todos os arquivos de configuração, como o arquivo `bd.config`, necessário para o acesso remoto ao banco de dados da aplicação `SinAU (SUT)`.

Os pacotes `br.usp.icmc.carga` e `br.usp.icmc.util`, contém as classes responsáveis tanto pela carga e descarga dos dados de teste da Universidade quanto as classes utilitárias, respectivamente. As classes utilitárias são utilizadas desde a construção de chamadas REST, codificação e decodificação (*parse*) de arquivos em JSON até a geração de relatórios. No entanto todas essas funcionalidades foram extraídas para uma biblioteca java com o objetivo principal de

facilitar o entendimento do código de teste ao reduzir a complexidade e a quantidade de pacotes e classes do projeto final.

A abordagem convencional é acionada a partir da execução do teste pelo JUnit. Por exemplo, o testador ao executar o teste `deveAdicionarUmaUniversidade()`, será executado o método `inicializa()` anotado com `@Before`. Esse método é responsável por preparar todas as condições necessárias para a execução do teste automatizado, como por exemplo, instanciar o `PageObject UniversidadesPage`. Caso não exista um *driver*, o `PageObject UniversidadesPage` instanciará um *driver* por meio do `WebDriverSingleton` com todas as configurações das capacidades e endereço do `RemoteWebDriver` necessários para a execução dos testes no fornecedor de TaaS remoto. O método `inicializa()` instância a classe `UniversidadeCarga`, responsável por utilizar a classe `PropriedadeUtils` para acessar as configurações do arquivo `bd.config` por meio da generalização da classe `AbstractCarga`, essa responsável por conectar com o banco de dados remoto e executar a carga de dados no SUT. Em seguida, o método `inicializa()` acessa a *url* principal da aplicação web por meio da classe `Constantes`.

Após executar o método `inicializa()`, o fluxo de execução retorna para o teste. Nesse momento é realizado o acesso da *url* para o cadastro de uma nova Universidade por meio da classe de `Constantes`. Em seguida, é instanciado o `PageObject NovaUniversidadePage` por meio do `WebDriver`. Posteriormente, a classe `NovaUniversidadePage` fica responsável pela localização dos elementos da página de cadastro, receber os dados de entrada do teste, preencher os campos do formulário de cadastro da página web e acionar o botão *submit* para a submissão do formulário pela aplicação `SInAU`, ou seja, todas as ações do usuário final são simuladas pelo TaaS.

Após realizar o cadastro de uma nova Universidade, o teste verifica por meio do método `assertTrue()` do JUnit se a Universidade realmente existe e foi cadastrada com sucesso no banco de dados e se as suas informações são exibidas na página de listagem da aplicação. Em seguida, o JUnit retornará o resultado do teste, verificando se passou ou falhou. Caso o teste finalize com alguma falha será necessário limpar o banco de dados pelo `MySQL Workbench`³⁷, pois a abordagem convencional não possui um mecanismo que realize a descarga automática de dados.

Caso o teste automatizado finalize e seja bem sucedido, será realizada a execução do método `finaliza()` anotado com a anotação `@After`. Esse método é responsável apenas por realizar a descarga dos dados, não sendo necessário fechar o navegador, pois ao finalizar a execução do teste, o `RemoteWebDriver` encerra a conexão com o fornecedor de TaaS, que por sua vez fecha o navegador e consequentemente encerra também a execução da máquina virtual alocada para o teste, concluindo assim todo o processo de teste na nuvem.

³⁷ *MySQL Workbench*: <<https://www.mysql.com/products/workbench/>>

2.6.2 Implementação da Abordagem Convencional - 2ª Parte

A Figura 15 apresenta o diagrama UML da implementação referente à 2ª parte da abordagem convencional responsável pelo gerenciamento, coleta de informações e sumarização dos resultados dos testes automatizados. A seguir, são descritas de forma detalhada cada uma das classes utilizadas na implementação.

Figura 15 – Diagrama UML das configurações dos fornecedores de TaaS da abordagem convencional

Fonte: Autoria própria

- **TesteRelatorio** é a classe que funciona como um ponto de partida para a geração de relatórios que serão utilizados, posteriormente, no gerenciamento dos resultados dos testes, conforme mostrado no exemplo de Código fonte 8 do Apêndice E.1. Essa classe é responsável também por executar a geração do relatório por meio da classe ReportUtils;

- **ReportUtils** é a classe responsável por coletar e consolidar todas as informações obtidas a partir das APIs REST e construir a página web formatada com os resultados para o engenheiro de software;
- **SaucelabsService**, **BrowserstackService** e **TestingbotService** são as classes que consomem as API REST proprietárias de cada um dos fornecedores de TaaS. Cada uma dessas classes é responsável pela obtenção dos resultados da última execução dos testes automatizados e obter todos os parâmetros que serão utilizados na geração do relatório final, com os resultados dos testes;
- **Rest** é a classe que oferece todo o suporte necessário para a construção de um cliente de web service RESTful, por meio da implementação de referência Jersey³⁸ da API JAX-RS do Java. Essa classe é responsável também por realizar todo o *parse*, ou seja, todo o processo de codificação e decodificação dos dados necessários à obtenção dos resultados dos testes retornados no formato JSON por cada um dos fornecedores de TaaS;
- **Java Beans** são as classes que representam as entidades dos serviços REST de cada fornecedor TaaS. Essas classes são utilizadas para a serialização e desserialização dos resultados de teste entre o consumidor e fornecedor de TaaS. Essas classes especificam como deve ser realizado o *parse* dos dados em JSON retornado pelo fornecedor de TaaS.

Os Java beans foram omitidos dos diagramas UML por questão de simplificação, pois são classes básicas que representam as entidades dos serviços REST e que contém apenas os atributos privados e os seus respectivos métodos getters e setters. Essas classes servem para encapsular, abstrair e transportar todas as informações de integração entre a aplicação cliente e o fornecedor da abordagem convencional.

O mecanismo de `TesteRelatorio` foi criado para acionar a geração do relatório da abordagem convencional. No entanto poderiam ser utilizadas outras abordagens além de um teste do JUnit, como por exemplo, uma classe simples com um método `main()`.

A abordagem convencional é acionada a partir da execução do teste pelo JUnit, quando o testador acessa a classe `TesteRelatorio` e executa o teste `testeRelatorioSaucelabs()`. Nesse momento, é criada uma instância da classe `SaucelabsService` que é responsável por passar por parâmetro o endereço do TaaS desejado e o *token* do engenheiro de software que está usando o serviço de teste. A classe `SaucelabsService` faz uso de uma instância da classe `Rest` que é responsável por criar as configurações básicas para as chamadas REST via JSON. Em seguida, é invocado o método estático `geraArquivoHTML()` da classe `ReportUtils` passando por parâmetro o caminho do *path* ou diretório onde o arquivo final será salvo e os demais parâmetros necessários para a geração do relatório de teste.

³⁸ Jersey: <<http://jersey.java.net/>>

Os demais parâmetros são obtidos a partir da chamada do método `getParametrosParaGeracaoArquivo()` por meio da instância do `SaucelabsService`. O método `getParametrosParaGeracaoArquivo()`, por sua vez, realiza a chamada do método `carregarUltimaExecucao()`, responsável por realizar as chamadas REST para o Saucelabs e coletar todas as informações referentes aos resultados da última execução dos testes automatizados, como por exemplo, obter as informações do navegador, sistema operacional, dispositivos, *screenshots* e links dos vídeos gravados. Finalmente todas essas informações serão setadas no Java beans do Saucelabs.

Todos os resultados dos testes serão extraídos a partir do respectivo Java Bean do Saucelabs para um mapa de `<String, String>` por meio do método `getParametrosParaGeracaoArquivo()`. Esse mapa contém todos os parâmetros necessários para a geração do relatório final. Por fim, todas as informações do mapa são repassadas para o método estático `geraArquivoHTML()` da classe `ReportUtils` por parâmetro finalizando assim a geração do relatório final dos resultados dos testes automatizados no formato HTML.

Antes de gerar o relatório final, todos os parâmetros necessários são obtidos por meio da instância do `SaucelabsService` pelo método `getParametrosParaGeracaoArquivo()` que por sua vez invoca o método `carregarUltimaExecucao()` que fará a chamada REST para obter todos os dados da última execução do fornecedor de TaaS, como por exemplo, obter as informações referentes ao navegador, sistema operacional, dispositivo, *screenshots* e *links* dos vídeos gravados para o teste automatizado. Todas essas informações serão setadas diretamente no Java Bean correspondente do fornecedor TaaS e ao final desse processo, todas as informações serão extraídas dos Java Beans pelo método `getParametrosParaGeracaoArquivo()` que, por sua vez, repassa todos os parâmetros necessários para o método `geraArquivoHTML()` da classe `ReportUtils`, finalizando, assim a geração do relatório final dos resultados dos testes automatizados.

Finalmente, o gerenciamento da execução, coleta de informações e sumarização dos resultados de teste para os fornecedores de TaaS Browserstack e Testingbot funcionam de forma semelhante ao que foi descrito para o Saucelabs.

2.7 Taxonomia Arquitetônica de Desenvolvimento na Nuvem

Grozev e Buyya (2014a) apresentam uma taxonomia arquitetônica de desenvolvimento entre nuvens. Entre as abordagens para o desenvolvimento de federações *intercloud*, os autores destacam classificações centralizadas e *peer-to-peer*. A primeira baseia-se em uma entidade central que gerencia a alocação de recursos entre fornecedores de nuvem, enquanto a segunda envolve uma comunicação direta entre os provedores da nuvem através de um consenso. No que diz respeito ao desenvolvimento de várias nuvens, os autores destacam dois outros modelos, nomeadamente o serviço *multi-cloud* e a biblioteca *multi-cloud*. No primeiro, um *broker* ou

camada intermediária executa e implanta uma aplicação entre fornecedores de nuvem, enquanto que o segundo geralmente usa um *broker* personalizado que fornece uma aplicação de todas as nuvens de forma unificada, o que facilita o uso de múltiplas nuvens.

A partir desses modelos e desafios de desenvolvimento de federação *intercloud* essas classificações foram adaptadas para discutir as possíveis soluções que exigem mais esforços de pesquisa em relação ao problema de *vendor lock-in* no TaaS. A Figura 16 ilustra as abordagens destacadas pelos autores com adaptações no campo de TaaS. Além disso, as Figuras 16 (a) e (b) mostram a federação *intercloud* centralizada e a federação *intercloud* ponto-a-ponto, respectivamente. As arquiteturas de serviços Multi-TaaS e a biblioteca Multi-TaaS são exibidas nas Figuras 16 (c) e (d), respectivamente.

Figura 16 – Abordagens de Federação e *multi-cloud* para TaaS

Fonte: Adaptado de Grozev e Buyya (2014a)

A solução ideal para resolver o problema de *vendor lock-in* no TaaS seria a criação de um padrão ou protocolo aberto único entre os diferentes fornecedores de TaaS no mercado. As Figuras 16 (a) e (b) mostram o desenvolvimento de uma Federação de Testes como Serviço (FTaaS), no entanto, exigiria um padrão aberto interoperável ou protocolo de interesse amplamente aceito pelos provedores de TaaS. O FTaaS permitiria a abstração de diferentes TaaS no mercado por uma especificação comum que descreveria um TaaS. A ideia seria semelhante ao CloudFoundry³⁹, um PaaS que promove a abstração de diferentes IaaS. No entanto a proposta é atualmente impraticável no contexto TaaS, devido à falta de consenso entre os fornecedores.

As soluções Multi-TaaS descritas nas Figuras 16 (c) e (d) permitem o manuseio do *vendor lock-in* de duas maneiras: por meio de um serviço como RESTful, que promove a abstração de complexidades e das especificidades de cada fornecedor TaaS e por meio do uso de uma

³⁹ CloudFoundry: <<https://www.cloudfoundry.org/>>

biblioteca pelos consumidores TaaS em que uma camada de abstração seria criada entre os diferentes provedores de TaaS (GROZEV; BUYYA, 2014a; TOOSI; CALHEIROS; BUYYA, 2014; GROZEV; BUYYA, 2014b).

A abordagem Multi-TaaS desenvolvida por este trabalho de doutorado corresponde à solução representada na Figura 16 (d). O objetivo principal é a identificação da forma como os impactos do *vendor lock-in* ocorre no processo de teste da nuvem e a agregação de características comuns e específicas dos provedores de TaaS como Saucelabs, Browserstack e Testingbot. Além disso, pretende-se futuramente publicar a biblioteca Multi-TaaS no repositório central do Maven e disponibilizá-la para toda a indústria e comunidade de pesquisadores interessados.

Cabe destacar que os provedores de TaaS não colaboram de forma mútua, não há consenso sobre a integração e a agregação de funcionalidades entre eles. Além disso, as diferentes funcionalidades dos TaaS não são independentes. A funcionalidade de cada provedor TaaS depende da execução completa do teste e do uso da plataforma específica do fornecedor. Portanto os testes automatizados não podem ser executados no Saucelabs para a obtenção do vídeo de sua execução através do Browserstack. Caso os recursos fossem independentes, os custos para os consumidores TaaS poderiam ser otimizados (GROZEV; BUYYA, 2014a; TOOSI; CALHEIROS; BUYYA, 2014).

2.8 Considerações Finais

O propósito deste capítulo foi apresentar uma visão geral sobre os principais conceitos que envolvem o teste de software, como a descrição dos critérios de teste e das técnicas funcional, estrutural e baseada em defeitos. Também foram descritos os conceitos e as principais características sobre os Testes de Aceitação do Usuário (UAT), e as principais ferramentas de automação de testes. Foram apresentadas as principais características relacionadas ao novo paradigma conhecido por computação em nuvem e os principais conceitos sobre modelos de serviço, modelos de implantação, portabilidade e interoperabilidade no contexto de nuvem. Em seguida, foram introduzidos os principais conceitos, técnicas e classificações do modelo de serviço TaaS. O modelo de serviço conhecido por Teste como Serviço (TaaS) não depende necessariamente do ambiente de computação em nuvem. Porém, atualmente, a maioria dos fornecedores TaaS executam os testes no ambiente de nuvem, devido às suas diversas vantagens. Foi discutido ainda o problema conhecido por *vendor lock-in* na computação em nuvem. Foram descritas também a implementação utilizada na abordagem convencional e a taxonomia proposta por Grozev e Buyya (2014a), para o desenvolvimento de aplicações *intercloud*, federação e *multi-cloud*. No próximo capítulo, será apresentada a arquitetura de teste da biblioteca Java Multi-TaaS desenvolvida para obter a portabilidade dos testes automatizados UAT e a sumarização dos resultados de teste entre os diferentes fornecedores de TaaS.

ARQUITETURA DE TESTE MULTI-TAAS

3.1 Considerações Iniciais

O ritmo acelerado em que a nuvem está sendo adotada por usuários e corporações, advém da nova mudança de paradigma na forma como a tecnologia é entregue e consumida. Por isso não só as empresas, mas também os engenheiros de software, precisam compreender melhor os diferentes tipos de testes na nuvem para a tomada de decisões. As equipes de teste devem equipar-se com estratégias viáveis para atenuar os riscos e problemas associados com a nuvem (BALARAJU, 2011).

A migração de testes na nuvem e de programas que consomem APIs REST dos fornecedores de TaaS possui custos elevados, tornando-se um enorme desafio para os engenheiros de software que contratam o serviço de teste na nuvem. Estratégias técnicas e de negócio devem ser adotadas em decorrência da escolha de um fornecedor de TaaS. Com o objetivo de auxiliar a tomada de decisão dos engenheiros de software foi desenvolvida a abordagem Multi-TaaS.

Neste capítulo serão utilizados três termos de forma dissociada, sendo eles, **abordagem**, **biblioteca** e **arquitetura** Multi-TaaS. A abordagem Multi-TaaS refere-se a solução proposta por este doutorado de forma abstrata, ampla e genérica assim como a abordagem convencional. O termo arquitetura Multi-TaaS refere-se à arquitetura da biblioteca Java desenvolvida no formato “.jar”. A biblioteca Multi-TaaS refere-se à instância ou a forma como a abordagem Multi-TaaS foi concebida de forma específica e concreta.

Este capítulo apresenta os conceitos relacionados à migração de testes e da integração REST responsável pelo gerenciamento dos resultados de teste da abordagem Multi-TaaS. Este capítulo está organizado da seguinte forma: na Seção 3.2 são apresentados os principais conceitos e termos sobre efetividade, eficiência, migração e esforço no ambiente de nuvem, fundamentais para o entendimento da abordagem desenvolvida e das vantagens associadas ao uso da biblioteca Multi-TaaS. A Seção 3.3 apresenta de forma detalhada todo o processo de teste na nuvem. Na

Seção 3.4 são descritas as principais etapas do processo de teste de software na nuvem que são afetadas pelo problema de *vendor lock-in* além de apresentar também a categorização desse problema em maior e menor grau, com base no processo de teste na nuvem. Na Seção 3.5 é apresentado todo o funcionamento da arquitetura de teste da biblioteca Multi-TaaS baseada em padrões de projeto, integração REST e nos arquivos de configurações necessários para reduzir os impactos do problema de *vendor lock-in* de maior e menor grau. Na Seção 3.6 são descritas as principais vantagens e desvantagens da biblioteca Multi-TaaS. A Seção 3.7 apresenta de forma simples e prática todos os procedimentos necessários para a utilização da biblioteca Multi-TaaS para testar outras aplicações web. A Seção 3.8 apresenta o protótipo e o estudo de caso desenvolvido como prova de conceito utilizado no experimento controlado e finalmente na Seção 3.9 são discutidas as considerações finais.

3.2 Métricas e estimativas de migração de software

Antes de detalhar a abordagem desenvolvida por esta pesquisa, bem como as vantagens oferecidas pelo uso da biblioteca Multi-TaaS além de facilitar o entendimento dos termos empregados neste trabalho, é importante definir os principais termos utilizados como base de cálculo das métricas e estimativas de migração de software: esforço, migração, efetividade e eficiência.

O primeiro conceito mensurado neste trabalho é o esforço. A estimativa de esforço pode contribuir para a tomada de decisão dos engenheiros de software quanto aos riscos do *vendor lock-in* associados aos fornecedores de serviço de testes na nuvem. A estimativa pode auxiliar também no planejamento de projetos de software, orçamentos, análises de custos e investimentos e estratégias de negócio de uma forma geral (SOMMERVILLE, 2010; JORGENSEN, 2014).

O processo de estimar o esforço de manutenção é desafiador, pois depende de vários fatores, dentre os quais estão o tamanho do projeto, o tipo de manutenção ou migração, a experiência dos sujeitos, o nível de familiaridade do programador com o sistema sendo mantido, a complexidade, a documentação e as tecnologias utilizadas no projeto (NGUYEN; BOEHM; DANPHITSANUPHAN, 2009).

A norma ISO25010 (2011) define esforço de forma ampla como o tempo produtivo associado a uma tarefa específica do projeto. A norma ISO25010 (2011) divide o esforço de duas maneiras: o esforço individual e o esforço de tarefa, no qual o esforço individual, como o próprio nome sugere, considera o esforço necessário de um engenheiro de software completar uma determinada tarefa e o tempo gasto para concluí-la. Esforço de tarefa corresponde ao esforço de uma equipe de desenvolvimento, ou seja, o valor acumulado do tempo gasto para concluir uma determinada tarefa de forma conjunta. Observa-se que ambos os conceitos de esforço estão diretamente relacionados com a produtividade e a eficiência, ou seja, a conclusão de um maior número de tarefas no menor espaço de tempo, de forma individual ou por equipe de

desenvolvimento. Neste trabalho será considerado apenas o esforço individual. O esforço pode ser definido também como o processo de prever o custo necessário para concluir uma atividade de desenvolvimento ou manutenção de software, podendo ser expresso, por exemplo, em termos de horas pessoa ou dinheiro (SOMMERVILLE, 2010; PRESSMAN, 2009).

No entanto, a norma ISO25010 (2011) não define especificamente o termo migração, porém caracteriza a portabilidade com subcaracterística de adaptabilidade semelhante ao termo migração. A adaptabilidade possui o objetivo principal de medir atributos do comportamento do sistema ou do usuário que está tentando adaptar o software para diferentes ambientes específicos. Conclui-se, de acordo com a norma ISO25010 (2011) que a migração (ou adaptação) é uma subcaracterística da portabilidade, podendo ou não ser realizadas manutenções no software que está sendo movido de uma plataforma para outra.

Enquanto a portabilidade é um atributo de qualidade de software (ISO25010, 2011), segundo o qual, uma aplicação só é considerada portátil se ela pode ser executada em diferentes plataformas. A migração envolve a modificação ou manutenção do software para que ele se adapte e execute em um novo ambiente, ou seja, é o processo de mover um software de uma plataforma para outra (DELANGE *et al.*, 2015). A migração pode envolver diferentes etapas e atividades, como por exemplo, levantamento de requisitos de migração, escolha das ferramentas necessárias para realizar a migração, conversão de produtos e dados, execução, verificação e suporte. Além disso, pode envolver também atividades adicionais, como o *backup* dos dados do software antigo (SOCIETY; BOURQUE; FAIRLEY, 2014).

Por fim, adotam-se dois conceitos vinculados à qualidade de uso da abordagem por programadores: efetividade e eficiência. A efetividade, de acordo com as normas ISO25066 (2016) e ISO25010 (2011), refere-se à conclusão de uma determinada tarefa de forma adequada e se os objetivos foram alcançados independentemente dos custos.

A eficiência, de acordo com a norma ISO25066 (2016) e ISO25010 (2011), refere-se ao uso de recursos utilizados pelo usuário para alcançar as suas metas ou objetivos com precisão e de forma otimizada. Este termo está associado também com a produtividade de produtos e serviços. Nesse contexto, assume-se a premissa de que o tempo é um recurso. Assim, a eficiência permite alcançar os melhores resultados ou desempenho no menor espaço de tempo. Além disso, a norma ISO25010 (2011) estabelece que a eficiência é um atributo de qualidade do software referente à capacidade do software oferecer o desempenho adequado de acordo com os recursos disponíveis. Conforme as definições de eficiência descritas, utilizou-se o tempo de conclusão da tarefa do experimento controlado como principal recurso e indicador de eficiência.

Espera-se que a abordagem Multi-TaaS permita reduzir os impactos ocasionados pelo problema de *vendor lock-in* no processo de teste na nuvem e aumentar a portabilidade dos testes automatizados e do componente de sumarização dos resultados dos testes, facilitando o processo de migração entre os diferentes TaaS. Passaria a ser um processo muito simples, que envolve apenas a modificação de um arquivo de configuração. Na abordagem convencional, onde os

testes têm uma baixa portabilidade, dado o problema de *vendor lock-in*, é necessário realizar todo o processo de migração de forma manual, com bastante mudanças no código dos testes automatizados UAT e no componente de sumarização dos resultados.

Por fim, a dificuldade foi avaliada de forma subjetiva neste trabalho, apenas para verificar o quão difícil foi na opinião dos participantes do experimento, realizar as tarefas de migração de um fornecedor TaaS para outro.

3.3 Processo de Teste na Nuvem

A ocorrência do problema de *vendor lock-in* no processo de TaaS pode afetar negativamente os seus consumidores, tornando-se um desafio para os engenheiros de software. Por isso, torna-se importante entender todo o processo de teste na nuvem. A partir da Figura 17 é possível visualizar 3 entidades, sendo elas, o consumidor TaaS, fornecedor TaaS e a aplicação em teste (SUT). O consumidor TaaS é responsável pela escrita dos testes, configuração das capacidades do Selenium e gerenciamento dos resultados. Os fornecedores de TaaS são responsáveis por realizar a execução dos testes sobre a aplicação em teste e sumarização dos respectivos resultados de teste. Por último, mas não menos importante, temos a própria aplicação web em teste que é submetida às execuções de testes automatizados.

A Figura 17 apresenta também as principais etapas do processo de teste na nuvem. Na primeira etapa, o engenheiro de software, tratado aqui como consumidor TaaS, escreve os testes automatizados e define as configurações das capacidades; em seguida, na segunda etapa, o consumidor TaaS utiliza o Selenium a partir do JUnit para a interação da execução com o fornecedor TaaS, realizando assim o teste automatizado na aplicação web. Durante a execução o JUnit retorna à situação dos testes, por exemplo, passou ou falhou. Após a execução dos testes, via API REST, o consumidor TaaS obtém os resultados processados pelo fornecedor TaaS, concluindo o processo de teste na nuvem. Por último a etapa de configurações do ambiente de teste, paralelismo e Máquinas Virtuais (VM) do fornecedor de TaaS, cabe ressaltar que essa etapa é independente das demais e pode ser realizada a qualquer momento durante o processo de teste na nuvem.

Durante o desenvolvimento do projeto de doutorado, o problema de *vendor lock-in* no processo de automatização de testes na nuvem foi classificado em maior e menor grau. O consumidor TaaS é o responsável pela resolução do problema de *vendor lock-in* tanto de maior quanto de menor grau. O *vendor lock-in* de menor grau refere-se ao impacto ocasionado na escrita dos testes, configuração das capacidades do Selenium e execução dos testes automatizados. Já o *vendor lock-in* de maior grau ocorre nas configurações de máquinas virtuais alocadas para os testes paralelos e principalmente no gerenciamento dos resultados dos testes, devido às diferenças significativas das APIs REST dos fornecedores de TaaS. Na próxima seção, descreve-se como o problema de *vendor lock-in* afeta o processo de automatização de testes no ambiente de nuvem

Figura 17 – Etapas do Processo de Teste na Nuvem

Fonte: Autoria própria

conforme o seu respectivo grau.

3.4 O Problema de *Lock-in* no contexto de Teste como Serviço

Um dos principais fatores que ocasionam o problema de *vendor lock-in* é a ausência de padronização de serviços, que ocorre quando cada fornecedor oferece funcionalidades diferentes ou disponibilizam as mesmas funcionalidades de forma diferente. Um exemplo disso ocorre quando o *upload* de arquivos temporários oferecido pelo Saucelabs por um período de 7 dias está indisponível no BrowserStack e no Testingbot. Apesar dos três fornecerem a mesma funcionalidade de obter o *status* do teste por meio da API REST e retornarem a resposta do *status* no formato JSON, a estrutura de dados da resposta em JSON é totalmente diferente.

Nesse mesmo processo, em relação aos serviços oferecidos pelos fornecedores TaaS, verificam-se casos em que, por exemplo, o tamanho do *screenshot* e do vídeo são diferentes, prejudicando o desempenho do software intermediário que consome a API REST do fornecedor de TaaS, o que depende também da estratégia adotada por meio de *downloads* ou *links*. Em que pese a diferença entre os serviços de TaaS ser um processo natural, ocasionado por questões de negócio na obtenção de um diferencial que contribui para a competitividade, torna-se prejudicial para a adoção do TaaS na nuvem, o que consequentemente inviabiliza financeiramente a troca de fornecedor, resultando no problema de *vendor lock-in*. Neste contexto, a presente pesquisa identificou as principais etapas do processo de testes automatizados no ambiente de nuvem que são afetadas pelo problema de *vendor lock-in* devido à troca de fornecedores de serviço de teste, além de classificar e categorizar o problema de *vendor lock-in* em menor e maior grau, conforme

ilustrado na Figura 17 e descrito a seguir:

1. *Vendor Lock-in* de Menor Grau

- 1.1. **Impacto do *vendor lock-in* na Escrita de Teste:** ocorre devido à diferença na escrita das capacidades do Selenium, pois cada fornecedor impõe o seu padrão de escrita e consequentemente de execução, conforme os exemplos de códigos de testes ilustrados nos Apêndices D.1 e G.1;
- 1.2. **Impacto do *vendor lock-in* na Execução dos Testes:** apesar de os fornecedores utilizarem o Selenium Server para a execução dos testes, a maneira como cada fornecedor configura as suas capacidades do Selenium impacta diretamente na codificação e também na execução dos testes de uma forma geral;

2. *Vendor Lock-in* de Maior Grau

- 2.1. **Impacto do *vendor lock-in* no Gerenciamento dos Testes:** ocorre por causa de informações diferentes entre os fornecedores de TaaS. Apesar de os TaaS's oferecerem o mesmo tipo de serviço, por exemplo *screenshot*, a forma como eles disponibilizam esses serviços por meio das suas APIs REST é diferente, sendo necessário gerar programas diferentes para o gerenciamento e sumarização dos resultados para cada um dos fornecedores de TaaS;
- 2.2. **Impacto do *vendor lock-in* na Configuração dos Testes:** ocorre devido às diferentes informações de configuração entre provedores de TaaS no que diz respeito ao ambiente de teste. Informações sobre as configurações do ambiente, como a seleção do número de máquinas virtuais (VMs) alocadas para os testes paralelo e execução simultânea, todas essas informações podem ser recuperadas e alteradas por meio das capacidades do Selenium ou das APIs REST dos fornecedores TaaS.

É evidente que todos os processos referentes a **execução, gerenciamento e configuração** são etapas que podem ser automatizadas. Há atividades que podem ser executadas por computadores. Já a escrita dos testes pode ser realizada tanto de forma automatizada, por meio de geração automática de casos de teste, quanto de forma manual, por um ser humano, na qual o engenheiro de software escreve e configura manualmente o teste para execução no ambiente de nuvem. No decorrer desta tese, o uso do termo “escrita de teste” refere-se ao processo manual.

Pensando na conveniência para os testadores, os fornecedores de TaaS criaram um conjunto de capacidades específicas do Selenium. Porém devido às customizações das capacidades e das configurações de paralelismo surge o problema de *vendor lock-in*, categorizado nessa pesquisa como sendo de menor grau. É importante ressaltar, no entanto, que as configurações das capacidades fazem parte da escrita e da execução dos testes automatizados na nuvem (por exemplo: qual o navegador, versão do navegador, *timeout* dos testes, *devices*, S.O. etc).

Outros tipos de configurações são realizados por meio da interface gráfica do TaaS (área do usuário) ou geralmente via API do TaaS (exemplos - paralelismo, quantidade de máquinas virtuais, enviar e-mail ao fim da execução, permissões, *screenshots* a cada X segundos, video/gravação da execução dos testes, etc.). As configurações das capacidades não têm nenhuma relação direta com o Selenium. O Selenium fica responsável apenas por guiar e operar os navegadores na execução dos testes UAT.

O objetivo principal desta tese é reduzir os impactos ocasionados pelo problema de *vendor lock-in* de menor e maior grau em todo o processo de teste na nuvem de forma semelhante ao Selenium Webdriver. O Selenium já padroniza uma parte da escrita dos testes por meio da especificação do Webdriver, que permite escrever testes independentemente da implementação dos navegadores (FirefoxDriver ou ChromeDriver). Se não houvesse essa padronização seria necessário escrever um tipo de teste para usar o Firefox e outro teste para usar o Google Chrome. O Webdriver cria uma camada de abstração que permite que os desenvolvedores ou testadores possam escrever os seus testes independentemente da implementação do navegador (SELENIUMHQ, 2017).

A abordagem desenvolvida por este trabalho, intitulada “Multi-TaaS”, funciona de forma análoga ao Selenium, basicamente como se fosse um Selenium Webdriver para TaaS. Conforme ilustrado na Figura 18, é possível perceber que o Selenium estabelece a portabilidade dos testes automatizados UAT entre os diferentes navegadores. Em contrapartida, a abordagem Multi-TaaS estabelece a portabilidade dos testes automatizados UAT e da sumarização dos resultados de testes por meio da API REST entre os diferentes TaaS.

Figura 18 – Analogia do Selenium Webdriver com a abordagem Multi-TaaS

Fonte: Autoria própria

3.5 Arquitetura da biblioteca Multi-TaaS

Nesta seção, são apresentados o desenvolvimento e o funcionamento básico da arquitetura de teste Multi-TaaS. Além disso, será apresentado também a maneira como a arquitetura desenvolvida permite auxiliar os engenheiros de software na tomada de decisão em relação à troca de fornecedores do Saucelabs para o Browserstack ou outras plataformas de TaaS, como por exemplo, o Testingbot.

Ao comparar o código de teste dos fornecedores Saucelabs, Browserstack e Testingbot da abordagem convencional descrita na Seção 2.6, evidenciou-se o problema do *vendor lock-in* de menor e maior grau. O *vendor lock-in* de menor grau está presente na escrita das capacidades descritas na Subseção 2.5.2 e na criação do `RemoteWebDriver`, responsável por disponibilizar as configurações das capacidades para o fornecedor TaaS.

As configurações das capacidades bem como a criação do `RemoteWebDriver` fazem parte da escrita dos testes automatizados e estão diretamente relacionados com a execução dos testes no ambiente de nuvem. Já o *vendor lock-in* de maior grau foi detectado ao realizar o mapeamento das APIs REST proprietárias de cada um dos fornecedores de TaaS, descrito na Subseção 2.5.3. Após obter todas as evidências dos impactos ocasionados pelo problema de *vendor lockin* de maior e menor grau no processo de teste na nuvem, foi desenvolvida a abordagem Multi-TaaS, a partir da implementação da abordagem convencional descrita na Seção 2.6.

Inicialmente, foi desenvolvida uma arquitetura de teste inicial utilizando padrões de projeto e arquivos de configuração para que a biblioteca Java Multi-TaaS não dependesse das informações *hard coded*, permitindo assim o desacoplamento do *script* de teste da execução do teste na plataforma de nuvem. Posteriormente, a arquitetura foi evoluída e adaptada de modo a permitir a sua flexibilização e extensão.

3.5.1 Visão Geral da Arquitetura Multi-TaaS

A abordagem Multi-TaaS desenvolvida neste trabalho é adequada para os engenheiros de software e gerentes de controle de qualidade na construção de uma solução que seja portátil, particularmente no contexto de grandes empresas que possuem diversos projetos.

Ambas as abordagens Multi-TaaS e convencional foram desenvolvidas na linguagem de programação Java. A plataforma Java foi escolhida simplesmente por ser *open source* e pelo doutorando ter maior experiência com a linguagem Java e inclusive possuir a certificação *Sun Certified Java Programmer SE 6 (SCJP6)*. A linguagem de programação Java é amplamente utilizada pela academia, além de ser aceita e utilizada no mundo empresarial e estar inserida no contexto de grandes empresas. Além disso, a linguagem de programação Java foi escolhida principalmente pela sua robustez e portabilidade entre múltiplas plataformas, ou seja, diferentes sistemas operacionais.

É importante destacar que a abordagem Multi-TaaS funciona como uma camada de abstração desenvolvida por meio de uma biblioteca Java no formato “.jar”. A arquitetura da solução adotada permite abstrair e flexibilizar a troca de fornecedores de TaaS de forma portátil, pois permite encapsular toda a complexidade da implementação do desenvolvedor/testador referente a comunicação e integração entre consumidor e fornecedor TaaS. Sendo assim, a abordagem Multi-TaaS permite desacoplar o teste automatizado de qual plataforma TaaS ele será executado, bem como abstrair os aspectos da comunicação e integração entre as APIs REST proprietárias dos diferentes fornecedores de TaaS. Além disso, a abordagem Multi-TaaS possibilita sumarizar os resultados dos testes automatizados de forma independente das tecnologias da plataforma TaaS subjacente.

O arquivo de configuração chamado de `taas.config`, conforme mostrado na Figura 19 e no Apêndice F.1, possui as principais vantagens de ser independente de linguagens de programação e sintaxe muito semelhante à escrita de um arquivo de propriedades chave valor. Cada linha do arquivo de configuração representa as capacidades com as suas respectivas chaves e valores associadas. Sendo assim, para realizar a troca de fornecedor, basta modificar o valor da capacidade `taas_servico`. Se o valor estiver definido como “saucelabs” basta mudar para “browserstack” juntamente com as demais capacidades desejadas que os testes usarão automaticamente o novo fornecedor, nesse caso o Browserstack.

A arquitetura da biblioteca Multi-TaaS funciona conforme ilustrado na Figura 19. O engenheiro de software escreve o teste utilizando o JUnit e recebe o `RemoteWebDriver` criado e configurado pela biblioteca Multi-TaaS. Durante a execução do teste (*runtime*), a biblioteca Multi-TaaS instancia o `RemoteWebDriver` baseando-se nas capacidades descritas no arquivo de configuração `taas.config`. Dessa forma, uma das principais contribuições da abordagem Multi-TaaS é o desacoplamento entre o código de teste do JUnit e da plataforma TaaS onde o teste será executado. Os padrões de projeto representados na Figura 19, na etapa 1, são responsáveis por abstrair os detalhes e especificidade de cada fornecedor TaaS referentes às configurações das capacidades e da execução dos testes automatizados na nuvem. Na etapa 2, eles permitem abstrair toda a complexidade da comunicação e integração REST com os diferentes fornecedores de TaaS na sumarização dos resultados dos testes automatizados.

A abordagem poderia ser estendida para que ocorra a troca de fornecedores TaaS em tempo de execução, caso um determinado fornecedor esteja sobrecarregado ou com problemas de disponibilidade, sendo necessário apenas implementar o mecanismo de paralelização para que cada teste possa ser executado em um processo independente do Sistema Operacional utilizado na máquina do engenheiro de software.

Antes de detalhar todos os componentes do artefato da arquitetura de teste, cabe ressaltar que a arquitetura foi dividida em duas etapas referentes ao número de vezes em que ocorre o acesso à plataforma TaaS, conforme mostrado nas Figuras 19 e 20. A divisão em etapas possibilita também facilitar a explicação.

Figura 19 – Funcionamento básico da arquitetura Multi-TaaS a partir da perspectiva do teste automatizado

Fonte: Autoria própria

A primeira parte da arquitetura é referente à configuração das capacidades, escrita e execução dos testes automatizados. Na Figura 20, é possível visualizar que o engenheiro de software fica responsável por escrever os testes automatizados juntamente com as configurações necessárias das capacidades. A execução dos testes automatizados no ambiente de nuvem ocorre quando o engenheiro de software aciona o JUnit, que é responsável por orquestrar a execução do Selenium Webdriver.

Em seguida, a arquitetura utiliza a API do Selenium Webdriver para realizar a comunicação remota com o fornecedor de TaaS e envia os comandos necessários para a execução dos testes. O fornecedor de TaaS, por sua vez, recebe os comandos do Selenium e executa os testes simulando um usuário real, abrindo o navegador, preenchendo campos, clicando em botões e interagindo com a aplicação e utilizando toda a infraestrutura de nuvem. Após a execução dos testes, o fornecedor de TaaS retornará os resultados para o JUnit, que verifica as informações via *assert* dos comandos executados pelo Selenium Server.

Já a segunda parte do funcionamento da arquitetura de teste é referente ao gerenciamento dos resultados de testes composto pela coleta e sumarização dos resultados. A Figura 20 mostra

Figura 20 – Arquitetura da biblioteca Java Multi-TaaS

Fonte: Autoria própria

o engenheiro de software acionando o mecanismo de sumarização dos resultados dos testes automatizados por meio do JUnit. A arquitetura possui o componente de integração de serviços TaaS, responsável por solicitar os dados de execução e gerar o relatório a partir da API REST de cada fornecedor. Todas as partes da arquitetura são descritas detalhadamente nas próximas subseções.

3.5.2 Funcionamento da Arquitetura Multi-TaaS

O funcionamento da abordagem Multi-TaaS foi dividido em três partes para facilitar a sua explicação. A primeira parte refere-se à utilização da arquitetura para a escrita dos testes automatizados e da carga e descarga dos dados de teste. A segunda parte refere-se às configurações das capacidades e execução dos testes na nuvem. A terceira e última parte refere-se ao gerenciamento, coleta de informações e sumarização dos resultados de testes. A seguir será descrita, de forma detalhada, a implementação da primeira parte da abordagem Multi-TaaS apresentada no diagrama de classes da Figura 21.

3.5.2.1 Escrita dos testes automatizados e da carga e descarga dos dados de teste

Todos os mecanismos relacionados à automatização dos testes e de carga e descarga de dados da abordagem Multi-TaaS são mostrados na Figura 21 e descritos com maiores detalhes

na listagem a seguir.

Figura 21 – Diagrama de classes da arquitetura Multi-TaaS

Fonte: Autoria própria

- **@Carga** é uma anotação usada para especificar os metadados, informações de que a classe anotada faz uso de reflexão para extrair os valores da anotação e realizar a carga e descarga dos dados de teste em tempo de execução, independentemente de qualquer classe ou objeto;
- **PluginRunner** é uma extensão do JUnit, criada para permitir a utilização da arquitetura Multi-TaaS, responsável por orquestrar e controlar a execução dos testes automatizados. Além disso, a classe PluginRunner também faz uso de reflexão do Java para examinar os métodos marcados com a anotação @Carga e, em seguida, instanciar a(s) classe(s) passada(s) por parâmetro na anotação @Carga e executar a carga de dados para o teste, ou seja, a inserção de dados no banco de dados do sistema web. Após a execução dos testes, a anotação @Carga também se encarrega de realizar a descarga do banco de dados de forma automática. Caso aconteça algum erro na execução do teste, a arquitetura consegue também efetuar a descarga do sistema automaticamente, permitindo que os testes sejam independentes entre si;

- **AbstractCarga** é a classe abstrata responsável por realizar a conexão com o banco de dados da aplicação além de permitir executar comandos em SQL. Essa classe possui também a assinatura dos métodos `realizaCarga()` e `realizaDescarga()` que serão utilizados pela classe `PluginRunner` na carga e descarga dos dados de teste, respectivamente;
- **ConstantesInicializacao** são as constantes referentes às pré-configurações de inicialização da arquitetura, como por exemplo, a leitura de informações da *url* da aplicação a partir do arquivo `startup.config`. As configurações de *startup* e de *exception* são utilizadas pelo `PluginRunner` por meio dos métodos `avaliarStartup()` e `avaliarExceptionCatch()`, que são responsáveis por realizar o acesso à aplicação e manter os *scripts* independentes entre si durante a execução. Quando um teste falhar não vai impactar na execução dos outros testes, respectivamente;
- **ConstantesBancoDeDados** são as constantes que representam as informações do endereço da *url* do banco, usuário e senha de acesso que são extraídas a partir do arquivo `bd.config` e que posteriormente serão utilizadas pela classe `AbstractCarga` na construção da conexão com o banco de dados;
- **ConstantesTaaS** são as constantes extraídas do arquivo `taas.config`, permitindo assim, um melhor gerenciamento das configurações das capacidades na migração entre fornecedores de TaaS;
- **PropriedadeUtils** é a classe responsável pela criação e manipulação de todos os arquivos de configuração, como o arquivo `bd.config` necessário para o acesso ao banco de dados da aplicação;
- **WebDriverSingleton** é a classe responsável por implementar o padrão de projeto Singleton e manter apenas uma única instância do `RemoteWebDriver` no conjunto de testes automatizados. Além disso, essa classe delega a instanciação do `RemoteWebDriver` para uma *Factory* que conhece os detalhes de configuração de cada fornecedor, como por exemplo, as capacidades do Selenium específicas de cada fornecedor;
- **AbstractPageObject** é uma classe abstrata responsável por inicializar os `WebElements` dos `PageObjects` de acordo com os parâmetros definidos na anotação `FindBy`;
- **AbstractCenario** é a classe responsável por instanciar os `PageObjects` por meio do método `UsuarioEm()` via mecanismo de reflexão do Java, ou seja, definindo o tipo do retorno em tempo de execução. Além disso, essa classe permite melhorar também a legibilidade do código de teste;
- **ObjetoChecado** é uma classe utilitária que evita os alertas (*Warnings*) de objeto não checado, centralizando a conversão forçada quando ela é usada.

É importante destacar que o `PluginRunner` é a classe central que foi criada para possibilitar a utilização da arquitetura, permitindo maior controle sobre os testes. Conforme descrito anteriormente, o `PluginRunner` é uma extensão do `JUnit`, responsável pelo gerenciamento da execução dos testes e geração de relatórios. Com a sua utilização, torna-se necessário o uso de um padrão de escrita de testes que facilite o trabalho do engenheiro de software. Dessa forma, o *plugin* desenvolvido não necessita da utilização do `@Before` e nem do `@After` do `JUnit`, podendo ser utilizadas opcionalmente.

De forma análoga à abordagem convencional de testes automatizados na nuvem descrita na Seção 2.6, foram extraídas as funcionalidades de carga e descarga dos dados de teste em SQL do pacote `br.usp.icmc.carga` para uma biblioteca java independente. A biblioteca de carga dos dados foi importada para o projeto posteriormente, com o objetivo principal de facilitar o entendimento do código de teste e reduzir o número de pacotes e classes para não confundir os alunos, permitindo assim, que os participantes focassem apenas no que realmente era necessário para a realização das tarefas do experimento.

Quando o engenheiro de software executar o teste utilizando a abordagem Multi-TaaS mencionada acima, a arquitetura realizará a leitura do arquivo `startup.config` e executará o método de inicialização configurado (`acessar()`), permitindo assim que o navegador seja inicializado e acesse a *url* principal da aplicação web.

Todos os testes automatizados da abordagem Multi-TaaS estendem de `AbstractCenario`, permitindo assim, aumentar a legibilidade do código de teste e instanciar os `PageObjects` por meio do método `UsuarioEm()`. Além disso, todos os `PageObjects` estendem de `AbstractPageObject`, responsável por inicializar os `WebElements` por meio do método `deve()`, de acordo os parâmetros definidos nas anotações `FindBy` dos `PageObjects`.

3.5.2.2 Configurações das capacidades e execução dos testes na nuvem

Para mitigar o problema de *vendor lock-in* na configuração das capacidades e execução dos testes automatizados, foram combinados dois padrões de projeto na implementação da arquitetura Multi-TaaS, sendo eles, *Abstract Factory* ou simplesmente *Factory*, e *Strategy*, descritos por *Gang of Four* (GoF) como padrões de categorias de criação e comportamento respectivamente (GAMMA *et al.*, 1995).

O padrão de projeto *Strategy* representado pelas classes `SaucelabsWebDriver`, `BrowserstackWebDriver`, `TestingbotWebDriver`, `RemoteWebDriverInterface` e `ConstantesTaaS` foi utilizado para separar os comportamentos e isolar os fornecedores conforme ilustrado na Figura 22. Além disso, o padrão de projeto *Strategy* permite que o comportamento de cada um dos fornecedores possa variar de maneira transparente para os clientes que utilizam o serviço de teste, diminuindo o acoplamento entre consumidor e fornecedor, favorecendo a manutenção do código e permitindo também a adição de novos fornecedores de forma flexível.

O padrão de projeto *Factory* é representado pelas classes *Vendor* e *RemoteWebdriverFactory*, conforme ilustrado também na Figura 22. A implementação da *Factory* contém o método `getWebDriverByVendor(String vendor)` que dado um fornecedor passado por parâmetro, cria em tempo de inicialização uma instância do Selenium *RemoteWebDriver* do respectivo fornecedor e retorna o *driver* devidamente configurado com base no arquivo de configuração `taas.config` para a classe *WebDriverSingleton* que se encarrega de utilizar o fornecedor específico.

Conforme mostrado na Figura 22, as implementações de *RemoteWebDriverInterface* representadas pelas classes *SaucelabsWebDriver*, *BrowserstackWebDriver* e *TestingbotWebDriver* são responsáveis pela manipulação do navegador e do fornecimento das capacidades necessárias para a execução dos testes pelo fornecedor de TaaS.

A estratégia usada permite abstrair o fornecedor por meio dos padrões de projeto e do arquivo de configuração `taas.config`, criando uma dependência única por meio da classe *ConstantesTaaS*, representada na Figura 22. Em contraste com a escrita de capacidades *hard coded*, em que ao criar as capacidades específicas para um determinado fornecedor, diretamente no código-fonte da aplicação, resulta no aumento do acoplamento e da dependência com o serviço de teste contratado. Essa dependência eleva os custos envolvidos na migração de um fornecedor para outro, devido à manutenção, reescrita e reengenharia das diferentes combinações das capacidades e também do código de teste, tornando o processo de migração muito mais caro e oneroso.

A combinação dos padrões de projeto *Strategy* e *Factory* permite também flexibilizar a arquitetura Multi-TaaS quanto à adição de novos fornecedores TaaS, sem a necessidade de modificar o código da *Factory* e nem do *RemoteWebDriverFactory*. Dessa forma, os detalhes de implementação e criação de cada um dos fornecedores de TaaS são encapsulados pela *Factory* que sempre referencia a interface *RemoteWebDriverInterface*. Esse mecanismo é conhecido por polimorfismo (ORSO; SILVA, 1998).

Para adicionar novos fornecedores futuramente, como por exemplo, o *Browsershots*¹ basta criar uma classe *Browsershots.java* que implemente a interface *RemoteWebDriverInterface* por meio do método `getRemoteWebDriver()` com as propriedades de configuração do fornecedor, conforme os outros fornecedores ilustrados na Figura 22. Em seguida, deve-se adicionar o atributo *Browsershots* na classe Enum *Vendor* e implementar o método `getWebDriver()` com o seu respectivo comportamento.

A *Factory* é responsável por definir o fornecedor a partir da constante `SELENIUM_SERVICE` presente na classe *ConstantesTaaS*, ilustrada na Figura 22. Cada constante possui uma chave com o seu respectivo valor associado no arquivo de configuração, como a chave “`taas_servico`” corresponde à constante `SELENIUM_SERVICE`.

¹ *Browsershots*:
<<http://browsershots.org/>>

Figura 22 – Diagrama de classes dos Padrões de Projeto *Abstract Factory* e *Strategy*

Fonte: Autoria própria

A instância da classe *ConstantesTaaS* fica responsável por utilizar a instância da classe *PropriedadeUtils* para ler e carregar o valor da chave “taas_servico” do arquivo de configuração para a constante *SELENIUM_SERVICE*. Se o valor da chave *taas_servico* estiver com o valor “saucelabs”, os testes usarão automaticamente a instância do *RemoteWebDriver* criada para o Saucelabs. Para trocar de fornecedor, basta configurar o valor da chave *taas_servico* no arquivo de configuração juntamente com as demais configurações das capacidades desejadas, como por exemplo, *taas_nome_browser* (configuração do navegador), *taas_versao_browser* (configuração da versão do navegador) e *taas_sistema_operacional* (configuração do sistema operacional) que os testes automaticamente utilizarão o novo fornecedor.

Resumindo, a *Factory* cria o *driver* do *RemoteWebDriver* para o respectivo fornecedor de TaaS com base no arquivo de configuração. Em seguida, a classe *WebDriverSingleton* recebe

o *driver* devidamente configurado da Factory e disponibiliza-o para os testes automatizados.

A execução dos testes na plataforma do Saucelabs, Broserstack, Testingbot ou outras plataformas de TaaS é determinada apenas pela correta atribuição da chave “*taas_servico*” no arquivo de configuração *taas.config*. Essa configuração torna o código genérico, pois a estratégia adotada para abstrair o fornecedor depende do valor da chave “*taas_servico*”, que referencia o fornecedor correto, permitindo consequentemente o desacoplamento dos *scripts* de teste da plataforma TaaS em que ele será executado. Dessa forma, os testes tornam-se independentes da execução dos fornecedores de TaaS remoto.

Portanto a nova abordagem permite auxiliar os engenheiros de software na tomada de decisão em relação à configuração de capacidades, escrita e execução de testes automatizados, quanto à troca de fornecedores entre o Saucelabs, Browserstack e Testingbot, entre outros. A abordagem Multi-TaaS possibilita também contornar o problema de *vendor lock-in* de menor grau descrito anteriormente, facilitando a troca de fornecedores TaaS e, ao mesmo tempo, permitindo aumentar o reuso e melhorar a legibilidade do código de teste. Fica a cargo do engenheiro de software realizar apenas o gerenciamento adequado das informações do arquivo de configuração, vantagens que permitem que o modelo de serviço TaaS na computação em nuvem seja amplamente adotado, de forma consciente, no futuro.

3.5.2.3 Gerenciamento de execução, coleta de informações e sumarização dos resultados de testes

Devido à limitada capacidade humana de gerenciar os defeitos de um software, constantes manutenções do software para atender os requisitos dos clientes e à crescente demanda por novos produtos e serviços, torna-se extremamente importante o processo de gerenciamento de teste para o sucesso do projeto. As atividades de gerenciamento de teste do sistema possuem o principal objetivo de organizar e controlar todo o processo de teste de software de forma eficaz. Entre as diversas atividades importantes desse processo, pode-se destacar o gerenciamento de casos de teste, configuração, planejamento de teste, automação de testes, execução do teste e relatórios de defeitos entre outras atividades envolvidas (SHAH, 2014).

Conforme descrito no Capítulo 2, as fases do teste funcional costumam ser divididas em testes unitários, de integração, sistemas, aceitação e regressão (SOMMERVILLE, 2010; PRESSMAN, 2009). Para cada fase de teste é definido um processo de teste que envolve basicamente quatro etapas: Planejamento de Testes, Projeto dos Casos de Teste, Execução do Teste e a coleta e avaliação dos resultados dos testes (IEEE, 2008).

O gerenciamento de execução, coleta de informações e sumarização dos resultados de testes, por meio de relatório de execução e de defeitos, possibilita fornecer informações importantes para auxiliar as análises dos engenheiros de software na tomada de decisão sobre o andamento do projeto. O relatório de teste facilita também a rastreabilidade dos testes por meio de *logs*, *screenshots* e vídeos, por exemplo.

As soluções para o gerenciamento do processo de teste em tempo real, no ambiente de computação em nuvem, possibilita reduzir o *time-to-market*, facilita o trabalho em equipe, além de impactar diretamente na qualidade do produto de software por meio do rastreamento, acompanhamento e monitoramento de defeitos, contribuindo principalmente para a tomada de decisão pelos engenheiros de software e gerentes de controle de qualidade em relação aos resultados da execução dos testes. Além disso, a abordagem de gerenciamento de teste na nuvem é eficiente, pois permite economizar tempo em relação à instalação de diferentes ferramentas e configuração do ambiente de teste, em comparação com a abordagem tradicional de teste (GAO; BAI; TSAI, 2011).

A automação do gerenciamento dos testes foi utilizada neste trabalho para a extração de dados dos resultados de teste em forma de relatórios com informações da execução dos testes pelos TaaS, tais como:

1. Recuperar o *status* dos testes (passou, falhou);
2. Recuperar a lista de testes executados;
3. Recuperar o *log* do Selenium;
4. Recuperar o tempo de execução dos testes;
5. Recuperar as informações tais como projetos, *jobs*, *builds*, *sessions* etc.;
6. Recuperar as informações das contas de usuários;
7. Recuperar *screenshots*;
8. Recuperar vídeos, entre outras informações importantes.

A Figura 23 mostra um relatório detalhado para o gerenciamento dos resultados de teste no ambiente de computação em nuvem com todas as informações em comum entre os fornecedores de TaaS avaliados no escopo deste trabalho. O relatório gerado foi exatamente o mesmo tanto para a abordagem Multi-TaaS quanto pela abordagem convencional. Conforme será descrito melhor na Subseção 3.6, apenas por motivos de comparações experimentais, optou-se pela geração do mesmo relatório entre os diferentes TaaS. É importante ressaltar também que a abordagem Multi-TaaS não se limita apenas as funcionalidades em comum e pode ser expandida para geração de relatórios distintos, uns com informações mais completas que outros.

A biblioteca Multi-TaaS auxilia os engenheiros de software a revelar os defeitos a partir da validação tanto do *script* de teste quanto do próprio sistema em desenvolvimento, por meio dos artefatos de teste gerados: o relatório de execução e de defeitos gerado com informações do *status* do testes, tempo de execução, mensagens de *log*, *screenshots*, vídeos, entre outras informações, conforme ilustrado na Figura 23.

Figura 23 – Sumarização dos Resultados da execução dos Testes na Nuvem

Resultados da Execução TaaS	
Parâmetro	Valor
Browser	firefox
Browser_version	38.0.1.
Build	
Duration	969 seconds
Log_url	Visualizar
Name	Teste Firefox - Windows XP
Os	Linux
Status	complete
Video_url	Visualizar

Screenshots	
Show <input type="text" value="10"/> entries	Search: <input type="text"/>
Nome da Imagem	
screenshot-000screenshot.png	Visualizar
screenshot-0001screenshot.png	Visualizar
screenshot-0002screenshot.png	Visualizar
screenshot-0003screenshot.png	Visualizar
Showing 1 to 10 of 26 entries	
Previous 1 2 3 Next	

Fonte: Autoria própria

A seguir é detalhada a terceira e última parte da arquitetura de teste Multi-TaaS, referente ao funcionamento do mecanismo de gerenciamento, coleta de informações e sumarização dos resultados dos testes, os quais são mostrados na Figura 24 e descritos com maiores detalhes na listagem a seguir.

- **TesteRelatorio** é o mecanismo responsável por acionar toda arquitetura para a geração do relatório;
- **AutomacaoService** é uma classe abstrata com métodos utilitários para a geração do relatório;
- **AutomacaoServiceSingleton** é a classe que implementa o padrão de projeto Singleton, responsável por utilizar a Factory AutomacaoServiceFactory e garantir que exista uma única instância de AutomacaoService por meio do mecanismo de polimorfismo, como por exemplo, SaucelabsService, BrowserstackService ou TestingbotService;
- **AutomacaoServiceFactory** é a classe responsável por criar e retornar uma instância do serviço (ou cliente REST) devidamente configurado de um fornecedor específico e em tempo de execução para a classe AutomacaoServiceSingleton;

- **Vendor** é a classe que representa o Enum Java que faz parte do padrão de projeto Strategy e implementa os métodos `getWebDriver()` e `getAutomacaoService()`, responsáveis respectivamente por construir o WebDriver e toda a parte de integração REST com os diferentes fornecedores de TaaS no processo de teste na nuvem;
- **SaucelabsService**, **BrowserstackService** e **TestingbotService** são as classes que consomem as API REST proprietárias de cada um dos fornecedores de TaaS. Além disso, cada uma dessas classes são responsáveis por obter os resultados da última execução dos testes automatizados e obter todos os parâmetros necessários para a geração do relatório final com os resultados dos testes;
- **Rest** é a classe que oferece todo o suporte necessário para a construção de um cliente de web service RESTful por meio da implementação de referência Jersey² da API JAX-RS do Java. Essa classe é responsável também por realizar todo o *parse*, ou seja, todo o processo de codificação e decodificação dos dados necessários para obtenção dos resultados dos testes retornados no formato JSON pelos fornecedores de TaaS.

A classe abstrata `AutomacaoService` é responsável por abstrair os diferentes fornecedores de TaaS, garantindo que os serviços `SaucelabsService`, `BrowserstackService` e `TestingbotService` estendam-na e implementem o método `getParametrosParaGeracaoArquivo()` que retorna um mapa de `<String, String>` com todos os parâmetros necessários para a geração do relatório final de teste. Além disso, a classe `AutomacaoService` é flexível e permite a adição de novos fornecedores. Para adicionar um novo fornecedor basta estender a classe `AutomacaoService` e implementar o método `getParametrosParaGeracaoArquivo()` ou implementar apenas os métodos específicos, como por exemplo, `getVideoURL()` para recuperar a *url* do vídeo do teste, caso o engenheiro de software não necessite gerar um arquivo. Por último, a classe `AutomacaoService` possui também métodos de apoio para à formatação do relatório de teste em HTML.

Os Java beans foram omitidos apenas por questão de simplificação dos diagramas UML, pois são classes básicas que representam as entidades dos serviços REST e que contém apenas os atributos privados e os seus respectivos métodos `getters` e `setters`. Essas classes servem também para encapsular, abstrair e transportar todas as informações de integração entre a aplicação cliente e o fornecedor da abordagem Multi-TaaS.

O mecanismo de `TesteRelatorio` foi criado para acionar a geração do relatório da biblioteca Multi-TaaS, mas poderiam ser utilizadas outras abordagens além de um teste do JUnit, por exemplo, uma classe simples com um método `main()`. O JUnit foi utilizado apenas por facilitar o entendimento e a geração de relatórios pelos participantes do experimento, uma vez que o treinamento oferecido no experimento foi totalmente voltado para o contexto de teste de software na computação em nuvem.

² Jersey: <http://jersey.java.net/>

Figura 24 – Diagrama de classes dos Padrões de Projeto *Abstract Factory* e *Strategy*

Fonte: Autoria própria

A abordagem Multi-TaaS é acionada a partir da execução do teste pelo JUnit. Quando o engenheiro de software executa o teste `geraRelatorio()` da classe `TesteRelatorio`, se a configuração estiver definida para o fornecedor Saucelabs, é criada uma instância da classe `SaucelabsService` a partir do endereço web do Saucelabs presente no construtor da classe e é feita a leitura do nome de usuário e o *token* de acesso do arquivo de configuração `taas.config` a partir da classe `PropriedadeUtils`, conforme mostrado no exemplo de Código 10 do Apêndice

F.1.

A classe `PropriedadeUtils` é responsável por ler e carregar as credenciais do nome de usuário e o *token* do arquivo de configuração a partir das chaves “`taas_usuario`” e “`taas_token`” para os respectivos atributos da classe do serviço, como por exemplo `SaucelabsService`, permitindo assim a autorização do acesso aos resultados dos testes por meio da API REST proprietária do fornecedor de TaaS.

A classe `SaucelabsService` faz uso de uma instância da classe `Rest` que é responsável pela criação das configurações básicas das chamadas REST em HTTPS (conexão segura) para obter algum recurso do fornecedor TaaS e receber uma resposta no formato JSON, sobre como recuperar o *status* do teste automatizado por meio do método HTTP GET.

Antes de gerar o relatório final dos resultados dos testes, é invocado o método estático `geraArquivoHTML()` da classe `ReportUtils` passando por parâmetro o caminho do *path* ou diretório no qual o arquivo final será salvo e os demais parâmetros necessários. Os demais parâmetros são obtidos a partir da chamada do método `getParametrosParaGeracaoArquivo()` por meio da instância do `SaucelabsService`. O método `getParametrosParaGeracaoArquivo()`, por sua vez, realiza a chamada do método `carregarUltimaExecucao()` responsável por realizar as chamadas REST para o Saucelabs e coletar todas as informações referentes aos resultados da última execução dos testes automatizados: obter as informações do navegador, sistema operacional, dispositivos, *screenshots* e links dos vídeos gravados a partir dos métodos *getters* e *setters* dos Java beans do Saucelabs preenchendo o mapa de resultados de `<String, String>`, conforme mostrado no exemplo de Código 7 do Apêndice E.1.

Todos os resultados dos testes serão extraídos a partir do respectivo Java Bean do Saucelabs para um mapa de `<String, String>`, por meio do método `getParametrosParaGeracaoArquivo()`. Esse mapa contém todos os parâmetros necessários para a geração do relatório final. Todas as informações do mapa são repassadas para o método estático `geraArquivoHTML()` da classe `ReportUtils` por parâmetro, finalizando, assim, a geração do relatório final dos resultados dos testes automatizados no formato HTML.

3.6 Vantagens e Desvantagens da Biblioteca Multi-TaaS

Nesta subseção são apresentados os *trade-offs* com as principais vantagens e desvantagens em decorrência da utilização da biblioteca Multi-TaaS.

Vantagens da Biblioteca Multi-TaaS:

1. **Flexibilidade e extensibilidade:** a biblioteca Multi-TaaS é flexível e extensível, pois permite a adição de novos fornecedores TaaS na escrita, configuração, execução e gerenciamento dos resultados dos testes automatizados na nuvem;

2. **Reusabilidade:** a biblioteca Multi-TaaS permite aumentar a reutilização do código de teste automatizado do JUnit, na medida em que possibilita desacoplá-lo da plataforma TaaS da qual ele será executado. A reusabilidade ocorre também por meio das abstrações dos aspectos de comunicação e integração REST com os distintos fornecedores TaaS no gerenciamento dos resultados dos testes automatizados;
3. **Legibilidade:** o padrão de escrita dos testes da biblioteca Multi-TaaS facilita a leitura e o entendimento do código de teste. A ideia principal da convenção foi automatizar os testes de modo que eles pudessem ser utilizados em forma de especificação executável e que pudesse ser dispensado o uso da documentação, ou seja, o engenheiro de software ao ler o teste seria equivalente a ler a respectiva documentação do teste, melhorando consequentemente a legibilidade e a manutenibilidade do código de teste (HUMBLE; FARLEY, 2010);
4. **Redução do esforço de migração:** a biblioteca Multi-TaaS reduz também a complexidade da migração para o engenheiro de software ao facilitar o processo de migração e, ao mesmo tempo, aumenta a portabilidade e reduz o esforço de migração necessário para a troca de fornecedores TaaS, pois exige apenas a modificação de um arquivo de configuração (GROZEV; BUYYA, 2014a; OLIVEIRA; MARTINS; SIMAO, 2017);
5. **Maior Eficiência:** a biblioteca Multi-TaaS possibilita reduzir também o tempo total do processo de migração de uma plataforma TaaS para outra, referente aos testes automatizados UAT e do software que faz a integração REST e sumariza os resultados dos testes, aumentando, consequentemente, a eficiência no processo de migração;
6. **Portabilidade:** a biblioteca Multi-TaaS favorece a portabilidade do código de teste entre as diferentes plataformas de TaaS devido à camada de abstração criada para facilitar o processo de migração, independentemente dos testes automatizados e das tecnologias dos fornecedores de TaaS. Além disso, existe o ganho de portabilidade em relação às aplicações de integrações REST que podem ser migradas entre diferentes plataformas de TaaS de forma transparente para os consumidores;
7. **Redução do custo de manutenção:** a biblioteca Multi-TaaS torna o processo de escrita e execução dos testes mais prático, eficiente e seguro, uma vez que é necessário realizar apenas o adequado gerenciamento de configurações. O desacoplamento do *script* da execução do teste por meio dos padrões de projeto e do arquivo de configuração, possibilita realizar as manutenções das configurações das capacidades sem impactar no restante do código de teste, pois não há necessidade de alterar o código-fonte do teste automatizado, reduzindo também a possibilidade de inserção de defeitos devido à manutenção do sistema. A redução dos custos de manutenção ocorrem na medida em que possibilitam a redução do esforço total de migração, pois é necessário realizar apenas o gerenciamento de configurações,

consequentemente melhorando a qualidade do software de uma forma geral (OLIVEIRA; MARTINS; SIMAO, 2017);

8. **Facilita o gerenciamento dos testes:** a biblioteca Multi-TaaS facilita a coleta de informações, gerenciamento da execução e a sumarização dos resultados dos testes por meio de um relatório de execução e defeitos.

Desvantagens da biblioteca Multi-TaaS:

1. **Padronização parcial das capacidades:** apesar da biblioteca Multi-TaaS desacoplar a escrita do teste do fornecedor TaaS em que o teste será executado, por meio de um arquivo do tipo propriedades chave valor, a padronização das capacidades é parcial, pois padroniza o lado da chave e não do valor do arquivo de configuração.

Suponha-se que o engenheiro de software deseje executar o mesmo teste automatizado nas plataformas do Saucelabs e do Browserstack com as seguintes configurações: Internet Explorer, Windows 8 e resolução de tela “1024x768”. As configurações são mostradas no Código 1.

Código-fonte 1 – Exemplo de capacidades do Saucelabs e Browserstack

```
1
2  //Exemplo de capacidades do Saucelabs
3  caps.setCapability("browserName", "Internet Explorer"
4  );
5  caps.setCapability("version", "10.0");
6  caps.setCapability("platform", "Windows 8");
7  caps.setCapability("screenResolution", "1024x768");
8
9  //Exemplo de capacidades do Browserstack
10 caps.setCapability("browser", "IE");
11 caps.setCapability("browser_version", "10.0");
12 caps.setCapability("os", "Windows");
13 caps.setCapability("os_version", "8");
14 caps.setCapability("resolution", "1024x768");
15
```

No exemplo de código das capacidades mostrado acima, percebe-se que a chave correspondente ao nome do navegador no Saucelabs é “browserName” enquanto que no Browserstack a mesma chave é representada apenas por “browser”, nesse caso a biblioteca Multi-TaaS padroniza tudo para “taas_nome_browser”. Porém o valor pode variar

de "Internet Explorer" para o Saucelabs e "IE" para o Browserstack e nesse caso a biblioteca Multi-TaaS não padroniza esses valores.

No entanto um dos maiores problemas de padronização da nomenclatura das capacidades ocorre quando o Saucelabs estabelece que a chave "platform" possui 2 informações integradas, sendo elas, o nome do sistema operacional juntamente com a sua versão, como por exemplo, "Windows 8". Já o Browserstack disponibiliza as informações do sistema operacional de forma separada da sua versão, como por exemplo, as capacidades "os" e "os_version". Nesse caso, a biblioteca Multi-TaaS também padroniza o sistema operacional para "taas_sistema_operacional" e a sua versão para "taas_versao_sistema_operacional", mas da mesma forma como mostrado no exemplo anterior do navegador, tanto o valor do sistema operacional quanto a sua versão não são padronizados pela biblioteca Multi-TaaS.

Finalmente, é possível perceber também, pelo exemplo do Código fonte 1, que a capacidade referente à resolução de tela possui o mesmo valor, porém com chaves diferentes: no Saucelabs a chave correspondente é "screenResolution" e no Browserstack a chave é apenas "resolution". Nesse caso, a biblioteca Multi-TaaS padroniza as duas chaves para "taas_resolucao_tela".

Uma solução para esse problema seria criar um mapa de chaves/valores intermediário como parte da biblioteca Multi-TaaS, que então converteria para as chaves e valores correspondentes do TaaS selecionado. Dessa forma o teste e o arquivo de configuração seriam completamente desacoplados dos fornecedores TaaS. Exemplo de padronização:

"taas_nome_browser": internet explorer (*Case insensitive*)

"taas_nome_browser": firefox (*Case insensitive*)

"taas_nome_browser": chrome (*Case insensitive*)

2. **Padronização de funcionalidades em comum:** por motivos de decisão de projeto, criou-se um relatório com a interseção das funcionalidades, justamente para poder comparar a geração do mesmo relatório entre os diferentes TaaS. Mas seria possível gerar relatórios distintos, uns mais completos que outros, dependendo do TaaS selecionado. Visto que, como foi descrito anteriormente. O Saucelabs oferece muito mais informações e funcionalidades que outros TaaS. Uma outra solução possibilitaria que a própria biblioteca Multi-TaaS alertasse o engenheiro de software que a funcionalidade não está disponível para o TaaS selecionado em tempo de execução;
3. **Risco da transferência de Vendor Lock-in:** existe o risco de transferir o *vendor lock-in* do TaaS para a biblioteca Java Multi-TaaS, mas cabe ressaltar que o objetivo principal desta tese de doutorado é disponibilizar a biblioteca para toda a comunidade *open source* e dar o primeiro passo para um padrão ou protocolo aberto para manipular e operar diferentes TaaS. Essa seria a solução ideal, porque dessa forma, seria possível automatizar

as tarefas de migração e realizar a troca de fornecedores em tempo de execução e de forma transparente para os consumidores de TaaS.

O risco de transferir o *vendor lock-in* para a biblioteca Multi-TaaS existe, conforme dito anteriormente, porém é relativo e torna-se importante fazer um comparativo com o próprio Selenium Webdriver. Se o engenheiro de software utiliza o Selenium no seu código Java, ele consequentemente estará preso ao Selenium, porém não ficará preso a nenhum navegador específico. Os navegadores possuem bibliotecas para manipulá-los que implementam o protocolo do Selenium Webdriver. Da mesma forma, o engenheiro de software ao utilizar a biblioteca Multi-TaaS estará preso a ela, porém não estará preso a nenhum fornecedor de TaaS específico.

O padrão de escrita de teste possui a principal desvantagem, pois exige que o engenheiro de software aprenda a convenção, porém os principais benefícios de um padrão de escrita de testes automatizados está na melhora da legibilidade e consequentemente da manutenibilidade do código de teste, o que facilita a manutenção do código de teste por outros membros da equipe.

Portanto somente um protocolo ou padrão aberto, utilizado tanto pelos consumidores quanto por fornecedores TaaS (para comunicação e colaboração intra TaaS) resolveria completamente o problema de *vendor lock-in*. Assim, tanto os fornecedores quanto os consumidores de TaaS estariam sempre se comunicando diretamente uns com os outros e usando o mesmo protocolo/idioma, sem a necessidade de uma solução intermediária.

3.7 Procedimentos para utilização da biblioteca Multi-TaaS

Nesta seção são descritos todos os procedimentos necessários para testar outras aplicações web com a utilização da biblioteca Multi-TaaS. O passo a passo a seguir ilustra de forma simplificada como importar e usar a biblioteca desenvolvida.

As extensões do JUnit já são utilizadas na indústria de software, mas não para solucionar o problema de *lock-in* no TaaS. As diretrizes descritas a seguir são voltadas para parametrizações e padrões de escrita de teste.

1. Importar o arquivo .jar correspondente à biblioteca Multi-TaaS;
2. Seguir a convenção para manter a legibilidade do código de teste, conforme ilustrado também no exemplo de Código 5 do Apêndice E.1.
 - 2.1. Anotar a classe de teste com o `@RunWith(PluginRunner.class);`
 - 2.2. Estender todas as classes de teste de `AbstractCenario;`

- 2.3. Utilizar a anotação `@Carga` no teste, como por exemplo, `@Carga(classe = {UniversidadeCarga.class, DepartamentoCarga.class})`, caso exista injeção de dados em banco (Opcional);
 - 2.4. Estender os `PageObject` de `AbstractPageObject`;
 - 2.5. Estender as classes de Carga do sistema de `AbstractCarga`;
 - 2.6. Usar a classe `ResultadoVerificavel` para manter o padrão de escrita (Opcional).
3. Criar os arquivos de configuração a seguir na raiz do projeto:
 - 3.1. `startup.config`: arquivo de configuração necessário para a inicialização do navegador e acesso à *url* principal da aplicação web a ser testada a partir da arquitetura Multi-TaaS, conforme o exemplo de Código 9 do Apêndice F.1;
 - 3.2. `taas.config`: arquivo de configuração responsável por armazenar todas as configurações das capacidades e demais informações necessárias para a migração do serviço de teste na nuvem. Esse arquivo também é utilizado para a geração do relatório dos resultados de teste automatizados a partir do *token* de autorização e acesso ao TaaS, conforme mostrado nos exemplos de Códigos 10, 11 e 12 do Apêndice F.1;
 - 3.3. `bd.config`: arquivo de configuração responsável por armazenar todas as informações necessárias para a conexão com o banco de dados da aplicação web, conforme ilustrado no exemplo de Código 13 do Apêndice F.1.

3.8 Implementação do Protótipo e Estudo de Caso

Nesta seção, é apresentado um estudo de caso com a biblioteca Multi-TaaS, e será apresentado também como a arquitetura desenvolvida permite auxiliar os engenheiros de software na tomada de decisão em relação à troca de fornecedores do Saucelabs para o Browserstack ou outras plataformas de TaaS, como por exemplo, o Testingbot.

Um protótipo foi desenvolvido como prova de conceito, constituído por um processo de seis passos, conforme a listagem a seguir, com o objetivo principal de comprovar a sua viabilidade e aplicabilidade no mundo real.

1. Implantação das aplicações servidor e cliente RESTful do Sistema de Informação de Ambiente Universitário (SInAU) referente ao domínio de sistema de gerenciamento educacional na plataforma de computação em nuvem do Heroku³;
2. Criação de um conjunto de *User Acceptance Tests* (UAT) C1 para a aplicação SInAU com a ferramenta Selenium WebDriver;

³ Heroku:
<<https://dashboard.heroku.com/>>

3. Execução local do conjunto de testes *C1*, por meio do *framework* JUnit;
4. Execução do conjunto de testes *C1* remotamente na plataforma TaaS do Sauce Labs, Browserstack e Testingbot;
5. Utilização dos padrões de projeto, descritos na Seção 2.3, para desacoplar a escrita das capacidades em relação à execução dos testes automatizados por meio do arquivo de configuração;
6. Criação de uma camada de abstração na arquitetura Multi-TaaS para desacoplar o *script* de teste automatizado UAT de qual TaaS ele será executado e consumir os serviços RESTful dos três fornecedores de TaaS para a sumarização dos resultados dos testes automatizados;
7. Replicação da aplicação SInAU no Heroku para cada participante do experimento controlado, pois a aplicação web não possuía as propriedades de uma aplicação *multi-tenancy*.

Após a construção do protótipo, foi desenvolvida uma arquitetura de teste inicial com o objetivo de reduzir os impactos ocasionados pelo problema de *vendor lock-in* de menor e maior grau. Conforme descrito anteriormente, a biblioteca Multi-TaaS utilizou a combinação de padrões de projeto com os arquivos de configuração para abstrair e facilitar a troca de fornecedores de TaaS, favorecendo, principalmente, a portabilidade dos testes automatizados UAT entre os diferentes TaaS e das aplicações responsáveis pela geração do relatório de execução e defeitos de cada fornecedor TaaS.

A escrita dos testes automatizados ocorreu por meio do Selenium Webdriver e do *framework* JUnit. Os testes automatizados utilizados em ambas as abordagens Multi-TaaS e Convencional foram exatamente os mesmos (*Create, Read, Update and Delete* (CRUD) de Universidade), visto que, antes do início de cada teste, foi realizada a inserção dos dados de teste diretamente no banco de dados. Após a execução dos testes, os dados foram removidos garantindo a independência entre os testes e mantendo a consistência do banco de dados. Além disso, os casos de testes e a sequência de comandos também foram equivalentes em ambas as abordagens desenvolvidas.

O estudo de caso foi implementado seguindo todos os procedimentos descritos na Seção 3.7. Após o desenvolvimento do protótipo, foi necessário dividi-los em subprojetos menores para avaliação experimental. Sendo assim, a divisão das abordagens Multi-TaaS e a abordagem convencional resultaram em 7 projetos.

Conforme ilustrado na Figura 25, os projetos GrupoA-1-Dia e GrupoB-1-Dia correspondem aos projetos utilizados no primeiro dia de curso. Da mesma forma os projetos GrupoA-2-Dia e GrupoB-2-Dia também foram utilizados para o segundo dia de curso do experimento. A divisão dos projetos foi realizada com base na divisão dos grupos do experimento controlado. Já os projetos GrupoAUtils e GrupoBUtils correspondem aos projetos das bibliotecas Java de carga e

descarga dos dados de testes da aplicação web SInAU. Assim, a modularização dos projetos teve como principal objetivo facilitar o entendimento do código de teste para todos os participantes do experimento. Por último, o projeto `ArquiteturaWebDriverPTaaS` corresponde a implementação da biblioteca Multi-TaaS, que foi importada apenas nos projetos dos participantes que utilizaram a abordagem proposta. Todos os projetos com a letra B no nome, correspondem aos projetos da abordagem convencional e os projetos com a letra A no nome, correspondem aos projetos da abordagem Multi-TaaS.

Figura 25 – Projetos desenvolvidos para o experimento controlado

Fonte: Autoria própria

A seguir, conforme apresentado na Figura 26, são listadas e descritas as cinco classes principais desenvolvidas para o estudo de caso da automatização de testes específicos voltado para a aplicação web SInAU.

- **UniversidadeSystemTest** é a classe de testes automatizados da aplicação SinAU (SUT) referentes ao CRUD (*Create, Update, Read e Delete*) da universidade, conforme o Código 5 disponível no Apêndice E.1;
- **IncluirUniversidadePage**, **EditarUniversidadePage** e **ListarUniversidadePage** são as classes que implementam o padrão de projeto `PageObject` responsáveis por encapsular todos os atributos, métodos, campos e ações das páginas web de cadastro, editar e listar universidades respectivamente. Posteriormente, serão utilizadas pelo teste automatizado `UniversidadeSystemTest`;
- **UniversidadeCarga** é uma especialização da classe `AbstractCarga` responsável por implementar os métodos de carga e descarga dos dados de teste da Universidade por meio de comandos SQL;
- **AplicacaoPage** é uma extensão da classe abstrata `AbstractPageObject` que permite acessar a *url* principal e navegar no menu da aplicação web;

Após instanciar o `PageObject` `AplicacaoPage` por meio do método `UsuarioEm()`, o teste navegará pelo menu da aplicação para encontrar e acessar o *link* de cadastro de uma

Figura 26 – Diagrama de classe do Estudo de Caso utilizando a biblioteca Multi-TaaS

Fonte: Autoria própria

nova universidade por meio do valor “Universidades>Nova Universidade”, definido no próprio teste automatizado. Do mesmo modo, o PageObject `IncluirUniversidadePage` fica responsável pela localização dos elementos da página de cadastro, receber os dados de entrada do teste, preencher os campos do formulário de cadastro da página web e acionar o botão *submit* para a submissão do formulário pela aplicação `SInAU`, ou seja, todas as ações do usuário final são simuladas pelo fornecedor TaaS.

Em seguida, ao realizar o cadastro de uma nova Universidade, o teste valida por meio do método `assertEquals()` do JUnit se a Universidade realmente existe e foi cadastrada com sucesso no banco de dados e se as suas informações são exibidas na página de listagem da aplicação `SInAU`. Em seguida, o JUnit retornará o resultado do teste, informando se passou ou falhou. Caso o teste finalize com alguma falha, será acionado o mecanismo que realiza a descarga automática dos dados de teste.

3.9 Considerações Finais

Este capítulo apresentou, inicialmente, uma visão geral sobre os principais conceitos relacionados com a abordagem Multi-TaaS. Em seguida, foi descrito todo o processo de teste na nuvem. Foi apresentado também como o problema de *vendor lock-in* afeta as diferentes etapas do processo de teste na nuvem com base na classificação de maior e menor grau. Foi apresentada no decorrer deste capítulo também uma visão geral do funcionamento da biblioteca Multi-TaaS, bem como as suas vantagens e desvantagens e todos os procedimentos necessários para a sua

utilização em outros domínios de aplicação. Foi apresentado também o protótipo e o estudo de caso desenvolvido como prova de conceito. No próximo capítulo são apresentados o processo da avaliação do experimento controlado conduzido bem como toda a instrumentação e ambiente controlado necessários para a replicação do experimento por outros pesquisadores interessados na área. Finalmente, serão apresentadas também todas as análises estatísticas.

AVALIAÇÕES EXPERIMENTAIS

4.1 Considerações Iniciais

A Engenharia de Software Experimental possibilita obter evidências por meio de experimentos envolvendo softwares. A experimentação é um processo sistemático composto por cinco atividades principais: **definição e planejamento** do experimento, **operação**, **análise e interpretação**, **empacotamento e apresentação** dos resultados (WOHLIN *et al.*, 2000). Além disso, os experimentos controlados possibilitam formar uma base de conhecimento e o compartilhamento das soluções entre os pesquisadores de forma eficiente (BASILI, 1996; BASILI; SELBY; HUTCHENS, 1986). Todas as atividades que compõem o processo experimental são mostradas na Figura 27 a seguir.

Figura 27 – Etapas do processo de experimentação

Fonte: Adaptado de Wohlin *et al.* (2000)

Durante o processo de experimentação, foram realizados quatro pré experimentos com o objetivo principal de verificar as possíveis falhas na condução do experimento e erros nos projetos de cada uma das abordagens antes de executar os experimentos finais. No entanto, não são detalhadas todas as etapas dos pré experimentos porque a quantidade de dados coletados não implicaram em resultados significativos para análise estatística.

Neste capítulo, são apresentados os estudos experimentais realizados para avaliar a qualidade da abordagem Multi-TaaS quanto à efetividade, eficiência, esforço e dificuldade no processo de migração dos testes automatizados UAT e da sumarização dos resultados de teste.

Todas as atividades do processo de experimentação são melhor detalhadas nas próximas seções e subseções deste capítulo. O restante deste Capítulo está organizado da seguinte forma: na Seção 4.2 são descritas todas as etapas do processo de experimentação, todo o planejamento do experimento com os objetivos, hipóteses, seleção dos indivíduos e descrição das tarefas dos experimentos. A Seção 4.3 descreve toda a preparação e configuração necessárias para a execução dos experimentos controlados. Na Seção 4.4 são descritas todas as etapas da condução dos experimentos realizados neste trabalho, bem como as avaliações das hipóteses por meio de testes estatísticos realizados ao longo da pesquisa desenvolvida. Finalmente, as considerações finais serão apresentadas na Seção 4.6.

4.2 Experimentos Controlados

Os métodos empíricos, tais como os experimentos controlados, são cruciais e essenciais para a web e também para a engenharia de software tanto no meio acadêmico quanto na indústria, pois além de incorporar o comportamento humano permitem avaliar e validar os resultados da pesquisa. Este método é necessário para poder comparar se uma abordagem é melhor do que outra. Além disso, os resultados de qualquer experimento controlado permitem que outros pesquisadores possam replicar o estudo, além de servir como referência para novas pesquisas na área (WOHLIN; HÖST; HENNINGSSON, 2006).

Para validar esta pesquisa, foi conduzido um experimento controlado em busca de evidências que comprovassem o aumento da eficiência e a redução da dificuldade e do esforço no processo de migração de fornecedores de TaaS, por meio de testes e modelos estatísticos. Todo o processo de experimentação utilizado nesta pesquisa foi extraído de Wohlin *et al.* (2000) e Juristo e Moreno (2010).

A seguir, todas as fases do processo de experimentação e as suas respectivas atividades serão descritas nas próximas subseções.

4.2.1 Definição e Planejamento do Estudo

Na etapa de definição, o experimento é descrito com base nos problemas e objetivos. Assim, são descritas as principais questões de pesquisa que originaram o experimento. Em seguida, são formuladas as hipóteses, a definição e o planejamento referentes ao escopo do experimento (WOHLIN *et al.*, 2000).

Na etapa do planejamento, o projeto do experimento foi melhor detalhado, incluindo também a avaliação dos riscos e ameaças à validade. Foram definidos o contexto, hipóteses, variáveis, participantes, o projeto experimental, a instrumentação e a avaliação de validade do projeto experimental.

4.2.2 Objetivos

O principal objetivo desta pesquisa foi avaliar e comparar a efetividade, eficiência, dificuldade e o esforço de migração, considerando o tamanho do projeto ou linhas de código de teste automatizados, complexidade ciclomática e o tempo gasto nas tarefas de migração entre os fornecedores de TaaS: Saucelabs, Browserstack e Testingbot. A avaliação comparativa foi realizada com o objetivo principal de responder às seguintes questões:

- Como solucionar ou reduzir os impactos ocasionados pelo problema de *vendor lock-in* no processo de teste na nuvem?
- Qual a efetividade e eficiência da abordagem Multi-TaaS em comparação com a abordagem convencional?
- Qual a dificuldade e o esforço de migração necessários para o engenheiro de software migrar ou portar um teste entre diferentes plataformas de fornecedores TaaS?
- Qual a dificuldade e o esforço de migração necessários para o engenheiro de software migrar de serviço de teste por meio das diferentes API REST proprietárias dos fornecedores TaaS?

4.2.3 Metas

Com o propósito de avaliar a qualidade da abordagem Multi-TaaS, foi estabelecida a seguinte meta para os experimentos controlados:

Analisar as abordagens Multi-TaaS e convencional em relação à migração de testes automatizados UAT e da sumarização de resultados de testes por meio da API REST dos fornecedores de TaaS Saucelabs, Browserstack e Testingbot

com o propósito de **avaliar e comparar**

com respeito à **efetividade, eficiência, dificuldade e esforço**

do ponto de vista **do engenheiro de software e empresas que consomem o serviço de teste**

na nuvem

no contexto de teste como serviço na computação em nuvem

4.2.4 Hipóteses

O principal objetivo da análise por meio de teste de hipótese é determinar o quanto as variável(eis) independente(s) são afetadas pelas variável(s) dependente(s). Todas as variáveis são definidas na próxima Subseção 4.2.5 (WOHLIN; HÖST; HENNINGSSON, 2006).

Nesse contexto, foram definidas as seguintes perguntas que pretendem ser respondidas por este trabalho, considerando para isso uma hipótese nula e uma hipótese alternativa:

1. *A abordagem Multi-TaaS é mais eficiente e efetiva em comparação com a abordagem convencional na migração de fornecedores de TaaS?*
2. *A abordagem Multi-TaaS permite reduzir o esforço de migração de fornecedores de TaaS em comparação com a abordagem convencional?*

Para avaliar a efetividade, eficiência e o esforço de migração entre plataformas de TaaS foram definidas as seguintes hipóteses:

Hipóteses 1 - Efetividade

Hipótese Nula(H_0): A efetividade da abordagem Multi-TaaS é menor ou igual à efetividade da abordagem convencional no processo de migração de fornecedores de TaaS.

H_0 : Efetividade (abordagem Multi-TaaS) = Efetividade (abordagem convencional)

Hipótese Alternativa(H_1): A efetividade da abordagem Multi-TaaS é maior do que a efetividade da abordagem convencional no processo de migração de fornecedores de TaaS.

H_1 : Efetividade (abordagem Multi-TaaS) $\neq$ Efetividade (abordagem convencional)

Hipóteses 2 - Eficiência

Hipótese Nula(H_0): A eficiência da abordagem Multi-TaaS é menor ou igual à eficiência da abordagem convencional no processo de migração de fornecedores de TaaS.

H_0 : Eficiência (abordagem Multi-TaaS) $\leq$ Eficiência (abordagem convencional)

Hipótese Alternativa(H_1): A eficiência da abordagem Multi-TaaS é maior do que a eficiência da abordagem convencional no processo de migração de fornecedores de TaaS.

H_1 : Eficiência (abordagem Multi-TaaS) $>$ Eficiência (abordagem convencional)

Hipóteses 3 - Esforço

Hipótese Nula(H_0): O esforço da abordagem Multi-TaaS é maior ou igual ao esforço da abordagem convencional no processo de migração de fornecedores de TaaS.

H_0 : Esforço de migração (abordagem Multi-TaaS) $\geq$ Esforço de migração (abordagem convencional)

Hipótese Alternativa(H_1): O esforço da abordagem Multi-TaaS é menor do que o esforço da abordagem convencional no processo de migração de fornecedores de TaaS.

H_1 : Esforço de migração (abordagem Multi-TaaS) $<$ Esforço de migração (abordagem convencional)

4.2.5 Variáveis do Experimento

Variáveis independentes: Wohlin *et al.* (2000) e Juristo e Moreno (2010) definem variáveis independentes como aquelas que podem ser manipuladas ou controladas no processo de experimentação. As variáveis independentes também são conhecidas como fatores do experimento e o termo tratamento refere-se aos possíveis valores que um fator pode assumir. Foram identificadas neste experimento as seguintes variáveis independentes:

1. Abordagem de interesse (tratamento);
2. Abordagem convencional (controle);
3. Testes UAT automatizados e arquivos de configuração;
4. Linguagem de Programação;
5. Sistema Web utilizado para teste;
6. Ambiente com todas configurações e ferramentas utilizadas no experimento;
7. Informações sobre os sujeitos – experiência dos participantes e treinamento;
8. Tarefas de migração.

A abordagem convencional também é chamada de controle, ao passo que a abordagem Multi-TaaS é chamada de tratamento.

Variáveis dependentes: aquelas nas quais são observados os resultados da manipulação das variáveis independentes, ou seja, as variáveis que expressam o efeito na relação causa-efeito definida no experimento (WOHLIN *et al.*, 2000; JURISTO; MORENO, 2010). As variáveis dependentes identificadas neste estudo são:

1. A efetividade, dificuldade e eficiência da abordagem Multi-TaaS comparada com a abordagem convencional;
2. O esforço de migração entre fornecedores de TaaS de ambas as abordagens.

A forma como as variáveis dependentes são mensuradas será melhor descrita nas Subseções 4.5.2 e 4.5.3.

4.2.6 Seleção dos Indivíduos - IFSULDEMINAS

Um curso sobre Teste como Serviço (TaaS) na Computação em Nuvem foi oferecido para todos os alunos do 2º e 4º período do curso de Engenharia de Computação do Instituto Federal de Educação, Ciência e Tecnologia do Sul de Minas - IFSULDEMINAS, campus Poços de Caldas.

Para participar do curso foram estabelecidos o critério de conhecimentos básicos de Java para Web e arquitetura Cliente-Servidor. Foram selecionados 8 alunos do 2º período e mais 6 alunos do 4º período, totalizando 14 participantes. Cada participante realizou 2 tarefas, gerando assim, 28 amostras. Todos os 8 sujeitos do 2º período, bem como os 6 participantes do 4º período foram divididos aleatoriamente em dois grupos: grupo A (Multi-TaaS) e grupo B (Convencional), dentro dos seus respectivos períodos.

Informações do experimento com os participantes do 2º Período:

Datas:

Terça-feira dia 29/11/2016

Quarta-feira dia 30/11/2016

Horário: 13:00hs às 17:00hs

Local: Laboratório do IFSULDEMINAS – Lab 2

Participantes: Alunos do 2º período do IFSULDEMINAS

Informações do experimento com os participantes do 4º Período:

Datas:

Terça-feira dia 06/12/2016

Quarta-feira dia 07/12/2016

Horários: Terça-feira das 16:00hs às 20:00hs e Quarta-feira das 13:00hs às 17:00hs

Local: Laboratório do IFSULDEMINAS – Lab 2

Participantes: Alunos do 4º período do IFSULDEMINAS

4.2.7 Seleção dos Indivíduos - UFSCar

Um curso sobre Teste como Serviço (TaaS) na Computação em Nuvem foi oferecido a todos os alunos da disciplina de Engenharia de Software da Universidade Federal de São Carlos (UFSCar) campus de São Carlos. Os participantes correspondem aos alunos que estão no 5º e 7º

período dos cursos de Ciência da Computação e da Engenharia de Computação, respectivamente. Além disso, a disciplina é oferecida no segundo semestre de cada ano e costuma ser dividida em duas turmas A e B, devido à grande quantidade de alunos matriculados na UFSCar. É importante destacar que em cada uma das duas turmas foi realizada a divisão dos grupos em A (Multi-TaaS) e B (Convencional) de forma independente.

Para participar do curso foram estabelecidos os critérios de conhecimentos básicos de Java para Web e arquitetura Cliente-Servidor. Foram selecionados 27 alunos da turma A e mais 35 alunos da turma B, totalizando 62 participantes no 1º experimento. Devido ao número de participantes que participaram do 1º experimento e não participaram do 2º experimento, houve 17 alunos da turma A e 23 da turma B, totalizando assim, 40 participantes no 2º experimento.

É importante destacar que dos 62 participantes do 1º experimento, apenas 48 entregaram o código fonte para as análises do esforço, totalizando assim, 14 participantes do 1º experimento que preencheram o formulário e concluíram as tarefas, porém não entregaram o código fonte.

Um total de 40 participantes preencheram o formulário e concluíram as tarefas do 2º experimento, no entanto 5 participantes não entregaram o código. Dessa forma, apenas 35 participantes foram utilizados para as análises estatísticas do esforço.

Devido a todos os acontecimentos explicados anteriormente, a Tabela 4 mostra o número de participantes com as suas respectivas quantidades de amostras consideradas para as análises estatísticas da efetividade, eficiência, dificuldades e esforço de migração.

Tabela 4 – Número de participantes e amostras - Experimento UFSCar

Avaliação	Experimento	Grupo	Nº de Participantes	Nº de Amostras
Eficiência	1º	A	23	46
		B	39	78
	2º	A	16	32
		B	24	48
Efetividade	1º	A	23	46
		B	39	78
	2º	A	16	32
		B	24	48
Esforço	1º	A	18	18
		B	30	36
	2º	A	14	28
		B	21	42
Dificuldade	1º	A	23	46
		B	38	76
	2º	A	16	32
		B	23	46

É importante destacar que após realizar as análises estatísticas do esforço para os experimentos do IFSULDEMINAS, percebeu-se que as duas tarefas do primeiro experimento eram muito semelhantes e acreditava-se que a coleta de dados de apenas uma tarefa já seria o suficiente para a análise estatística geral dos resultados do experimento da UFSCar. Sendo assim, não foi exigida a entrega do código fonte da 1ª Tarefa do 1º Experimento na UFSCar e percebeu-se que esses dados não disponíveis poderiam explicar melhor o comportamento da amostra. Porém

só após as análises dos dados dos experimentos da UFSCar, percebeu-se a real necessidade de realizar as análises estatísticas dos resultados independente das tarefas.

Como não foi exigida a entrega do código fonte da 1ª Tarefa do 1º Experimento na UFSCar, apenas 6 participantes entregaram os códigos das duas tarefas do 1º experimento e 2 participantes entregaram apenas o código da primeira tarefa do 1º experimento. Os restantes 40 participantes entregaram apenas o código da 2ª tarefa do 1º experimento, totalizando 48 participantes que preencheram o formulário e entregaram o código do 1º experimento para a análise do esforço de migração.

As informações detalhadas de datas e horários dos experimentos conduzidos em cada uma das duas turmas são descritas a seguir:

Informações do experimento com os participantes da turma A:

Datas:

Terça-feira dia 25/04/2017

Terça-feira dia 02/05/2017

Horário: 08:00hs às 12:00hs

Local: Laboratório da Secretaria Geral de Informática (SIn)

Participantes: Alunos da turma A de Engenharia de Software da UFSCar

Informações do experimento com os participantes da turma B:

Datas:

Sexta-feira dia 28/04/2017

Sexta-feira dia 05/05/2017

Horários: 14:00hs às 18:00hs

Local: Laboratório da Secretaria Geral de Informática (SIn)

Participantes: Alunos da turma B de Engenharia de Software da UFSCar

4.2.8 Tarefas Experimentais do 1º Dia de Curso

Conforme descrito nas subseções anteriores, os experimentos realizados tanto no IFSUL-DEMINAS quanto na UFSCar foram oferecidos em forma de curso com duração total de 8 horas, divididas em 2 dias, com 4 horas de curso em cada dia. O treinamento no primeiro dia foi sobre configuração de capacidades, escrita e execução de testes automatizados nos principais fornecedores de TaaS do mercado, sendo eles: Saucelabs¹, Browserstack² e Testingbot³, conforme ilustrado na Figura 56 do Apêndice B.1. Foi solicitado inicialmente que cada participante criasse a sua própria conta de teste (*trial*) gratuita nos três fornecedores de TaaS para a execução dos testes automatizados na nuvem. Essa atividade ocorreu antes de ser iniciada a tarefa de migração correspondente ao fornecedor específico.

¹ Saucelabs: <<https://saucelabs.com/>>

² Browserstack: <<https://www.browserstack.com/>>

³ TestingBot: <<https://testingbot.com/>>

Todos os indivíduos dos dois grupos receberam os projetos equivalentes, com as mesmas configurações de capacidades e um conjunto de testes automatizados definidos de acordo com as suas respectivas abordagens.

O projeto entregue para os participantes da abordagem convencional no primeiro experimento foi composta por 318 linhas de código (LOC) e complexidade ciclomática de 1,02. Todos esses valores ilustram o grau de complexidade dos testes automatizados e das configurações das capacidades antes da execução das tarefas pelos participantes. O projeto entregue para os participantes da abordagem Multi-TaaS no primeiro experimento foi composta por 336 linhas de código (LOC) e complexidade ciclomática de 1,15. Todos esses valores ilustram o grau de complexidade dos testes automatizados e das configurações das capacidades antes da execução das tarefas pelos participantes.

As tarefas de migração do 1º experimento são listadas a seguir e foram realizadas na mesma ordem, ou seja, da tarefa de número 1 a 4 e executadas, uma única vez, em cada um dos dois grupos A e B. Após o processo de migração, os participantes tinham que reexecutar todos os testes automatizados no novo fornecedor. As tarefas eram de natureza muito semelhante para ambos os grupos. Ao final da primeira tarefa, todos os participantes preencheram um formulário com o tempo de início e o tempo final, antes de iniciar a execução da próxima tarefa.

A primeira e a segunda tarefas possuíam apenas caráter didático e não foram coletados quaisquer tipos de dados. A seguir, é apresentada uma descrição de cada uma das tarefas realizadas durante a condução do experimento com base nas configurações das capacidades ilustradas na Tabela 5:

Tarefa 1 - Execução dos testes automatizados no computador local com o objetivo de introduzir e familiarizar os alunos com a API do Selenium Webdriver⁴ e do *framework* JUnit⁵;

Tarefa 2 - Cada um dos dois grupos recebeu um exemplo pronto como referência, de acordo com a sua respectiva abordagem e, em seguida, executaram os testes automatizados na plataforma do Saucelabs, com o objetivo de introduzir os alunos e familiarizá-los com os testes no ambiente de nuvem;

Tarefa 3 - O objetivo dessa tarefa consistia em cada grupo realizar a migração do fornecedor Saucelabs para o fornecedor de TaaS Browserstack na sua respectiva abordagem e, em seguida, executar os testes na plataforma do Browserstack;

Tarefa 4 - O objetivo dessa tarefa consistia em cada grupo realizar a migração do fornecedor Browserstack para o fornecedor de TaaS Testingbot na sua respectiva abordagem e, em seguida, executar os testes na plataforma do Testingbot.

⁴ *Selenium Webdriver*: <<http://www.seleniumhq.org/projects/webdriver/>>

⁵ *JUnit*: <<http://junit.org/junit4/>>

⁶ *Saucelabs*: <<https://wiki.saucelabs.com/display/DOCS/Test+Configuration+Options/>>

⁷ *Browserstack*: <<https://www.browserstack.com/automate/capabilities/>>

⁸ *Testingbot*: <<https://testingbot.com/support/other/test-options/>>

Tabela 5 – Configurações das capacidades utilizadas nos testes automatizados em cada uma das tarefas de migração

Configurações	Tarefa 1	Tarefa 2	Tarefa 3	Tarefa 4
Plataformas	Local	Saucelabs⁶	Browserstack⁷	TestingBot⁸
Nome da Build	Build_Local	Build_Saucelabs	Build_Browserstack	Build_Testingbot
Nome do Teste	Teste Local	Teste Saucelabs	Teste Browserstack	Teste Testingbot
Sistema Operacional	Windows	Linux	Windows	Yosemite
Versão do Sistema Operacional	10	-	10	-
Navegador	Mozilla Firefox	Mozilla Firefox	Internet Explorer	Google Chrome
Versão do Navegador	46.0 e 45.9.0	45.0	11.0	52.0
Resolução de Tela	1024x768	1024x768	1280x1024	2560x1600

Na terceira e quarta tarefas, foram coletados os dados necessários para a validação da pesquisa, com objetivo de quantificar a eficiência e o esforço de migração de ambas as abordagens. Após o término de cada tarefa cada estudante deveria também enviar todo o seu projeto para posterior análise estatística do esforço de migração, por meio do código fonte em Java.

As configurações das plataformas mostradas na Tabela 5 foram trocadas ao migrar de um provedor para o outro, justamente para medir o esforço e a dificuldade na troca de capacidades específicas entre os fornecedores de TaaS.

O tempo gasto em cada uma das tarefas foi considerável, pois avaliou-se o cenário com apenas uma única configuração de capacidades de teste para cada fornecedor de TaaS. Em um cenário real, poderiam ser incluídas diversas outras combinações de capacidades com diferentes sistemas operacionais, dispositivos e navegadores com as suas respectivas versões. Porém, tal cenário inviabilizaria o experimento, pois aumentaria exponencialmente a sua complexidade.

4.2.9 Tarefas Experimentais do 2º Dia de Curso

No segundo dia de curso realizado no IFSULDEMINAS e na UFSCar, foi fornecido aos participantes um novo treinamento com as instruções para realizar as tarefas de gerenciamento dos resultados dos testes executados no primeiro dia de curso. A coleta de informações e a sumarização dos resultados dos testes automatizados ocorreu por meio das APIs REST^{9,10,11} de cada um dos três fornecedores de TaaS, conforme ilustrado na Figura 57 do Apêndice B.1. Após o treinamento, os participantes do grupo A realizaram a migração e geração de relatórios nos três fornecedores de TaaS, com a abordagem Multi-TaaS, enquanto o grupo B executou as mesmas tarefas, porém utilizando a abordagem convencional.

As tarefas de migração do 2º experimento são listadas a seguir e foram realizadas na mesma ordem, ou seja, da tarefa de número 1 a 3 e executadas, uma única vez, em cada um dos

⁹ API REST Saucelabs: <<https://wiki.saucelabs.com/display/DOCS/The+Sauce+Labs+REST+API>>

¹⁰ API REST Browserstack: <<https://www.browserstack.com/automate/rest-api>>

¹¹ API REST Testingbot: <<https://testingbot.com/support/api>>

dois grupos A e B.

Todos os indivíduos dos dois grupos receberam os projetos equivalentes com os mesmos serviços RESTful do Saucelabs e um conjunto de testes automatizados definidos de acordo com as suas respectivas abordagens. O projeto entregue para o grupo A, no segundo experimento, foi composto por 14 linhas de código e complexidade ciclomática igual 1. Todos esses valores ilustram o grau de complexidade dos testes automatizados e do código responsável por consumir a API REST dos fornecedores de TaaS antes da execução das tarefas pelos participantes. O projeto entregue para o grupo B foi composto por 1059 linhas de código e complexidade ciclomática igual 1,10. Todos esses valores ilustram o grau de complexidade dos testes automatizados e do código responsável por consumir a API REST dos fornecedores de TaaS antes da execução das tarefas pelos participantes.

As tarefas de migração listadas a seguir utilizaram aplicações que consumiam os serviços RESTful dos fornecedores TaaS. Todas as tarefas foram realizadas na mesma ordem, ou seja, da tarefa de número 1 a 3 e executadas uma única vez em cada um dos dois grupos A e B. Após o processo de migração, os participantes tinham que reexecutar todos os testes automatizados no novo fornecedor. As tarefas eram de natureza muito semelhante para ambos os grupos. Ao final da primeira tarefa, todos os participantes preencheram um formulário com o tempo de início e o tempo final, antes de iniciar a execução da próxima tarefa.

A primeira tarefa possuía caráter didático, servindo apenas de referência para as demais tarefas e não foram coletados dados. A seguir, são apresentadas uma descrição de cada uma das tarefas realizadas durante a condução do 2º dia do experimento, com base nos serviços RESTful:

Tarefa 1 - Cada grupo deveria realizar a coleta e a sumarização dos resultados dos testes do Saucelabs com o objetivo de introduzir os alunos e familiarizá-los com a API REST do Saucelabs¹²;

Tarefa 2 - Cada grupo deveria realizar a migração do serviço de teste do Saucelabs para o Browsertack e coletar e sumarizar os resultados dos testes do Browserstack na sua respectiva abordagem;

Tarefa 3 - Cada grupo deveria realizar a migração do serviço de teste do Browserstack para o Testingbot e coletar e sumarizar os resultados dos testes do Testingbot na sua respectiva abordagem.

Os relatórios gerados em cada tarefa são de extrema importância para o gerenciamento dos resultados dos testes.

Foi desenvolvido também um programa em Java Swing¹³ para utilização em todas as tarefas do segundo dia de curso, necessário para a geração do *token* de autorização referentes ao *framework open source* OAuth2 utilizados pelos fornecedores no acesso de recursos por

¹² API REST Saucelabs: <<https://wiki.saucelabs.com/display/DOCS/The+Sauce+Labs+REST+API>>

¹³ Token Generator: <www.ricardoramosdeoliveira.com.br/ferramentas/TokenGenerator.jar>

aplicações clientes, conforme ilustrado na Figura 28. O *token* é responsável pela proteção às APIs REST dos fornecedores.

Figura 28 – Programa para a geração de *Token*

Fonte: Autoria própria

No segundo dia de curso no IFSULDEMINAS, foi gerado o relatório parcial pelos participantes, com quantidade de informações reduzidas, de modo a diminuir a complexidade da implementação a ser realizada pelos alunos, contendo apenas as informações básicas, como por exemplo, sistema operacional, navegador, versão do navegador e status do teste. No entanto, foi exigido que os participantes implementassem as chamadas REST referentes ao relatório parcial gerado conforme o passo a passo disponível no próprio site do pesquisador e de acordo com o exemplo fornecido do Saucelabs.

Constatou-se que a abordagem Multi-TaaS foi efetiva em comparação com a abordagem convencional, no experimento realizado no IFSULDEMINAS, uma vez que todos os participantes do grupo A referente aos alunos que utilizaram a abordagem Multi-TaaS conseguiram finalizar todas as tarefas em tempo hábil. Os participantes do grupo B, que utilizaram a abordagem convencional, não conseguiram finalizar as tarefas 2 e 3. Com o objetivo de que todos os participantes conseguissem concluir todas as tarefas do segundo dia de curso no tempo estipulado, ou seja, que ambas as abordagens fossem efetivadas, houve uma adaptação tanto de ajustes nos relatórios gerados por cada uma das abordagens, quanto na implementação de cada tarefa realizada na UFSCar, com o objetivo de tornar equivalentes as tarefas executadas pelos participantes do grupo B e do grupo A.

Considerando a inefetividade da abordagem convencional, no segundo dia de curso do IFSULDEMINAS, conforme visto anteriormente, foram realizadas adaptações e ajustes que foram aplicados no segundo dia de experimento realizado na UFSCar.

O relatório gerado pelos participantes da UFSCar continham tanto as informações básicas, como por exemplo, sistema operacional, navegador, versão do navegador e *status* do teste, quanto informações avançadas, como *screenshots*, vídeo e *logs* dos fornecedores. No entanto, não houve a necessidade de implementar os serviços RESTful, pois todos os participantes que utilizaram a abordagem convencional receberam todos esses códigos prontos. Dessa forma, os participantes deveriam entender o funcionamento da aplicação cliente para conseguirem implementar os principais métodos a seguir, necessários para geração do relatório final:

1. Método `carregarUltimaExecucao()` responsável por coletar e obter os dados dos resultados do último teste executado;
2. Método `getParametrosParaGeracaoArquivo()` responsável pela obtenção de todos os parâmetros necessários para a sumarização dos resultados em forma de relatório.

4.2.10 Descrição da Instrumentação do Experimento

A instrumentação de ambos os experimentos do IFSULDEMINAS e da UFSCar incluem os seguintes artefatos:

1. Formulário de caracterização dos sujeitos, desenvolvido a partir da ferramenta do Google Forms¹⁴ com o objetivo de obter informações pessoais, acadêmicas e profissionais dos sujeitos, abordando, inclusive, questões sobre o nível de conhecimento de automatização de testes¹⁵;
2. Material do curso sobre TaaS na nuvem¹⁶;
3. Projetos desenvolvidos para ambas as abordagens de migração¹⁷;
4. Documentação com a descrição das capacidades necessárias para migração entre os fornecedores de TaaS^{18,19,20};
5. Termo de consentimento devidamente assinado pelos participantes²¹;
6. Repositório utilizado durante a execução do experimento por meio do site²² do próprio pesquisador com o gerenciador de conteúdo Wordpress²³ para auxiliar os estudantes na execução das tarefas e também na documentação e configuração das capacidades dos fornecedores de TaaS;

¹⁴ Google forms: <<https://www.google.com/forms/about/>>

¹⁵ Formulário de caracterização: <<https://goo.gl/forms/DaFUwgeE22p5VNnD2>>

¹⁶ Slides do Curso: <http://www.ricardoramosdeoliveira.com.br/experimento/slides_curso.pdf>

¹⁷ Todos os projetos do doutorado serão disponibilizados após todas as publicações no link: <http://www.ricardoramosdeoliveira.com.br/experimento/projetos_finais_doutorado.zip>

¹⁸ Configuração das capacidades do Saucelabs: <<https://wiki.saucelabs.com/display/DOCS/Platform+Configurator#/>>

¹⁹ Configuração das capacidades do Browserstack: <<https://www.browserstack.com/automate/java>>

²⁰ Configuração das capacidades do Testingbot: <<https://testingbot.com/support/getting-started/junit.html>>

²¹ Termo de consentimento: <http://www.ricardoramosdeoliveira.com.br/experimento/termo_consentimento.pdf>

²² Site do Experimento: <<http://ricardoramosdeoliveira.com.br/experimento-doutorado/>>

²³ Wordpress: <<https://wordpress.com/>>

7. Formulário para coleta de dados das tarefas^{24 25};
8. Tutorial para a geração do *token* de autorização;²⁶;
9. Tabelas com as divisões dos grupos e das aplicações de teste;²⁷;
10. Formulário de conclusão do curso com o objetivo de obter os *feedbacks* dos alunos²⁸.

Para acessar o site do pesquisador com todos os procedimentos necessários para a execução de cada tarefa do experimento basta acessar o endereço web²⁹ e inserir a senha **taas123**.

4.2.11 Validade dos Experimentos

Nesta subseção são apresentados os principais riscos que podem comprometer os resultados e as conclusões obtidas no experimento. Assim, a avaliação de validade costuma ser dividida basicamente em dois tipos, descritos a seguir:

Validade interna: verifica-se se o relacionamento entre o tratamento observado e o resultado são causais (WOHLIN *et al.*, 2000).

Com o objetivo de aumentar a validade interna do experimento, os sujeitos foram selecionados aleatoriamente entre os períodos e divididos em dois grupos A (Multi-TaaS) e B (Convencional), de acordo com o conhecimento de automatização de testes obtidos no questionário de caracterização do perfil dos sujeitos. No experimento conduzido no IFSULDEMINAS nenhum dos participantes possuíam conhecimentos sobre automatização de testes UAT. No entanto no experimento conduzido na UFSCar houveram participantes com conhecimento sobre automação de testes, porém com o objetivo de equilibrar os grupos, esses participantes foram divididos igualmente e aleatoriamente entre os grupos A e B, tanto no 1º quanto no 2º experimento.

Um fator que pode comprometer a validade interna é a complexidade dos testes e dos programas utilizados. Portanto, os testes foram desenvolvidos especificamente para este experimento, destinado a apoiar a investigação das hipóteses experimentais apresentadas na Subseção 4.2.4. O sistema web utilizado para os testes foi baseado em uma aplicação web para apoio a Sistemas de Informação de Ambiente Universitário (SInAU), inspirado nas regras e procedimentos comuns de uma gestão acadêmica. Note que o domínio de aplicação escolhido

²⁴ *Formulário de Tarefa do 1 Dia de Curso:* <http://www.ricardoramosdeoliveira.com.br/experimento/formulario_tarefas_1_dia.pdf>

²⁵ *Formulário de Tarefa do 2 Dia de Curso:* <http://www.ricardoramosdeoliveira.com.br/experimento/formulario_tarefas_2_dia.pdf>

²⁶ *Tutorial Token Generator:* <<http://www.ricardoramosdeoliveira.com.br/arquivos/TokenGenerator.pdf>>

²⁷ *Tabela de Grupos:* <<http://www.ricardoramosdeoliveira.com.br/arquivos/TabelaGrupos.pdf>>

²⁸ *Formulário de conclusão:* <<https://goo.gl/forms/OkJSxms7ELmisxpY2>>

²⁹ *Tarefas do Experimento:* <<http://ricardoramosdeoliveira.com.br/experimento-doutorado/>>

(gerenciamento educacional) teve a vantagem de ser facilmente compreensível pelos participantes, garantindo assim que os requisitos do sistema poderiam ser facilmente interpretados pelos estudantes.

Validade externa: verifica a capacidade de generalizar os resultados de um experimento (WOHLIN *et al.*, 2000).

Como mencionado na Subseção 4.2.7, a seleção dos indivíduos para a pesquisa utilizou os alunos dos cursos da área de Ciência da Computação e Engenharia de Computação. Todos sujeitos familiarizados com o desenvolvimento de software, tanto para as disciplinas quanto para softwares pessoais. Portanto, assume-se que eles são representativos para a população do experimento (HÖST; REGNELL; WOHLIN, 2000; FALESSI *et al.*, 2017).

Uma limitação que impede de generalizar a pesquisa é quanto à maior quantidade de participantes e desenvolvedores/testadores mais experientes, bem como à maior quantidade de combinações de capacidades, diferentes domínios de aplicações de testes. Novas métricas e maior quantidade de testes automatizados e mais complexos possibilitariam aumentar ainda mais a confiabilidade dos resultados. Sendo assim, com o objetivo de reduzir as limitações e generalizar os resultados, novos experimentos com uma quantidade maior de participantes foram conduzidos. A reaplicação do experimento com um número maior de participantes é interessante para avaliar as possíveis diferenças nos resultados das tarefas executadas.

O escopo do estudo se restringiu a testes automatizados no ambiente de nuvem. Pode-se afirmar que as diferentes combinações de configurações das tarefas dentro desse domínio são bastante abrangentes. O conjunto de testes e configurações das capacidades não ameaçam a validade do estudo para esse domínio. Por fim, um processo de migração real pode envolver colaboração entre diferentes equipes de desenvolvedores e testadores. Por isso, o experimento não pode ser generalizado, pois a pesquisa avaliou as atividades de forma individual, sem a colaboração ou interação com outros membros do mesmo grupo.

Um outro fator que poderia comprometer os dados do experimento, seria os esquecimento por parte dos alunos ao anotar o tempo no formulário, porém os alunos eram periodicamente lembrados para não esquecerem de anotar o tempo inicial e final de cada tarefa no formulário. A expectativa do pesquisador também poderia influenciar os resultados, porém o pesquisador procurou ser o mais imparcial possível, evitando assim intervenções e auxiliando os alunos apenas nas dúvidas em relação ao não entendimento da tarefa.

4.3 Preparação e Configuração dos Ambientes

Após a definição e o planejamento do estudo, a próxima etapa no processo de experimentação é a execução. Esta é uma etapa muito delicada no experimento, pois envolve o fator humano. A coleta dos dados deve ser realizada de maneira que não cause efeito significativo

no processo sendo estudado. Nesta etapa, também é realizada a validação preliminar dos dados experimentais.

É importante destacar que, como não houve suporte para a instalação de máquinas virtuais, o ambiente de migração para condução dos experimentos do IFSULDEMINAS e da UFSCar foram construídos a partir dos seguintes artefatos:

- Ferramentas devidamente instaladas e configuradas no laboratório;
- Documentação e especificação de cada um dos *fornecedores de TaaS* analisados e
- Código fonte dos testes automatizados.

4.3.1 O Ambiente de Migração - IFSULDEMINAS

Devido às configurações limitadas das máquinas do laboratório, ao invés de utilizar máquinas virtuais, foi criado um usuário específico para o pesquisador, com privilégios de administrador, para não sobrecarregar as máquinas dos laboratórios. Caso houvesse qualquer imprevisto na condução do experimento, o pesquisador conseguiria resolver os eventuais problemas de permissão em qualquer uma das máquinas do laboratório.

As configurações de cada um dos computadores do laboratório do IFSULDEMINAS utilizados para o treinamento e também para realização das tarefas do experimento estão disponíveis na web³⁰.

4.3.2 O Ambiente de Migração - UFSCar

O ambiente de migração foi criado com auxílio da plataforma *open source* PortableApps³¹. De acordo com as informações do próprio site da PortableApps (PORTABLE-APPS.COM, 2017), a plataforma foi desenvolvida para o Windows, sendo amplamente utilizada por desenvolvedores de software e hardware do mundo inteiro.

O PortableApps pode ser utilizado a partir de uma unidade flash USB, HD externo ou outros dispositivos como o próprio computador pessoal. A utilização do PortableApps no experimento ocorreu por meio da instalação dos softwares desejados dentro da sua própria plataforma e, após a instalação dos softwares necessários para o experimento, foram efetuadas cópias manuais em cada uma das máquinas do laboratório da SIN.

Os benefícios do PortableApps são semelhantes ao uso de uma máquina virtual quanto a sua praticidade, como por exemplo, manter todas as configurações da plataforma e o estado das

³⁰ *Configurações do IFSULDEMINAS*: <http://www.ricardoramosdeoliveira.com.br/experimento/Configuracoes_IFSULDEMINAS.pdf>

³¹ *PortableApps*: <<https://portableapps.com/>>

aplicações em execução, garantindo assim, o controle total do ambiente e a portabilidade das ferramentas utilizadas, independentemente de privilégios de administrador.

Após a preparação do ambiente no PortableApps foi necessário realizar a instalação em todas as máquinas do laboratório da SIn manualmente.

Todas as configurações de cada um dos computadores do laboratório da UFSCar utilizados para o treinamento e também para realização das tarefas do experimento estão disponíveis na web³².

4.4 Execução dos Experimentos

O experimento controlado foi conduzido para avaliar a qualidade da abordagem Multi-TaaS quanto à efetividade, eficiência, dificuldade e esforço de migração. Sendo assim, foi possível encontrar evidências que comprovassem a efetividade, o aumento da eficiência e a redução da dificuldade e do esforço de migração de um fornecedor TaaS para outro, levando em consideração o tempo de migração, tamanho do projeto ou linhas de código e complexidade ciclomática dos testes automatizados.

A execução do experimento foi realizada em duas etapas, divididas em 2 dias. No primeiro dia, foi fornecido aos participantes um treinamento sobre os principais conceitos de automatização de teste, Teste como Serviço (TaaS) na Computação em Nuvem, bem como a utilização da ferramenta Selenium Webdriver e do *framework* de teste JUnit. Após a primeira sessão de treinamento, todos os participantes executaram as tarefas descritas na Subseção 4.2.8 referente ao primeiro experimento controlado e às tarefas descritas na Subseção 4.2.9 referente ao segundo experimento controlado. Ambos os grupos A e B de cada experimento utilizaram os mesmos fornecedores de TaaS, porém um grupo de alunos utilizou a abordagem Multi-TaaS, enquanto o outro grupo utilizou a abordagem convencional.

As tarefas foram elaboradas o mais próximo possível de um cenário real. Cada participante dos grupos A e B tinha que efetuar a migração entre os fornecedores Saucelabs, Browserstack e Testingbot utilizando as suas respectivas abordagens. A Figura 29 mostra uma visão geral das tarefas de migração com três entidades envolvidas, nomeadas por consumidor TaaS, provedor TaaS e aplicação web. Durante o processo de migração dos fornecedores de TaaS, os participantes tinham que executar 4 testes automatizados referentes ao CRUD (*Create*, *Read*, *Update* e *Delete*) de Universidade da aplicação SInAU, descrita melhor na subseção 4.2.11. Após a conclusão do processo de migração e reexecução de todos os testes pelo novo fornecedor de TaaS, os participantes deviam anotar o tempo final no formulário de tarefas.

A reexecução dos testes automatizados serve para garantir que não houve impacto durante o processo de migração. É necessário verificar se as alterações do código de teste ou

³² Configurações da UFSCar: <http://www.ricardoramosdeoliveira.com.br/experimento/Configuracoes_UFSCar.pdf>

Figura 29 – Visão geral do processo de migração das tarefas

Fonte: Autoria própria

configuração estão funcionando corretamente sem prejudicar as etapas anteriores (TRAN *et al.*, 2011). O tempo de migração mais o tempo de reexecução dos testes foram contabilizados de forma conjunta. O grupo B realizou as mesmas tarefas e na mesma ordem do grupo A, porém utilizando a abordagem convencional. Não houve revezamento, ou seja, troca de abordagens entre os grupos.

No segundo dia de curso foi fornecido aos participantes um novo treinamento com as instruções para realizar as tarefas de coleta e sumarização dos resultados dos testes executados no primeiro dia de curso. Após o treinamento, metade dos participantes realizaram a geração de relatórios bem como a migração entre fornecedores de TaaS com a abordagem Multi-TaaS, enquanto a outra metade executou as mesmas tarefas e com as mesmas ferramentas, porém utilizando a abordagem convencional.

Cabe ressaltar que não houve coleta de dados no segundo dia de curso do IFSULDEMINAS. Devido à quantidade reduzida de participantes, não obteve-se a quantidade mínima necessária para as análises estatísticas. Porém, a reaplicação do experimento com um número maior de participantes para avaliar e comparar a eficiência e o esforço de migração em relação à API REST de cada um dos fornecedores de TaaS foi realizada na UFSCar.

4.5 Análise dos Resultados e Avaliação das Hipóteses - IFSULDEMINAS e UFSCar

Nesta seção, são descritos os métodos estatísticos utilizados e apresentados os resultados da avaliação das hipóteses definidas na Seção 4.2.4 quanto à efetividade, eficiência, dificuldade e do esforço de migração das plataformas de teste na nuvem entre as abordagens Multi-TaaS e a abordagem convencional. Primeiramente, são descritos os resultados referentes à efetividade

e eficiência e em seguida são apresentados os resultados das análises estatísticas referentes à dificuldade e do esforço de migração.

Os experimentos controlados finais foram realizados no IFSULDEMINAS e na UFSCar. Todos os experimentos realizados e as suas respectivas análises estatísticas são mostrados na Tabela 6, onde é possível perceber claramente que no IFSULDEMINAS, o 1º Experimento (1º Exp) foi realizado duas vezes, uma no 2º período (2º P) e outra no 4º período (4º P), como os participantes não conseguiram concluir, em tempo hábil, as tarefas do 2º experimento no IFSULDEMINAS e devido à quantidade reduzida de participantes e amostras, nenhuma análise estatística referente à eficiência e ao esforço foram realizadas.

Apesar de o pesquisador ter realizado a coleta de dados no 1º experimento (1º Exp) da UFSCar tanto na turma A (TA) quanto na turma B (TB), houve um equívoco e os códigos da 1ª tarefa não foram entregues, apenas os dados dos formulários, sendo assim, não serão apresentadas as análises estatísticas do 1º experimento da UFSCar quanto ao esforço de migração, conforme mostrado pelo símbolo *x* na Tabela 6. O equívoco ocorreu porque o pesquisador não exigiu a entrega do código fonte e concluiu que apenas os dados da 2ª tarefa do 1º experimento seriam suficientes para explicar o comportamento de toda a amostra. Esse equívoco ocorreu porque as tarefas eram bem semelhantes e o pesquisador estava interessado em realizar uma análise estatística geral, assim como todas as análises realizadas no IFSULDEMINAS.

De acordo com a Tabela 6 é possível perceber também que como os participantes do IFSULDEMINAS não entregaram o código fonte do segundo experimento, foi realizada uma análise parcial do tipo Likert representada pelo símbolo ●.

Tabela 6 – Tabela de análises estatísticas realizadas para os experimentos do IFSULDEMINAS e da UFSCar

	IFSULDEMINAS				UFSCar			
	2º P		4º P		TA		TB	
	1º Exp	2º Exp	1º Exp	2º Exp	1º Exp	2º Exp	1º Exp	2º Exp
Efetividade	✓	✓	✓	✓	✓	✓	✓	✓
Eficiência	✓	<i>x</i>	✓	<i>x</i>	✓	✓	✓	✓
Esforço	✓	<i>x</i>	✓	<i>x</i>	<i>x</i>	✓	<i>x</i>	✓
Dificuldade	✓	●	✓	✓	✓	✓	✓	✓

É importante destacar que as análises estatísticas correspondentes à eficiência e à dificuldade foram realizadas a partir dos formulários entregues pelos participantes, com o tempo de início e término de cada tarefa e o nível de dificuldade, respectivamente. As análises estatísticas correspondentes ao esforço foram realizadas a partir do código fonte de teste entregue pelos participantes no repositório criado no próprio site do pesquisador.

O mesmo experimento controlado conduzido no IFSULDEMINAS foi reaplicado a um número maior de participantes na UFSCar, cabe ressaltar também que foram realizadas dois tipos de análises estatísticas. As análises estatísticas realizadas a partir dos resultados do IFSULDEMINAS foram gerais, ou seja, independente das tarefas, devido à quantidade reduzida

de participantes e amostras. Em contrapartida, as análises estatísticas realizadas para os resultados da UFSCar foram separadas por tarefa, devido não só à quantidade maior de participantes e amostras, mas principalmente para garantir que não houvesse interação do tratamento com as tarefas, ou seja, que as informações das tarefas não influenciassem nas informações do tratamento.

Os resultados dos experimentos conduzidos no IFSULDEMINAS e na UFSCar são sumarizados nas Tabelas 7, 8, 9, 10 e 11. Em que são apresentados para cada grupo os valores de mínimo, 1º quartil, mediana (2º quartil), média, 3º quartil, máximo e desvio padrão (DP). Considerando as estatísticas apresentadas em sua maioria, o grupo A possui valores menores que o grupo B. Destacando que para o 2º experimento, todos os valores do grupo A são ínfimos em relação aos valores do grupo B. No decorrer das próximas subseções descreve-se de forma detalhada todos os resultados obtidos da efetividade, eficiência, dificuldade e do esforço de migração de cada um dos grupos.

Tabela 7 – Resultados do experimento controlado (eficiência e esforço de migração) - IFSULDEMINAS

	Grupo	Min.	1º Qu.	Mediana	Média	3º Qu.	Max.	DP
Eficiência (Tempo)	A	5,000	6,250	10,000	14,070	20,500	32,000	9,841
	B	5,000	8,500	19,000	30,640	50,250	85,000	26,363
Esforço	A	0,044	0,055	0,089	0,124	0,183	0,280	0,087
	B	0,058	0,098	0,222	0,356	0,582	0,989	0,307

Tabela 8 – 1º Experimento controlado – Resultados da eficiência (tempo de migração) - UFSCar

	Grupo	Min.	1º Qu.	Mediana	Média	3º Qu.	Max.	DP
Tarefa 1	A	3,000	9,500	14,000	16,700	21,500	56,000	11,542
	B	8,000	13,000	22,000	26,280	35,000	71,000	16,590
Tarefa 2	A	2,000	5,500	13,000	16,090	25,500	37,000	11,143
	B	4,000	9,500	14,000	18,620	20,500	74,000	15,165

Tabela 9 – 1º Experimento controlado – Resultados do esforço de migração - UFSCar

	Grupo	Min.	1º Qu.	Mediana	Média	3º Qu.	Max.	NA's	DP
Tarefa 1	A	0,020	0,029	0,039	0,039	0,049	0,058	21	0,027
	B	0,027	0,035	0,059	0,080	0,127	0,161	33	0,059
Tarefa 2	A	0,012	0,026	0,045	0,048	0,072	0,094	7	0,027
	B	0,009	0,020	0,028	0,038	0,044	0,135	9	0,030

Tabela 10 – 2º Experimento controlado – Resultados da eficiência (tempo de migração) - UFSCar

	Grupo	Min.	1º Qu.	Mediana	Média	3º Qu.	Max.	DP
Tarefa 1	A	2,000	6,000	10,000	10,120	13,500	22,000	5,931
	B	22,000	41,000	63,500	64,790	87,000	115,000	28,652
Tarefa 2	A	3,000	3,000	4,500	6,000	7,250	17,000	4,066
	B	5,000	25,500	32,000	31,880	43,000	56,000	13,436

Tabela 11 – 2º Experimento controlado – Resultados do esforço de migração - UFSCar

	Grupo	Min.	1º Qu.	Mediana	Média	3º Qu.	Max.	NA's	DP
Tarefa 1	A	0,0002	0,0007	0,0010	0,0011	0,0014	0,0027	2	< 0,001
	B	0,1231	0,2263	0,4035	0,3727	0,5094	0,6299	3	0,157
Tarefa 2	A	0,0002	0,0004	0,0004	0,0007	0,0008	0,0026	2	< 0,001
	B	0,0411	0,2307	0,2629	0,2561	0,3318	0,4474	3	0,105

4.5.1 *Análise Geral da Efetividade*

De acordo com as normas (ISO25010, 2011) e (ISO25066, 2016), a efetividade avalia se os resultados de uma atividade foram alcançados em um determinado espaço de tempo, independentemente dos custos envolvidos. A análise da efetividade torna-se binária. Se os participantes concluíram de forma adequada todas as tarefas no tempo estipulado, a abordagem é considerada efetiva, caso contrário, a abordagem não é efetiva.

Dessa forma, conclui-se que ambas as abordagens, Multi-TaaS e convencional, foram efetivas no 1º experimento do IFSULDEMINAS, pois todos os participantes dos dois grupos Multi-TaaS e Convencional, conseguiram concluir todas as tarefas no tempo estipulado. No 2º experimento do IFSULDEMINAS, na qual apenas a abordagem Multi-TaaS foi efetiva, pois conforme detalhado na Subseção 4.2.9, os participantes deveriam implementar todas as chamadas REST e nenhum deles conseguiu concluir nenhuma das tarefas no tempo estipulado.

Quanto aos dois experimentos realizados na UFSCar, conclui-se que ambas as abordagens, Multi-TaaS e convencional foram efetivas, pois todos os participantes dos dois grupos A e B concluíram todas as tarefas dos experimentos no tempo estipulado.

4.5.2 *Análises Estatísticas do 1º Experimento (Escrita, Configuração e Execução de Testes)*

Nesta subseção, são descritos os resultados da avaliação comparativa da eficiência, dificuldade e esforço de migração entre a abordagem Multi-TaaS e a abordagem convencional. As análises estatísticas do 1º experimento são referentes à eficiência de migração quanto à escrita de testes automatizados, configuração das capacidades e execução dos testes automatizados.

Todas as análises estatísticas foram realizadas por meio do R. As Tabelas 27, 28 e 29 do Apêndice C.1 foram construídas para as análises dos dados dos experimentos conduzidos no IFSULDEMINAS e na UFSCar respectivamente.

Análises Estatísticas da Eficiência

Durante a condução do experimento controlado, os dados da pesquisa foram obtidos pelos próprios participantes por meio de formulário e ocorreu após o término de cada tarefa com supervisão do pesquisador. Para melhor visualizar o comportamento das 28 amostras do IFSULDEMINAS e 124 amostras da UFSCar, foram gerados os histogramas apresentados nas Figuras 30 e 31. Todos os histogramas mostram o número de participantes por intervalo de tempo de cada um dos grupos.

Conforme mostrado nas Figuras 30 e 31, é possível perceber claramente que o grupo A possui o tempo de execução distribuído em valores consideravelmente menores que o grupo B no IFSULDEMINAS e na 1ª tarefa do experimento realizado na UFSCar.

Figura 30 – Histogramas da avaliação da eficiência - IFSULDEMINAS

Fonte: Autoria própria

Figura 31 – Histogramas da avaliação da eficiência - UFSCar

(a) 1º Experimento - 1ª Tarefa

(b) 1º Experimento - 2ª Tarefa

Fonte: Autoria própria

Pode-se verificar também por meio dos gráficos ilustrados na Figura 32, que todos os quartis do grupo A estão abaixo dos respectivos quartis do grupo B e a diferença na mediana (2º quartil) é de aproximadamente 9 minutos, e no terceiro quartil a diferença é de aproximadamente 30 minutos. No entanto, para validar estatisticamente que essas diferenças são significativas, foram considerados os testes *t*-student e Wilcoxon. Além dos testes, utilizou-se também o

modelo de regressão linear para descrever a relação entre os dados.

O gráfico da Figura 33 (a) mostra que todos os quartis do grupo A estão abaixo dos respectivos quartis do grupo B e a diferença na mediana (2º quartil) é de aproximadamente 8 minutos, enquanto que no terceiro quartil a diferença é de aproximadamente 13,5 minutos.

Figura 32 – Box plot da eficiência - IFSULDEMINAS

Fonte: Autoria própria

Figura 33 – Box plots da avaliação da eficiência - UFSCar

(a) 1º Experimento - 1ª Tarefa

(b) 1º Experimento - 2ª Tarefa

Fonte: Autoria própria

Testes Estatísticos - IFSULDEMINAS

1. Teste *t*-student (ou test-*t*) é um teste paramétrico para médias, ou seja, compara-se a média do grupo A com a média do grupo B. Os resultados mostram que o tempo no grupo A é significativamente inferior ao do grupo B. Além disso, o test-*t* indica, com nível de significância de 95% (p -valor = 0,021), que a média de tempo do grupo A é significativamente menor que a média de tempo do grupo B.
2. Teste Wilcoxon não pareado para duas amostras, também conhecido por Mann-Whitney teste. Esse é um teste não paramétrico para família de distribuições, ou seja, compara-se se o grupo A e o grupo B vieram da mesma família de probabilidade. Os resultados mostram que o tempo do grupo A é disperso em valores significativamente menores quando comparados ao grupo B. O teste de Wilcoxon indica, com nível de significância de 95% (p -valor = 0,0463), que o grupo A é distribuído em valores significativamente menores que o grupo B.

Os resultados dos testes indicam uma variação significativa (p -valores < 0,05) na comparação dos grupos A e B, portanto, aceita-se a hipótese alternativa experimental H_1 e conclui-se que a abordagem proposta é mais eficiente do que a abordagem convencional.

Testes Estatísticos - UFSCar

Para validar estatisticamente que as diferenças da eficiência da primeira tarefa são significativas, foram considerados os testes *t*-student e Wilcoxon. A Tabela 12 apresenta os resultados do teste-*t* e Wilcoxon nas duas tarefas do primeiro experimento, além do ganho médio de eficiência, ou seja, a proporção do ganho, o quanto a média do grupo A é menor do que a média do grupo B, conforme mostrado na Equação 4.1.

$$\text{Ganho Médio} = \left(\frac{\text{Média B} - \text{Média A}}{\text{Média B}} \right) \times 100 \quad (4.1)$$

Os resultados da Tabela 12 mostram que o tempo do grupo A é disperso em valores significativamente inferiores quando comparados ao do grupo B na primeira tarefa. Considerando o nível de significância de 5% (p -valores < 0,05), temos que a abordagem Multi-TaaS requer menor tempo dos participantes comparada com a abordagem convencional na migração do Saucelabs para o Browserstack. Portanto aceita-se a hipótese alternativa experimental H_1 e conclui-se que a abordagem proposta é mais eficiente do que a abordagem convencional na primeira tarefa de migração.

Tabela 12 – Resultados dos testes-*t* e Wilcoxon para o 1º experimento controlado

1º Experimento - Eficiência				
	Tarefa	Test- <i>t</i>	Wilcoxon	Ganho
Eficiência	1	<i>p</i> -valor = 0,0048	<i>p</i> -valor = 0,0044	36,5%
	2	<i>p</i> -valor = 0,2275	<i>p</i> -valor = 0,2922	13,6%

Apesar dos resultados significativos da primeira tarefa, é possível visualizar na Tabela 12 que os resultados da eficiência na segunda tarefa referente à migração do Browserstack para o Testingbot não foi significativa (*p*-valores > 0,05) devido às semelhanças entre as tarefas, pode ter ocorrido o efeito de aprendizado dos participantes após a primeira tarefa. Um outro motivo seria a documentação escassa para essa tarefa específica, o que pode ter levado os participantes da abordagem Multi-TaaS a se confundirem ou se enganarem durante a execução da tarefa. Apesar dos testes não terem apontado diferença significativa, observou-se que o ganho na utilização da abordagem Multi-TaaS sugere uma redução do tempo de migração de 13,6% em média.

Modelo de Regressão Linear - IFSULDEMINAS

Com o objetivo de medir o quanto a abordagem Multi-TaaS interfere no tempo total das tarefas, foi elaborado um modelo de Regressão Linear, com base nas 28 amostras obtidas no IFSULDEMINAS, conforme a Equação 4.2. A aplicação do modelo de regressão tem por finalidade não só identificar se o tratamento interfere ou não no tempo de execução das tarefas, mas também de avaliar o quanto o tratamento interfere nesse tempo.

O modelo de regressão linear paramétrico foi ajustado por mínimos quadrados, conforme apresentado na Equação 4.2 a seguir:

$$T_i = \alpha_0 + \alpha_1 X_{1,i} + \alpha_2 X_{2,i} + \varepsilon_i \quad (4.2)$$

Sendo as variáveis: T_i representa a variável de resposta, ou seja, o tempo total da tarefa. Os valores de X representam as variáveis explicativas (co-variáveis), sendo que $X_{1,i}$ representa as tarefas, $X_{2,i}$ indica o tratamento. Os parâmetros α_0 (interceptor), α_1 , α_2 auxiliam no ajuste do modelo e são obtidos minimizando a soma dos erros ao quadrado $f(\alpha_0, \alpha_1, \alpha_2) = \sum_{i=1}^{28} \varepsilon_i^2$. O valor de ε_i é uma componente aleatória do modelo baseada no tempo total das tarefas, que representa o erro que o modelo não consegue explicar. Cabe ressaltar, no entanto, que a turma foi removida do modelo, pois não apresentou significância para explicar o tempo de execução das tarefas.

Com os parâmetros a ser estimados por mínimos quadrados e a componente aleatória não observada, que será assumida como independente e identicamente distribuída seguindo a distribuição de probabilidade Gaussiana, com média zero e variância constante ($N(0, \sigma^2)$). A estimativa, erro padrão e *p*-valor do teste de significância de cada co-variável são apresentados na Tabela 13.

Tabela 13 – Estimativa, erro padrão e p -valor do teste de significância (IFSULDEMINAS)

Parâmetros	Estimativa	Erro Padrão	p -valor
Interceptor	69,571	18,210	0,001
Tarefa	-7,786	3,556	0,036
Tratamento	-16,571	7,009	0,026

Pela Tabela 13 pode-se concluir que a co-variável Tratamento, referente ao uso da abordagem de interesse, pode reduzir em média aproximadamente 17 minutos no tempo de execução total de cada tarefa.

Portanto, com as análises de todos os testes, é possível concluir que a nova abordagem diminui de forma significativa o tempo de conclusão das tarefas. Consequentemente, o modelo estatístico indica que a abordagem de interesse é mais eficiente do que a abordagem convencional. Além disso, as análises revelaram que houve um aumento da eficiência média em 54% com a utilização da abordagem Multi-TaaS.

Modelo de Regressão Linear - UFSCar

Além dos testes t -student e Wilcoxon, considerou-se também o mesmo modelo de regressão linear paramétrico apresentado anteriormente na equação 4.2 para descrever a relação entre os dados das 124 amostras obtidas na UFSCar. O modelo de regressão foi utilizado com o objetivo de medir o quanto a abordagem Multi-TaaS interfere ou não no tempo total de execução das tarefas e avaliar quanto o tratamento interfere nesse tempo.

De acordo com a Tabela 14, é possível visualizar os resultados do modelo de regressão para as duas tarefas do primeiro experimento controlado. É possível concluir também, por meio da Tabela 14, que o Tratamento referente ao uso da abordagem de interesse foi significativo (p -valor $< 0,05$), reduzindo aproximadamente 10 minutos no tempo de execução total da primeira tarefa. Na segunda tarefa, a co-variável Tratamento não obteve significância no modelo de regressão (p -valor $> 0,05$). No entanto, o modelo apontou que o Tratamento reduziu em aproximadamente 3 minutos o tempo de execução da tarefa.

Tabela 14 – Estimativa, erro padrão e p -valor do teste de significância do 1º experimento controlado

1º Experimento - Eficiência				
	Parâmetros	Estimativa	Erro Padrão	p -valor
Tarefa 1	Interceptor	26,282	2,392	$< 0,001$
	Tratamento	-9,586	3,928	0,017
Tarefa 2	Interceptor	18,615	2,214	$< 0,001$
	Tratamento	-2,528	3,635	0,489

Considerando as análises de todos os testes e do modelo de regressão linear paramétrico, é possível demonstrar que a nova abordagem diminui de forma significativa o tempo para

conclusão das tarefas, consequentemente pode-se concluir também que a abordagem de interesse é mais eficiente do que a abordagem convencional. Portanto considera-se que houve um aumento da eficiência média em 36,5% na primeira tarefa e de 13,6% na segunda tarefa com a utilização da abordagem Multi-TaaS.

Análises Estatísticas do Esforço

O processo de estimar o esforço de manutenção é extremamente desafiador, pois depende de vários fatores. Entre eles, pode-se citar o tamanho do projeto, tipo de manutenção, experiência dos sujeitos, nível de familiaridade do programador com o sistema sendo mantido, complexidade do projeto, documentação do projeto e tecnologias utilizadas (NGUYEN; BOEHM; DANPHITSANUPHAN, 2009).

Sun e Li (2013a) destacam que a estimativa de custo ou esforço por meio de modelos é muito importante para a engenharia de software e ao mesmo tempo desafiadora quanto a precisão. As abordagens para a estimativa de esforço geralmente são divididas entre estimativas baseadas na intuição de um especialista (*experts*) e aquelas realizadas por modelos algoritmos (SUN; LI, 2013a). Neste doutorado utilizou-se a estimativa de esforço por meio do modelo de regressão linear com base no tempo de migração, tamanho e complexidade dos testes automatizados.

No contexto do experimento, considerou-se que a experiência dos sujeitos era a mesma para todos os participantes, uma vez que os participantes da UFSCar que possuíam conhecimento de automatização de testes foram divididos de forma aleatória e equivalentes entre os dois grupos com o objetivo principal de balancear os grupos, além disso, nenhum dos participantes do IFSULDEMINAS possuíam conhecimentos de automatização de testes e todas as tecnologias e as documentações utilizadas nos experimentos foram as mesmas. Considerou-se todos esses fatores relativamente invariantes, porque os participantes realizaram as mesmas tarefas de migração no mesmo ambiente com os testes automatizados semelhantes. O esforço do programador/testador foi definido como o tempo total gasto na tarefa de migração e reexecução dos testes, tamanho ou linhas de código e complexidade ciclomática do projeto.

A estimativa de esforço desta pesquisa pode contribuir para a tomada de decisão dos Engenheiros de Software quanto aos riscos associados ao *vendor lock-in* na troca de fornecedores de serviço de testes na nuvem. Além disso, essa estimativa pode auxiliar também no planejamento de projetos de software, orçamentos, análises de investimentos e custos de uma forma geral (SOMMERVILLE, 2010; JORGENSEN, 2014).

A Equação 4.3 foi criada para auxiliar os investidores na avaliação de riscos de empresas de software (SOMMERVILLE, 2010; BOEHM *et al.*, 2000). Essa equação é voltada para o custo do projeto e desenvolvimento de software no geral, porém não leva em consideração os aspectos do tempo de interação entre consumidor e fornecedor e nem o tempo de migração entre

fornecedores de TaaS no ambiente de computação em nuvem.

$$E = A * Tamanho^B * M \quad (4.3)$$

O termo A é uma constante que depende das práticas utilizadas pela organização ou empresa. O termo $Tamanho$ é referente ao tamanho do código fonte que pode ser definido, por exemplo, em número de linhas de código (LOC) ou pontos de função. O termo B corresponde ao tamanho e a complexidade do sistema. Além disso, o expoente B indica que o esforço não é linear ao tamanho do projeto. O termo M é um multiplicador que depende de diferentes atributos relacionados ao processo, produtos e desenvolvimento (SOMMERVILLE, 2010). Os valores de B e M podem variar conforme a formação ou a experiência da equipe de desenvolvimento.

O modelo conhecido como *COConstructive COst MOdel* (COCOMO) II permite estimar o esforço de desenvolvimento, prazo e tamanho da equipe em projetos de software. Porém não leva em consideração as características de manutenção ou migração no contexto da nuvem (SUN; LI, 2013a). De acordo com Sun e Li (2013a), na prática, o esforço de migração depende do contexto e geralmente são criadas estimativas baseadas em modelos de regressão. Neste doutorado, as estimativas foram obtidas por meio de um modelo de regressão linear baseado nos parâmetros de tempo de migração, tamanho e complexidade ciclomática do código de teste automatizado.

A partir da Equação 4.3, foi elaborada uma outra equação para o esforço de migração entre fornecedores de TaaS conforme a Equação 4.4. Essa equação considera que o esforço (E) deve ser diretamente proporcional à complexidade ciclomática do teste (CC), tamanho ou linhas de código de teste (S) e ao tempo de migração (T) (SUN; LI, 2013a; TRAN *et al.*, 2011). Cabe ressaltar que o esforço foi calculado para cada um dos participantes dos grupos A e B. Cada termo foi dividido pelo seu valor máximo com intuito de normalizar os valores resultantes entre 0 e 1, pois cada termo da Equação 4.4 possui unidades de medida diferentes.

$$E = \frac{CC}{Max(CC)} * \frac{S}{Max(S)} * \frac{T}{Max(T)} \quad (4.4)$$

Para auxiliar a coleta dos dados referentes ao tamanho e à complexidade ciclomática, foram utilizadas as ferramentas Metrics³³ na versão 1.3.8 da IDE de desenvolvimento Eclipse e um repositório web no próprio site do pesquisador³⁴. O *plugin* Metrics auxiliou na coleta dos dados referentes às linhas de código e à complexidade ciclomática.

A Tabela 27 foi construída para a análise dos dados. Nessa tabela, é possível perceber que os valores da complexidade ciclomática mantiveram-se constantes, ou seja, o grupo do tratamento com o uso da abordagem proposta manteve o valor 1, o valor do grupo de controle com o uso da abordagem convencional foi de 1,09. O tratamento é um valor booleano: quando o

³³ *Plugin Metrics*: <<http://metrics2.sourceforge.net/>>

³⁴ *Web Repository*: <<http://www.ricardoramosdeoliveira.com.br/>>

tratamento é igual a 1 significa o uso da abordagem proposta (grupo A) e quando o tratamento é igual a 0 significa o uso da abordagem convencional (grupo B).

A análise estatística do esforço descrita nesta seção segue as mesmas instruções citadas nas análises anteriores referentes à eficiência. Sendo assim, foram gerados também os histogramas representados na Figura 34, com os resultados das análises do esforço referentes a cada um dos grupos, onde é possível perceber claramente que o grupo A possui o esforço distribuído em valores consideravelmente menores que o grupo B.

Figura 34 – Histogramas da avaliação do esforço - IFSULDEMINAS

Fonte: Autoria própria

Nota-se no gráfico boxplot ilustrado na Figura 35 que todos os quartis do grupo A estão abaixo dos respectivos quartis do grupo B e a diferença na mediana (2º quartil) é de aproximadamente 0,133, enquanto no terceiro quartil a diferença é de aproximadamente 0,399. No entanto para validar estatisticamente que essas diferenças são significativas, considera-se os testes de média *t*-student e Wilcoxon, além de um novo modelo de regressão linear.

1) Sendo as médias dos Grupos A e B de 0,124 e 0,356, respectivamente, os resultados do teste-*t* mostram com nível de significância de 95% (p -valor = 0,0078) que o esforço médio do grupo A é significativamente menor ao do grupo B.

2) Os resultados do teste de Wilcoxon mostram com nível de significância de 95% (p -valor = 0,0114), que o esforço do grupo A é distribuído em valores significativamente menores que os do grupo B.

Os resultados do teste-*t* e Wilcoxon indicam uma variação significativa (p -valores < 0,05) na comparação do esforço entre os grupos A e B, portanto, aceita-se a hipótese alternativa experimental H_1 e conclui-se que a utilização da abordagem proposta reduz o esforço de migração

Figura 35 – Box plot do esforço - IFSULDEMINAS

Fonte: Autoria própria

em relação à abordagem convencional.

Com o objetivo de medir o quanto a abordagem Multi-TaaS interfere no esforço das tarefas, foi elaborado um outro modelo de Regressão Linear, com base nas 28 amostras obtidas. O modelo foi ajustado por mínimos quadrados conforme apresentado na Equação 4.5:

$$E_i = \beta_0 + \beta_1 X_{1,i} + \beta_2 X_{2,i} + \varepsilon_i \quad (4.5)$$

Sendo as variáveis: E_i representa a variável de resposta, ou seja, o esforço da tarefa. Os valores de X representam as mesmas variáveis explicativas dos modelos de regressão anterior. Os parâmetros β_0 (interceptor), β_1 , β_2 auxiliam no ajuste do modelo e são obtidos minimizando a soma dos erros ao quadrado. O valor de ε_i é uma componente aleatória do modelo baseada no esforço das tarefas que representa o erro que o modelo não consegue explicar.

A estimativa, erro padrão e p -valor do teste de significância de cada co-variável são apresentadas na Tabela 15, onde pode-se concluir que o uso da abordagem proposta reduziu, em média, aproximadamente 0,232 no esforço de cada tarefa.

Tabela 15 – Estimativa, erro padrão e p -valor do teste de significância (IFSULDEMINAS)

Parâmetros	Estimativa	Erro Padrão	p -valor
Interceptor	0,798	0,206	< 0,001
Tarefa	-0,088	0,039	0,035
Tratamento	-0,232	0,079	0,007

Portanto com a análise de todos os testes, é possível demonstrar que a nova abordagem

diminui de forma significativa o esforço no processo de migração entre fornecedores de TaaS. Além disso, pode-se concluir que o esforço médio foi reduzido em 65% com a utilização da abordagem proposta.

A Tabela 28 do Apêndice C.1 foi construída para a análise dos dados para o primeiro experimento da UFSCar, onde é possível perceber uma grande quantidade de valores NA (*Not Available*) que representam os dados não disponíveis e NA é a forma como a linguagem de programação R interpreta os valores ausentes.

A mesma Tabela 28 apresenta também os dados da segunda tarefa, onde é possível perceber que os valores da complexidade ciclomática mantiveram-se constantes, ou seja, o grupo do tratamento com o uso da abordagem proposta manteve o valor 1,15 e o valor do grupo de controle com o uso da abordagem convencional foi de 1,05. O tratamento é um valor booleano, quando o tratamento é igual a 1 significa o uso da abordagem proposta (grupo A) e quando o tratamento é igual a 0 significa o uso da abordagem convencional (grupo B).

Como não foi exigida a entrega do código fonte da 1ª Tarefa do 1º Experimento na UFSCar pelos motivos explicados anteriormente na Seção 4.5 e na Subseção 4.2.7, não houve amostras suficientes para conclusões estatísticas da primeira tarefa. Decidiu-se, então, não realizar os testes estatísticos para o esforço de migração para o 1º experimento da UFSCar.

Avaliação da Dificuldade

A avaliação da dificuldade utilizou dados do tipo Likert baseada em medidas quantitativas de atitudes, carácter ou traços de personalidade (BOONE; BOONE, 2012). De acordo com Boone e Boone (2012), é necessário relacionar o tipo Likert com a escala de medição de Steven para obter os procedimentos de análises estatísticas adequados.

A escala de medição de Steven é composta por outras 4 escalas: nominal, ordinal, intervalo e proporção. Na escala nominal, as observações expressam categorias com base em equivalências, por exemplo, gênero, cor de olhos e raça. Na escala ordinal, os números possuem alguma ordem e expressam de forma direta ou indireta o relacionamento “Maior que”, como por exemplo, notas de letras, rankings e conquistas (baixa, média, alta). A escala de intervalo possui uma característica específica, pois não possui um zero absoluto e os números representam a distância relativa entre os próprios pontos da escala com base em uma ordem pré-definida. A única diferença entre a escala de proporção da escala de intervalo, é que a escala de proporção possui um zero absoluto, um exemplo da escala de proporção é a idade ou anos de experiência (BOONE; BOONE, 2012).

A escala de medição de Steven adotada para esse experimento foi a ordinal, uma vez que é possível atribuir uma pontuação ou uma ordem pré-determinada para as respostas dos

questionários^{35 36}, como por exemplo, “Muito Difícil” pode representar o valor -2, “Difícil” pode representar o valor -1, “Regular” pode representar o valor 0, “Fácil” pode representar o valor 1 e “Muito Fácil” pode representar o valor 2. A resposta “Regular” equivale à resposta neutra.

Os resultados das análises estatísticas do primeiro experimento referente ao 2º período do IFSULDEMINAS são apresentados na Tabela 16 e ilustrados na Figura 36, onde é possível perceber claramente que 75% dos participantes que utilizaram a abordagem Multi-TaaS avaliaram a primeira tarefa como “Muito Fácil” ou “Fácil”, enquanto que 50% dos participantes da abordagem convencional avaliaram a primeira tarefa como “Muito Difícil” ou “Difícil”. Já na segunda tarefa, 75% dos participantes que utilizaram a abordagem Multi-TaaS avaliaram a abordagem como “Muito Fácil” contra 50% dos participantes que utilizaram a abordagem convencional.

Tabela 16 – Resultados da dificuldade do 2º período - IFSULDEMINAS

		1º Experimento - 2º Período				
	Grupo	Muito Difícil	Difícil	Regular	Fácil	Muito Fácil
Tarefa 1	A	25,0%	0,0%	0,0%	50,0%	25,0%
	B	25,0%	25,0%	25,0%	25,0%	0,0%
Tarefa 2	A	0,0%	0,0%	25,0%	0,0%	75,0%
	B	0,0%	25,0%	25,0%	0,0%	50,0%
Total de Amostras						16

Figura 36 – Histogramas da avaliação da Dificuldade - 1º Experimento do IFSULDEMINAS

Fonte: Autoria própria

Os resultados das análises estatísticas do primeiro experimento referente ao 4º período do IFSULDEMINAS são apresentados na Tabela 17 e ilustrados na Figura 37, onde é possível perceber claramente que todos os participantes dos dois grupos marcaram a resposta neutra na

³⁵ Formulário de Tarefa do 1 Dia de Curso: <http://www.ricardoramosdeoliveira.com.br/experimento/formulario_tarefas_1_dia.pdf>

³⁶ Formulário de Tarefa do 2 Dia de Curso: <http://www.ricardoramosdeoliveira.com.br/experimento/formulario_tarefas_2_dia.pdf>

primeira tarefa, enquanto na segunda tarefa o grupo B obteve um melhor resultado. Acredita-se que esse resultado ocorreu de forma aleatória.

Tabela 17 – Resultados da dificuldade do 4º período - IFSULDEMINAS

	1º Experimento - 4º Período						
	Grupo	Muito Difícil	Difícil	Regular	Fácil	Muito Fácil	Total
Tarefa 1	A	0,0%	0,0%	100,0%	0,0%	0,0%	3
	B	0,0%	0,0%	100,0%	0,0%	0,0%	3
Tarefa 2	A	0,0%	0,0%	66,7%	33,3%	0,0%	3
	B	0,0%	0,0%	0,0%	66,7%	33,3%	3
	Total de Amostras						12

Figura 37 – Histogramas da avaliação da Dificuldade - 1º Experimento do IFSULDEMINAS

Fonte: Autoria própria

Os resultados das análises estatísticas do primeiro experimento da UFSCar são apresentados na Tabela 18 e mostrados na Figura 38, onde é possível perceber claramente que 39,13% dos participantes que utilizaram a abordagem Multi-TaaS avaliaram a primeira tarefa como “Muito Fácil” ou “Fácil”, enquanto que 44,74% dos participantes da abordagem convencional avaliaram a primeira tarefa como “Muito Difícil” ou “Difícil”. Na segunda tarefa 39,13% dos participantes que utilizaram a abordagem Multi-TaaS avaliaram a abordagem como “Muito Fácil” ou “Fácil” contra apenas 28,95% dos participantes que utilizaram a abordagem convencional.

Tabela 18 – Resultados da dificuldade do 1º experimento - UFSCar

	1º Experimento - 2º Período						
	Grupo	Muito Difícil	Difícil	Regular	Fácil	Muito Fácil	Total
Tarefa 1	A	4,3%	21,7%	34,8%	30,4%	8,7%	23
	B	2,6%	42,1%	34,2%	21,1%	0,0%	38
Tarefa 2	A	4,3%	34,8%	21,7%	39,1%	0,0%	23
	B	10,5%	15,8%	44,7%	28,9%	0,0%	38
Total de Amostras							122

Figura 38 – Histogramas da avaliação da Dificuldade - 1º Experimento da UFSCar

Fonte: Autoria própria

4.5.3 Análises Estatísticas do 2º Experimento (Gerenciamento, Coleta de informações e Sumarização dos Resultados)

Nesta subseção, são descritos os resultados da avaliação comparativa da eficiência, dificuldade e esforço de migração entre a abordagem Multi-TaaS e a abordagem convencional. O segundo experimento avaliou por meio de análises estatísticas a eficiência de migração quanto ao gerenciamento, a coleta de informações e sumarização dos resultados por meio das APIs REST de cada fornecedor de TaaS.

Análises Estatísticas da Eficiência

Conforme descrito anteriormente na Seção 4.5, como os participantes não conseguiram concluir as tarefas do 2º experimento no IFSULDEMINAS no tempo estipulado e devido à quantidade reduzida de participantes e amostras, as análises estatísticas da eficiência e do esforço não foram realizadas. Portanto essa subseção mostra apenas as análises estatísticas da eficiência do 2º experimento realizado na UFSCar.

Os histogramas apresentados na Figura 39 ilustram o comportamento das 80 amostras e a quantidade de participantes por intervalo de tempo de cada um dos grupos referentes ao segundo experimento controlado. Conforme mostrado também na Figura 39, é possível perceber claramente que o grupo A possui o tempo de execução das tarefas distribuídos em valores consideravelmente inferiores ao do grupo B, tanto na primeira tarefa apresentada pela Figura 39 (a), quanto na segunda tarefa apresentada na Figura 39 (b).

Pode-se verificar também em ambos os gráficos *boxplots*, tanto o da primeira tarefa apresentado na Figura 40 (a), quanto o gráfico da segunda tarefa apresentado na Figura 40 (b), que todos os quartis do grupo A estão abaixo dos respectivos quartis do grupo B e a diferença na mediana (2º quartil) da primeira tarefa é de aproximadamente 54 minutos, enquanto que na segunda tarefa a diferença na mediana (2º quartil) é de aproximadamente 28 minutos. Os gráficos

Figura 39 – Histogramas da avaliação da eficiência - UFSCar

Fonte: Autoria própria

da Figura 40 ilustram também que a diferença do terceiro quartil na primeira tarefa (Figura 40 (a)) é de aproximadamente 74 minutos e na segunda tarefa (Figura 40 (b)) essa diferença é de aproximadamente 36 minutos.

Figura 40 – Box plots da avaliação da eficiência - UFSCar

Fonte: Autoria própria

Os resultados da Tabela 19 mostram que o tempo do grupo A é disperso em valores

significativamente inferiores quando comparados ao grupo B, revelando indícios de que a abordagem Multi-TaaS requer menor tempo e portanto maior eficiência por parte dos participantes comparada com a abordagem convencional (p -valores $< 0,05$). Portanto aceita-se a hipótese alternativa experimental H_1 e conclui-se que a utilização da abordagem Multi-TaaS aumenta em média em 84,4% a eficiência na primeira tarefa e em 81,2% na segunda tarefa em comparação com a abordagem convencional referente à migração entre fornecedores de TaaS quanto ao gerenciamento, coleta de informações e sumarização dos resultados de testes automatizados.

Tabela 19 – Resultados dos testes- t e Wilcoxon para o 2º experimento controlado - UFSCar

2º Experimento - Eficiência				
Eficiência	Tarefa	Teste- t	Wilcoxon	Ganho
	1	p -value $< 0,0001$	p -value $< 0,0001$	84,4%
	2	p -value $< 0,0001$	p -value $< 0,0001$	81,2%

Além dos testes, utilizou-se também o mesmo modelo de regressão linear paramétrico apresentado anteriormente na equação 4.2 para descrever a relação entre os dados das 80 amostras obtidas. O modelo de regressão foi utilizado com o objetivo de medir o quanto a abordagem Multi-TaaS interfere ou não no tempo total de execução das tarefas e avaliar o quanto o tratamento interfere nesse tempo.

De acordo com a Tabela 20, pode-se concluir que o Tratamento, referente ao uso da abordagem de interesse, foi significativo (p -valor $< 0,05$), reduzindo aproximadamente 55 minutos no tempo de execução total da primeira tarefa. É possível concluir também por meio da Tabela 20 que o modelo de regressão foi significativo para a segunda tarefa (p -valor $< 0,05$), reduzindo aproximadamente 26 minutos no tempo de execução total da tarefa.

Tabela 20 – Estimativa, erro padrão e p -valor do teste de significância do 2º experimento controlado - UFSCar

2º Experimento - Eficiência				
	Parâmetros	Estimativa	Erro Padrão	p -valor
	Interceptor	64,792	4,613	$< 0,001$
Tarefa 1	Tratamento	-54,667	7,294	$< 0,001$
	Interceptor	31,875	2,197	$< 0,001$
Tarefa 2	Tratamento	-25,875	3,473	$< 0,001$

Considerando as análises de todos os testes e do modelo de regressão linear paramétrico das duas tarefas do segundo experimento, conclui-se que a abordagem Multi-TaaS reduz, de forma significativa, o tempo para conclusão das tarefas. Pode-se concluir que a abordagem de interesse é mais eficiente do que a abordagem convencional. Portanto considera-se que houve um aumento da eficiência média em mais de 80% nas duas tarefas com a utilização da abordagem Multi-TaaS.

Após analisar a eficiência das abordagens, os resultados das análises estatísticas do esforço obtidos nos experimentos do IFSULDEMINAS e da UFSCar serão detalhados nas próximas subseções.

Análise Estatística do Esforço

A Tabela 29 do Apêndice C.1 foi construída para a análise dos dados para o segundo experimento e segue as mesmas convenções da Tabela 28, com destaque apenas para os valores da complexidade ciclomática que foi de 1 para a abordagem adotada e variando entre 1,09 e 1,15 para a abordagem convencional.

Os histogramas apresentados na Figura 41 apresentam a frequência ou número de sujeitos por intervalo da estimativa de esforço de cada um dos grupos. Além disso, o histograma ilustra o comportamento das 80 amostras de acordo com os resultados das análises do esforço, onde é possível perceber claramente que o grupo A possui o esforço distribuído em valores consideravelmente menores que o grupo B nas duas tarefas, ou seja, todas as amostras concentraram-se na escala entre 0,0 e 0,1, enquanto a maior parcela dos participantes do grupo B concentraram-se entre o intervalo 0,1 e 0,7.

Os resultados da segunda tarefa apresentados na Figura 41 (b) mostram que a concentração de participantes da abordagem convencional reduziram o intervalo do esforço entre 0,0 e 0,5 em comparação com a primeira tarefa. Essa redução ocorreu devido à semelhança entre as duas tarefas, pois os participantes do grupo B aprenderam a realizar a migração e consequentemente isso se refletiu nos resultados finais do esforço de migração. No entanto os participantes que utilizaram a abordagem Multi-TaaS mantiveram o esforço constante entre 0,0 e 0,1.

Pode-se verificar também, por meio do gráfico *boxplot* da primeira tarefa mostrado na Figura 42 (a), que todos os quartis do grupo A estão abaixo dos respectivos quartis do grupo B, nas duas tarefas. A diferença na mediana (2º quartil) da primeira tarefa é de aproximadamente 0,403, enquanto que no terceiro quartil a diferença é de aproximadamente 0,508. Na segunda tarefa, a diferença da mediana (2º quartil) é de aproximadamente 0,263, enquanto que no terceiro quartil a diferença é de aproximadamente 0,331 na escala de esforço.

A análise estatística do esforço descrita nesta subseção segue as mesmas instruções citadas nas análises anteriores referentes à eficiência. Sendo assim, foram considerados os testes *t*-student e Wilcoxon para validar estatisticamente que as diferenças do esforço nas duas tarefas são significativas.

A Tabela 21 apresenta os resultados do teste-*t*, Wilcoxon e da proporção do ganho médio do esforço para as duas tarefas do primeiro experimento. Os resultados da Tabela 21 mostram também que o esforço do grupo A são dispersos em valores significativamente inferiores quando comparados ao do grupo B nas duas tarefas. Considerando o nível de significância de 5% (p -valores $< 0,05$), temos que a abordagem Multi-TaaS reduz o esforço de migração dos

Figura 41 – Histogramas da avaliação do esforço - 2º Experimento - UFSCar

Fonte: Autoria própria

Figura 42 – Box plots da avaliação do esforço - 2º Experimento - UFSCar

Fonte: Autoria própria

participantes comparada com a abordagem convencional na migração do Saucelabs para o Browserstack quanto na migração do Browserstack para o Testingbot. Portanto aceita-se a hipótese alternativa experimental H_1 e conclui-se que a abordagem Multi-TaaS requer menor esforço de migração comparado com a abordagem convencional.

Tabela 21 – Resultados dos testes-*t* e Wilcoxon para o 2º experimento controlado - UFSCar

2º Experimento - Esforço				
	Tarefa	Teste- <i>t</i>	Wilcoxon	Ganho
Esforço	1	$p\text{-value} < 0,0001$	$p\text{-value} < 0,0001$	99,7%
	2	$p\text{-value} < 0,0001$	$p\text{-value} < 0,0001$	99,7%

Além dos testes, considera-se também o mesmo modelo de regressão linear paramétrico apresentado anteriormente pela Equação 4.5 para descrever a relação entre os dados das 80 amostras obtidas. O modelo de regressão foi utilizado com o objetivo de medir o quanto a abordagem Multi-TaaS interfere, ou não, no esforço total das tarefas e avaliar o quanto o tratamento interfere nesse esforço.

De acordo com a Tabela 22, é possível visualizar os resultados do modelo de regressão linear do esforço para as duas tarefas do segundo experimento controlado. É possível concluir também que o Tratamento referente ao uso da abordagem de interesse foi significativo ($p\text{-valor} < 0,05$), reduzindo aproximadamente 0,37 do esforço total da primeira tarefa e aproximadamente 0,26 pontos do esforço total da segunda tarefa.

Tabela 22 – Estimativa, erro padrão e *p*-valor do teste de significância do 2º experimento controlado - UFSCar

2º Experimento - Esforço				
	Parâmetros	Estimativa	Erro Padrão	<i>p</i> -valor
Tarefa 1	Interceptor	0,372	0,026	$< 0,001$
	Tratamento	-0,371	0,042	$< 0,001$
Tarefa 2	Interceptor	0,256	0,017	$< 0,001$
	Tratamento	-0,255	0,028	$< 0,001$

Considerando as análises de todos os testes e do modelo de regressão linear paramétrico, é possível demonstrar que a nova abordagem reduz de forma significativa o esforço de migração das tarefas em comparação com a abordagem convencional. Portanto, de acordo com a Tabela 21, considera-se que houve uma redução do esforço médio em 99,7% nas duas tarefas com a utilização da abordagem Multi-TaaS.

Avaliação da Dificuldade

Os resultados das análises estatísticas do segundo experimento referente ao 2º período do IFSULDEMINAS são apresentados na Tabela 23 e ilustrados na Figura 43, onde é possível perceber claramente que 50% dos participantes que utilizaram a abordagem Multi-TaaS avaliaram a primeira tarefa como “Fácil”. Já na segunda tarefa, 75% dos participantes que utilizaram a abordagem Multi-TaaS avaliaram-na como “Fácil” ou “Muito Fácil”. No entanto, não foi exigido aos participantes que preenchessem o formulário da dificuldade, dessa forma não houve coleta de dados do grupo B em nenhuma das duas tarefas do 2º experimento, pois nenhum dos participantes

Figura 44 – Histogramas da avaliação da Dificuldade - 2º Experimento

Fonte: Autoria própria

Os resultados das análises estatísticas do segundo experimento da UFSCar são apresentados na Tabela 25 e ilustrados na Figura 45, onde é possível perceber claramente que 62,2% dos participantes que utilizaram a abordagem Multi-TaaS avaliaram a primeira tarefa como “Muito Fácil” ou “Fácil”, enquanto que 78,26% dos participantes da abordagem convencional avaliaram a primeira tarefa como “Difícil” ou “Muito Difícil”. Na segunda tarefa todos os participantes que utilizaram a abordagem Multi-TaaS, 75% dos participantes avaliaram a tarefa como “Muito Fácil” ou “Fácil”. Em contrapartida, mais de 65% dos participantes da abordagem convencional avaliaram a segunda tarefa como “Difícil” ou “Muito Difícil”.

Tabela 25 – Resultados da dificuldade do 2º experimento - UFSCar

		1º Experimento - 2º Período					Total
		Grupo	Muito Difícil	Difícil	Regular	Fácil	
Tarefa 1	A		0,0%	12,5%	25,0%	31,3%	16
	B		39,1%	39,1%	21,7%	0,0%	23
Tarefa 2	A		4,3%	34,8%	21,7%	39,1%	23
	B		17,4%	47,8%	21,7%	13,0%	23
Total de Amostras							78

Portanto conclui-se que, nos dois experimentos do IFSULDEMINAS e da UFSCar de uma forma geral, a abordagem Multi-TaaS possibilitou facilitar o processo de migração entre fornecedores de TaaS devido à reutilização do código de teste, impactando consequentemente na redução de custos com manutenção.

4.6 Considerações Finais

Este capítulo apresentou todas as atividades e etapas dos experimentos conduzidos no IFSULDEMINAS e na UFSCar. Em seguida, foram descritos todo o planejamento e condução do experimento. Por fim, foram apresentadas as avaliações das hipóteses por meio de análises estatísticas. No próximo capítulo, serão apresentadas as conclusões desta tese de doutorado, bem

Figura 45 – Histogramas da avaliação da Dificuldade - 2º Experimento

Fonte: Autoria própria

como as principais contribuições, produções científicas, dificuldades e limitações. Finalmente, serão apresentados os trabalhos futuros.

CONCLUSÕES

5.1 Conclusão Geral

A portabilidade e a interoperabilidade são dois atributos de qualidade importantes para proporcionar a ampla adoção, de forma eficiente e efetiva, das novas tecnologias de Testes como Serviço (TaaS) na Computação em Nuvem. A falta desses dois requisitos, incluindo a falta de padronização, integra o problema de *vendor lock-in* de maior e menor grau no TaaS.

Durante o ciclo de vida do software, ocorrem constantes evoluções das aplicações web. Com o desenvolvimento de novas funcionalidades, torna-se necessário automatizar novos testes na nuvem. Nesse cenário, é possível perceber claramente que a dependência do consumidor em relação ao fornecedor de TaaS específico aumenta ao longo do tempo. Devido à forte dependência entre consumidor e fornecedor TaaS, problemas como o aumento de preço abusivo ou falta de disponibilidade técnica, por exemplo, podem ocorrer. A partir desses motivos, o consumidor pode desejar realizar a migração do serviço de teste na nuvem, ou seja, o processo de troca de um fornecedor TaaS para outro.

Nesse contexto, esta tese de doutorado teve o objetivo principal de investigar uma abordagem que reduzisse os impactos ocasionados pelo problema de *vendor lock-in* de maior e menor grau no TaaS e facilitasse o processo de migração de testes automatizados de aceitação do usuário (UAT) e sumarização de resultados dos testes para os engenheiros de software.

As seguintes questões de pesquisa foram elaboradas com objetivo principal de orientar todo o desenvolvimento do projeto ao longo desta tese, foram:

1. Como solucionar ou reduzir os impactos ocasionados pelo problema de *vendor lock-in* no contexto de Teste como Serviço (TaaS)?
2. Quais as etapas do processo de automatização de teste na nuvem são afetadas pelo problema

de *vendor lock-in*?

Para responder às questões de pesquisa, inicialmente foi realizada uma revisão sistemática, disponível no Apêndice A.1, com objetivo de encontrar e mapear as principais soluções para o problema de *vendor lock-in* no contexto de TaaS. As soluções encontradas giram em torno, por exemplo de, MDE, APIs ou padrão/protocolos abertos e camadas de abstração (solução adotada por este trabalho), entre outras. Após realizar todo o processo da revisão sistemática de planejamento, execução e análise dos resultados e, a partir dos principais artigos selecionados, foi possível vislumbrar além das vantagens e desvantagens de cada solução encontrada, entender e conhecer melhor todo o processo de automatização de testes no ambiente de nuvem, bem como todo o funcionamento do Teste como Serviço.

Em seguida, para entender melhor como o problema de *vendor lock-in* se manifestava nas diferentes etapas do processo de teste na nuvem, foram realizadas as seguintes atividades: projeto, planejamento, desenvolvimento e implantação de aplicação web na nuvem, automatização de testes UAT, configuração de capacidades, execução de testes na nuvem, gerenciamento quanto à coleta de informações e sumarização dos resultados de testes por meio das APIs REST proprietárias de cada um dos três fornecedores de TaaS utilizados. Todas essas atividades foram fundamentais para conhecer e entender melhor quais eram as etapas do processo de automatização de teste na nuvem afetadas pelo problema de *vendor lock-in*. As atividades de configuração de capacidades, escrita e execução de testes correspondem ao problema de *vendor lock-in* de menor grau. As etapas de gerenciamento dos resultados, configurações de Máquinas Virtuais (VMs) e propriedades de paralelismo do ambiente de testes automatizados na nuvem correspondem ao *vendor lock-in* de maior grau.

Para reduzir os impactos ocasionados pelo problema de *vendor lock-in* de menor e maior grau, em todo o processo de teste na nuvem, foi desenvolvida a nova abordagem, intitulada Multi-TaaS que é uma camada de abstração (ou *middleware*) que funciona de forma semelhante ao Selenium Webdriver, basicamente como se fosse um Selenium Webdriver para TaaS. Enquanto o Selenium Webdriver estabelece a portabilidade dos testes automatizados UAT entre os diferentes navegadores, a biblioteca Multi-TaaS estabelece a portabilidade dos testes automatizados UAT e da sumarização dos resultados de testes por meio da API REST entre os diferentes TaaS.

Devido à criação da extensão do JUnit e o uso dos padrões de projeto, o processo de migração da abordagem Multi-TaaS é simples e prático, pois ocorre por meio de arquivos de configurações, enquanto que na abordagem convencional, o processo é manual e ocorre de forma programática (*hard coded*), sem o devido planejamento ou preocupação com os riscos do *vendor lock-in*, contribuindo, conseqüentemente, para aumentar drasticamente os custos de manutenção da migração.

As principais características da abordagem Multi-TaaS são facilitar o processo de migração entre fornecedores de TaaS para os engenheiros de softwares (desenvolvedores e testadores),

favorecer o reúso, reduzir os custos de manutenção de migração e melhorar a legibilidade do código de teste automatizado UAT e consequentemente melhorar a qualidade do software no geral. Em contrapartida, a abordagem convencional dificulta o processo de migração entre os fornecedores de TaaS, encarecendo todo o processo de manutenção e migração tanto dos testes automatizados quanto do mecanismo de sumarização dos resultados, pois exige que o engenheiro de software realize todo o processo de reengenharia e reescrita dos testes automatizados UAT entre os diferentes fornecedores de TaaS.

É importante destacar que quanto menor o esforço, mais simples e eficiente for o processo de migração, maior será a portabilidade, ao passo que quanto mais complexo, difícil e custoso for o processo de migração, menor será a portabilidade. Neste contexto, a abordagem Multi-TaaS permite aumentar a portabilidade dos testes automatizados e do componente de sumarização dos resultados entre os diferentes fornecedores TaaS, reduzindo, consequentemente, os impactos ocasionados pelo problema de *vendor lock-in* no processo de teste na nuvem.

Dessa forma, utilizou-se o mecanismo de extensão do JUnit, padrões de projeto e arquivos de configuração para desacoplar o *script* de teste automatizado da plataforma TaaS em que ele será executado, facilitando consequentemente o processo de migração dos testes automatizados UAT para o engenheiro de software entre os diferentes fornecedores de TaaS. A abordagem adotada permite também abstrair todos os detalhes, especificidades e complexidade da comunicação e integração REST entre o consumidor e o fornecedor de TaaS. A geração de relatórios de defeitos ocorre por meio de chamadas às APIs REST proprietária de cada um dos fornecedores de TaaS e corresponde a uma das principais etapas do gerenciamento dos resultados dos testes na nuvem. A abordagem Multi-TaaS facilita também o processo de migração do software que consome as APIs REST dos diferentes TaaS.

Como prova de conceito foram desenvolvidos dois protótipos, um da arquitetura Multi-TaaS e outro da abordagem convencional, que foram utilizados como objetos de estudo para a condução dos experimentos controlados. Assim como a revisão sistemática, o experimento controlado é composto por etapas e atividades sistematizadas, sendo elas: definição, planejamento, operação, análise e interpretação dos resultados e, por último, a apresentação e empacotamento de todos os artefatos gerados pelo experimento. O experimento auxiliou no processo de validação das novas descobertas.

Apesar das desvantagens listadas na Seção 3.6, o uso da abordagem proposta possibilitou melhorar a eficiência e reduzir o esforço de migração entre fornecedores de TaaS. Os resultados dos experimentos foram analisados por meio de testes e modelos estatísticos, com o objetivo principal de avaliar as hipóteses de pesquisa. Portanto, os resultados dos experimentos conduzidos no IFUSULDEMINAS indicam que a abordagem Multi-TaaS permite melhorar a eficiência média em até 54% e reduzir o esforço médio de migração em 65% entre os fornecedores de TaaS em comparação com a abordagem convencional quanto à configuração das capacidades, escrita e execução de testes automatizados na nuvem. Os resultados dos experimentos conduzidos na

UFSCar indicam que a abordagem Multi-TaaS aumentou a eficiência média em mais de 80% e reduziu o esforço médio em 99.7% em comparação com a abordagem convencional quanto ao gerenciamento e sumarização dos resultados de teste. Por último, em ambos os experimentos do IFSULDEMINAS e da UFSCar, a avaliação da dificuldade de migração entre as plataformas de TaaS foi consideravelmente menor com a utilização da abordagem Multi-TaaS, em comparação com a abordagem convencional.

Portanto com os resultados dos experimentos controlados realizados, foi possível obter indícios de que a abordagem Multi-TaaS permite aumentar consideravelmente a eficiência e reduzir significativamente a dificuldade e o esforço de migração em comparação com a abordagem convencional. Conclui-se, também, que o uso da arquitetura de teste automatizado na nuvem contribui de forma significativa para a redução dos impactos ocasionados pelo problema de *vendor lock-in* de maior e menor grau no TaaS de forma portátil. Os estudos realizados com os experimentos controlados são promissores visto que o problema de *vendor lock-in* é relevante e de suma importância para a portabilidade e interoperabilidade no TaaS. A portabilidade da solução Multi-TaaS é o primeiro passo para o desenvolvimento de uma solução interoperável para os testes na nuvem. A interoperabilidade entre TaaS deve ser almejada, uma vez que a portabilidade e a interoperabilidade são os principais fatores que contribuem para a ampla adoção do modelo de serviço TaaS na computação em nuvem.

5.2 Principais Contribuições

No contexto da terceirização de serviços de teste oferecidos no ambiente de nuvem, as principais contribuições desta pesquisa são listadas a seguir:

1. **Camada de abstração Multi-TaaS:** foi desenvolvido um protótipo da nova abordagem como prova de conceito que permite desacoplar a escrita das capacidades em relação à execução dos testes automatizados e abstrair toda a complexidade da integração e comunicação REST entre os fornecedores de TaaS por meio do *plugin* ou extensão do JUnit, padrões de projeto e arquivo de configuração, possibilitando, consequentemente, aumentar a reusabilidade e melhorar a legibilidade do código de teste automatizado. Além disso, a arquitetura pode ser reutilizada para testar outras aplicações web de diferentes domínios, sendo totalmente flexível e extensível, permitindo a adição de novos fornecedores de TaaS;
2. **Avaliações Experimentais:** com o objetivo principal de demonstrar a aplicabilidade da abordagem Multi-TaaS e a sua relevância no mundo real. Os experimentos controlados conduzidos neste trabalho possibilitaram avaliar e comparar a eficiência, efetividade e o esforço de migração. Todos os artefatos gerados no processo de experimentação estão disponíveis no Capítulo 4 do experimento e fazem parte da etapa de empacotamento, servindo também para que outros pesquisadores possam reaplicar o experimento.

Entre as principais atividades diretamente relacionadas às contribuições deste trabalho, pode-se destacar a definição do problema de *vendor lock-in* no contexto do Teste como Serviço (TaaS), além de apresentar uma discussão aprofundada do seu impacto nas diferentes etapas do processo de teste de software na computação em nuvem. Outra atividade importante, foi a realização de uma revisão sistemática, conforme o Apêndice A.1, apresentando as principais soluções para o problema de *vendor lock-in* no contexto de TaaS, e ampliar o campo de pesquisa e apresentar as lacunas de pesquisa sobre a área para auxiliar os pesquisadores novatos.

O trabalho identificou as quatro principais etapas do processo de teste de software que são afetadas pelo *vendor lock-in* no contexto de Teste como Serviço, sendo elas: escrita, execução, configuração e gerenciamento. Com os resultados do presente trabalho, foi possível classificar e categorizar o problema de *vendor lock-in* em maior e menor grau e estabelecer diretrizes para reduzir os seus impactos de acordo com o seu respectivo grau. As diretrizes permitem auxiliar os engenheiros de software na construção de uma solução que seja portátil e, ao mesmo tempo, contornar o problema de *vendor lock-in* nas etapas do processo de teste na computação em nuvem, bem como no controle de qualidade e na tomada de decisão quanto à migração do serviço de teste.

Utilizou-se uma metodologia própria para avaliação dos resultados dos experimentos controlados, que permitiu comparar a qualidade da abordagem Multi-TaaS em relação à abordagem convencional. A metodologia foi aplicada da seguinte forma: a partir das definições das normas de qualidade de software ISO25010 (2011), ISO25066 (2016), foram elaboradas as métricas para quantificar os resultados quanto à eficiência e a efetividade. Os resultados da eficiência e do esforço de migração foram mensurados com base em estimativas e em análises estatísticas descritivas. Por último, a avaliação da dificuldade foi realizada por meio de dados do tipo Likert (BOONE; BOONE, 2012).

A nova abordagem permite auxiliar os engenheiros de softwares no controle de qualidade e na tomada de decisão de forma eficiente e, ao mesmo tempo, reduzir os custos associados com o esforço de migração dos testes automatizados e da sumarização dos resultados de teste no ambiente de nuvem. Consequentemente, as empresas interessadas em evitar o problema de *lock-in* na troca de fornecedor TaaS ou do seu serviço de teste na nuvem, vão obter um menor custo com os gastos de manutenção dos seus testes automatizados e sumarização dos resultados por meio da abordagem Multi-TaaS.

5.3 Produção Científica

As produções científicas geradas a partir da execução deste doutorado são listadas a seguir:

Publicados:

- **OLIVEIRA, R. R.; MARTINS, R. M.; SIMÃO, A. S. Impact of the vendor lock-in problem on testing as a service (TaaS).** In: 2017 IEEE International Conference on Cloud Engineering, IC2E 2017, Vancouver, BC, Canada, April 4-7, 2017. [s.n.], 2017. p. 190–196. Disponível em: <<https://doi.org/10.1109/IC2E.2017.30>>.
- **PINTO, V. H. S. C.; LUZ, H. J. F.; OLIVEIRA, R. R.; SOUZA, P. S. L.; SOUZA, S. R. S. A systematic mapping study on the multi-tenant architecture of saas systems.** In: The 28th International Conference on Software Engineering and Knowledge Engineering, SEKE 2016, Redwood City, San Francisco Bay, USA, July 1-3, 2016. [s.n.], 2016. p. 396–401. Disponível em: <<https://doi.org/10.18293/SEKE2016-068>>.

Submetidos:

- **OLIVEIRA, R. R.; UNGARELLI, R. N.; FIORUCCI, J. A.; SIMÃO, A. S. Comparative evaluation of the effort of migration among cloud testing vendors.** In: 2017 IEEE International Conference on Cloud Computing Technology and Science, CloudCom 2017, Hong Kong, December 11-14, 2017.
- **OLIVEIRA, R. R.; PINTO, V. H. S. C.; LUZ, H. J. F.; SIMÃO, A. S. A research roadmap of vendor lock-in solutions to testing as a service (TaaS).** Information & Software Technology (IST), 2017. (Journal)

5.4 Dificuldades e Limitações

A seguir são apresentadas as principais limitações encontradas durante o desenvolvimento e execução deste trabalho:

1. **Teste de programas na linguagem de programação Java:** umas das principais limitações é o fato de que a abordagem só pode ser utilizada para criar *sripts* de teste na linguagem de programação Java que testem quaisquer aplicações web. Uma solução para esse problema seria transformar todas as operações da abordagem Multi-TaaS em operações de uma API REST e disponibilizá-la, por meio de uma licença *open source*. Essa estratégia possibilitaria a escrita do teste independentemente de qualquer linguagem de programação, conforme ilustrado na Seção 2.7 pela Figura 54 letra c). Cabe ressaltar, no entanto, que esse tema será alvo de futuras investigações;
2. **Estabelecimento de um padrão de escrita de testes automatizados:** uma outra limitação importante decorre da convenção na escrita do teste para sua utilização, conforme detalhado na Seção 3.7. A convenção possui anotações, obrigatórias e opcionais, e seu uso é voltado para parametrizações e padrões de escrita de teste. Portanto, apesar de estabelecer

um padrão de escrita de teste, possibilita melhorar a legibilidade e a manutenibilidade do código de teste;

3. **Previsão de esforço para adicionar novos fornecedores de TaaS:** com a nova abordagem não é possível prever o esforço total de adicionar um novo TaaS, pois as APIs podem variar muito entre os diferentes fornecedores de TaaS do mercado. Porém, uma vez implementadas as características particulares das capacidades e API REST do novo fornecedor, o *vendor lock-in* de maior e menor grau, estariam resolvidos com a abordagem Multi-TaaS;
4. **Capacidade de generalizar os resultados:** uma limitação importante dos experimentos controlados refere-se à capacidade de generalizar os resultados para todos os contextos e diferentes domínios de um processo de migração real. Para isso, um estudo mais detalhado deve ser realizado com novos testes automatizados para diferentes domínios de aplicações web, maior quantidade de combinações de capacidades, novas métricas, bem como um número maior de participantes experientes e até mesmo uma maior quantidade de testes automatizados e mais complexos seriam necessários;
5. **Efeito de aprendizado das tarefas entre as abordagens:** não houve a avaliação do efeito de aprendizado das tarefas dos experimentos. Entretanto, assume-se que existe a possibilidade de que tal efeito tenha ocorrido, devido às semelhanças entre as tarefas experimentais. Além disso, o efeito de aprendizado pode ter ocorrido dentro de cada uma das abordagens, uma vez que não houve o revezamento, ou seja, a troca de abordagens entre os grupos. Na hipótese de ocorrência de tal efeito, o que se poderia afirmar é que no primeiro experimento tanto do IFSULDEMINAS quanto da UFSCar, o efeito de aprendizagem melhorou a eficiência, ou seja, o tempo de migração dos participantes que utilizaram a abordagem Multi-TaaS em comparação com os participantes que utilizaram a abordagem convencional. Apesar de haver uma redução no esforço de migração do grupo que utilizou a abordagem convencional, observou-se que o efeito de aprendizado (caso tenha ocorrido) contribuiu para manter o esforço de migração inferior e praticamente constante com a utilização da abordagem Multi-TaaS.

A seguir são listadas as principais dificuldades decorrentes deste trabalho:

1. **Mudanças nas APIs REST dos fornecedores de TaaS:** umas das principais dificuldades encontradas foi devido às frequentes mudanças que ocorriam nas APIs REST dos fornecedores de TaaS, impactando diretamente nos projetos dos respectivos grupos do experimento. Cada mudança na API REST pelos fornecedores de TaaS implicava na manutenção de código, tornando-se um processo muito custoso de horas depurando código para encontrar os defeitos e corrigi-los. Esse é um dos maiores problemas da API REST

dos fornecedores de TaaS, pois ao evoluir as suas APIs, infelizmente eles não mantêm a retrocompatibilidade, impactando diretamente no código dos seus clientes;

2. **Bloqueio referente à criação de novas contas:** uma dificuldade enfrentada no experimento controlado aconteceu quando foi solicitado aos participantes da UFSCar que criassem uma conta gratuita de teste no fornecedor de TaaS solicitado. Devido à grande quantidade de alunos criando contas gratuitas, de forma simultânea, alguns participantes foram bloqueados pelos fornecedores de TaaS por motivos de segurança, temendo ataques externos de invasão. Nesse caso, foi solicitado aos alunos que não conseguiram criar a conta, que não contabilizassem o tempo e aguardassem algum outro participante finalizar a tarefa, para poder usar a conta de outro participante.

Conforme descrito anteriormente, a abordagem Multi-TaaS permite melhorar a eficiência e reduzir o esforço de migração entre os fornecedores de TaaS. No entanto, o problema de *vendor lock-in* no serviço de teste não pode ser completamente resolvido com a abordagem Multi-TaaS, principalmente devido às funcionalidades específicas de um determinado TaaS que estão ausentes em outros fornecedores de TaaS. No entanto a abordagem desenvolvida por este trabalho viabilizou uma nova estratégia para reduzir os impactos do *vendor lock-in* de maior e menor grau.

As empresas interessadas em consumir um serviço de teste na nuvem por meio da internet vão obter um menor custo de manutenção de migração com a abordagem Multi-TaaS. Este trabalho permite também orientar os engenheiros de software quanto aos riscos associados com *vendor lock-in* no TaaS. Esta tese poderá também ser usada como referência das melhores estratégias de pré-adoção e na tomada de decisão quanto à adoção do serviço de teste de forma consciente e com o objetivo principal de evitar ou reduzir os impactos ocasionados pelo problema de *vendor lock-in* durante e após o ciclo de desenvolvimento e teste de aplicações web.

5.5 Trabalhos Futuros

Um dos principais trabalhos futuros é investigar uma solução para a alcançar a interoperabilidade no contexto de TaaS de forma efetiva, o que por si só, representa um outro projeto de doutorado. A interoperabilidade no TaaS permite que os TaaS possam colaborar e se comunicar-se uns com os outros com o objetivo principal de agregar as funcionalidades, dispositivos e navegadores específicos de cada fornecedor de TaaS, englobando todas as tecnologias para os consumidores.

Um exemplo de colaboração e interoperabilidade entre fornecedores de TaaS seria a possibilidade de executar um teste do firefox pelo serviço de teste do Saucelabs e, em seguida, ou de forma simultânea executar outro teste no Chrome pelo Browserstack e sumarizar os resultados dos dois testes, por exemplo no Testingbot. De forma unificada, essa solução possibilitaria gerar

um relatório unificado de forma colaborativa entre os TaaS e de forma totalmente transparente para os consumidores, possibilitando inclusive otimizar e reduzir os custos para os consumidores. Porém atualmente essa solução é inviável, ou seja, é impossível executar um teste em um determinado TaaS e obter o relatório em outro TaaS, pois não existe um consenso entre os fornecedores de TaaS.

Uma proposta de mestrado seria uma avaliação comparativa da efetividade, eficiência e do esforço entre a criação ou automatização de testes na nuvem utilizando a arquitetura Multi-TaaS e com a abordagem convencional, para encontrar evidências de que a arquitetura desenvolvida permite facilitar também o processo de automatização de testes na nuvem. Outra abordagem muito interessante para um projeto de mestrado seria um estudo aprofundado para avaliar e comparar as estratégias de *billing* ou precificação entre os fornecedores de TaaS e propor novos modelos mais inteligentes para o sistema de cobrança, além do tempo de execução dos testes automatizados.

Um dos trabalhos futuros inclui publicar e disponibilizar a abordagem *open source* Multi-TaaS no repositório central do Maven. Pretende-se também transformar a biblioteca Java Multi-TaaS em uma API REST *open source*, sem custo para os consumidores e disponibilizá-la para toda a comunidade da indústria e pesquisadores interessados. Esse é o primeiro passo para a criação de um padrão aberto. A API REST ou Serviço RESTful funcionaria como um fornecedor TaaS intermediário para escrever, enviar, gerenciar e executar testes em outros TaaS, conforme ilustrado na Figura 54 pela letra c). Dessa forma, a própria API REST realizaria as transformações e traduções, unificando as respostas em uma API padrão entre as APIs REST proprietárias. Além disso, a própria especificação da biblioteca Multi-TaaS poderia contribuir para o desenvolvimento da API REST intermediária e na extração e implementação de um protocolo ou padrão aberto. Uma das principais vantagens desse tipo de solução seria ampliar e possibilitar a escrita dos testes automatizados nas outras linguagens de programação além da linguagem Java. A abordagem Multi-TaaS seria independente de linguagem de programação e funcionaria sobre o protocolo HTTP.

Já está em andamento também a elaboração de métricas para quantificar a portabilidade e a interoperabilidade no contexto de TaaS com base nas normas de qualidade de software (ISO25010, 2011) e de acordo com os parâmetros de número de testes automatizados, capacidades (navegadores, sistemas operacionais e dispositivos), funcionalidades específicas, tempo e esforço de migração, entre outros.

A abordagem Multi-TaaS apresentada nesta tese pode ser estendida para os TaaS não funcionais, pois a ideia conceitual seria exatamente a mesma. Além disso, a falta das características da arquitetura *multi-tenancy* em relação à aplicação SInAU dificultaram também a execução de testes paralelos no TaaS. Apesar da arquitetura Multi-TaaS ter sido desenvolvida com base nos testes sequenciais, a abordagem pode ser evoluída para executar também os testes paralelos no ambiente de nuvem. Porém, a complexidade do paralelismos reside, principalmente,

em dois aspectos importantes: o primeiro é referente à economia de tempo, ao executar 100 testes sequenciais, o tempo poderia ser reduzido teoricamente pela metade com a paralelização dos testes automatizados. O segundo aspecto é referente à cobertura de plataformas, em que o objetivo não é necessariamente a economia de tempo, mas executar o mesmo teste em diferentes combinações de plataformas (sistemas operacionais, navegadores e dispositivos) no TaaS, por exemplo.

Finalmente, as soluções correspondentes ao problema de *vendor lock-in* de maior grau, na etapa de configuração do ambiente de teste referentes às configurações de máquinas virtuais (VMs) e paralelismo dos testes automatizados no ambiente de nuvem, questões ainda não resolvidas, serão alvo de futuras investigações.

REFERÊNCIAS

- AKERELE, O.; RAMACHANDRAN, M.; DIXON, M. Testing in the cloud: Strategies, risks and benefits. In: MAHMOOD, Z.; SAEED, S. (Ed.). **Software Engineering Frameworks for the Cloud Computing Paradigm**. Springer London, 2013, (Computer Communications and Networks). p. 165–185. Disponível em: <http://dx.doi.org/10.1007/978-1-4471-5031-2_8>.
- ALEB, N.; KECHID, S. Path coverage testing in the cloud. In: **Communications and Information Technology (ICCIT), 2012 International Conference on**. [S.l.: s.n.], 2012. p. 118–123.
- ALI, A.; BADR, N. Performance testing as a service for web applications. In: **2015 IEEE Seventh International Conference on Intelligent Computing and Information Systems (ICI-CIS)**. [S.l.: s.n.], 2015. p. 356–361.
- ALLISON, M.; TURNER, S.; ALLEN, A. A. Towards interpreting models to orchestrate iaas multi-cloud infrastructures. In: **Computer Science Education (ICCSE), 2015 10th International Conference on**. [S.l.: s.n.], 2015. p. 80–85.
- ALTERNATIVETO. **Alternatives to Sauce Labs [On-line]**. 2017. Último acesso: 07/03/2017. Disponível em: <<https://alternativeto.net/software/sauce-labs/>>.
- AMMANN, P.; OFFUTT, J. **Introduction to Software Testing**. 1. ed. New York, NY, USA: Cambridge University Press, 2008.
- ANASTASI, G. F.; CARLINI, E.; COPPOLA, M.; DAZZI, P.; DISTEFANO, M. An ovf toolkit supporting inter-cloud application splitting. In: **Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on**. [S.l.: s.n.], 2014. p. 96–101.
- ARMBRUST, M.; FOX, A.; GRIFFITH, R.; JOSEPH, A. D.; KATZ, R. H.; KONWINSKI, A.; LEE, G.; PATTERSON, D. A.; RABKIN, A.; STOICA, I.; ZAHARIA, M. **Above the Clouds: A Berkeley View of Cloud Computing**. [S.l.], 2009. Disponível em: <<http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html>>.
- ARMBRUST, M.; FOX, A.; GRIFFITH, R.; JOSEPH, A. D.; KATZ, R.; KONWINSKI, A.; LEE, G.; PATTERSON, D.; RABKIN, A.; STOICA, I.; ZAHARIA, M. A view of cloud computing. **Commun. ACM**, ACM, New York, NY, USA, v. 53, n. 4, p. 50–58, abr. 2010. Disponível em: <<http://doi.acm.org/10.1145/1721654.1721672>>.
- BADGER, L.; BERNSTEIN, D.; BOHN, R.; VAULX, F. de; HOGAN, M.; IORGA, M.; MAO, J.; MESSINA, J.; MILLS, K.; SIMMON, E.; SOKOL, A.; TONG, J.; WHITESIDE, F.; LEAF, D. **US Government Cloud Computing Technology Roadmap - High-Priority Requirements to Further USG Agency Cloud Computing Adoption, Volume I**. 2014. Special Publication 500-293. Disponível em: <<http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.500-293.pdf>>.
- BAI, X.; LI, M.; CHEN, B.; TSAI, W.-T.; GAO, J. Cloud testing tools. In: **2011 IEEE 6th International Symposium on Service Oriented System Engineering (SOSE)**. [S.l.: s.n.], 2011. p. 1–12.

BALARAJU, A. Testing Cloud and Testing using Cloud. **Product Manager - Testing Practice**, 2011. Disponível em: <<http://www.sonata-software.com/?q=resources/testing-cloud-and-testing-using-cloud>>.

BASILI, V. R. The role of experimentation in software engineering: past, current, and future. In: **Proceedings of IEEE 18th International Conference on Software Engineering**. [S.l.: s.n.], 1996. p. 442–449.

BASILI, V. R.; SELBY, R. W.; HUTCHENS, D. H. Experimentation in software engineering. **IEEE Trans. Softw. Eng.**, IEEE Press, Piscataway, NJ, USA, v. 12, n. 7, p. 733–743, jul. 1986. Disponível em: <<http://dl.acm.org/citation.cfm?id=9775.9777>>.

BEAUFTRAGTEN, R. **V-Modell**. 1997.

BERMBACH, D.; KLEMS, M.; TAI, S.; MENZEL, M. MetaStorage: A Federated Cloud Storage System to Manage Consistency-Latency Tradeoffs. In: **2011 IEEE International Conference on Cloud Computing (CLOUD)**. [S.l.: s.n.], 2011. p. 452–459.

BERMBACH, D.; KURZE, T.; TAI, S. Cloud federation: Effects of federated compute resources on quality of service and cost*. In: **Cloud Engineering (IC2E), 2013 IEEE International Conference on**. [S.l.: s.n.], 2013. p. 31–37.

BERNSTEIN, D.; LUDVIGSON, E.; SANKAR, K.; DIAMOND, S.; MORROW, M. Blueprint for the Intercloud - Protocols and Formats for Cloud Computing Interoperability. In: **Fourth International Conference on Internet and Web Applications and Services, 2009. ICIW '09**. [S.l.: s.n.], 2009. p. 328–336.

BEZEMER, C.-P.; ZAIDMAN, A. Multi-tenant SaaS Applications: Maintenance Dream or Nightmare? In: **Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE)**. New York, NY, USA: ACM, 2010. (IWPSE-EVOL '10, v. 1), p. 88–92. Disponível em: <<http://doi.acm.org/10.1145/1862372.1862393>>.

BINDER, R. V. **Testing Object-oriented Systems: Models, Patterns, and Tools**. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

BLOKLAND, K.; MENDERINK, J.; POL, M. **Testing Cloud Services: How to Test SaaS, PaaS & IaaS**. [S.l.]: Rocky Nook, 2013.

BOEHM, B. W.; CLARK; HOROWITZ; BROWN; REIFER; CHULANI; MADACHY, R.; STEECE, B. **Software Cost Estimation with Cocomo II with Cdrom**. 1st. ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2000.

BOONE, J. H. N.; BOONE, D. A. Analyzing likert data. **Journal of Extension - Sharing Knowledge, Enriching Extension (Tools of the Trade (2TOT2))**, v. 50, n. 2, April 2012. Disponível em: <<https://www.joe.org/joe/2012april/tt2.php>>.

BROWSERSTACK. “**Docs: Local Testing Internals**”. [On-line]. 2014. Accessed: 27-Mar-2017. Disponível em: <<http://www.browserstack.com/local-testing-internals>>.

_____. “**Docs: Local Testing Internals**”. [On-line]. 2016. Accessed: 21-Jan-2016. Disponível em: <<http://www.browserstack.com/local-testing-internals>>.

_____. **Documentação do Browserstack [On-line]**. 2017. Último acesso: 10/05/2017. Disponível em: <<https://www.browserstack.com/automate/junit>>.

BUDD, T. A. Mutation Analysis: Ideas, Example, Problems and Prospects. In: _____. Amsterdam: North-Holand Publishing Company, 1981. cap. Computer Program Testing: Proceedings of the Summer School on Computer Program Testing held at SOGESTA, Urbino, Italy, p. 129–148.

BUYYA, R.; RANJAN, R.; CALHEIROS, R. N. InterCloud: Utility-oriented Federation of Cloud Computing Environments for Scaling of Application Services. In: **Proceedings of the 10th International Conference on Algorithms and Architectures for Parallel Processing - Volume Part I**. Berlin, Heidelberg: Springer-Verlag, 2010. (ICA3PP'10, v. 1), p. 13–31. Disponível em: <http://dx.doi.org/10.1007/978-3-642-13119-6_2>.

CANDEA, G.; BUCUR, S.; ZAMFIR, C. Automated Software Testing as a Service (TaaS). In: **Proceedings of the 1st ACM Symposium on Cloud Computing**. [S.l.: s.n.], 2010.

CHANA, I.; CHAWLA, P. Testing Perspectives for Cloud-Based Applications. In: MAHMOOD, Z.; SAEED, S. (Ed.). **Software Engineering Frameworks for the Cloud Computing Paradigm**. Springer London, 2013, (Computer Communications and Networks). p. 145–164. Disponível em: <http://dx.doi.org/10.1007/978-1-4471-5031-2_7>.

CHEN, Y.; LI, X.; CHEN, F. Overview and analysis of cloud computing research and application. In: **2011 International Conference on E -Business and E -Government (ICEE)**. [S.l.: s.n.], 2011. p. 1–4.

CIORTEA, L.; ZAMFIR, C.; BUCUR, S.; CHIPOUNOV, V.; CANDEA, G. Cloud9: A Software Testing Service. **SIGOPS Oper. Syst. Rev.**, ACM, New York, NY, USA, v. 43, n. 4, p. 5–10, jan. 2010. Disponível em: <<http://doi.acm.org/10.1145/1713254.1713257>>.

CLOUD Computing Use Cases Whitepaper. 2009. Disponível em: <http://www.cloud-council.org/Cloud_Computing_Use_Cases_Whitepaper-4_0.pdf>.

CUNHA, P. R. D.; MELO, P.; SILVA, C. F. D. Avoiding Lock-In: Timely Reconfiguration of a Virtual Cloud Platform on Top of Multiple PaaS and IaaS Providers. In: **2013 IEEE Sixth International Conference on Cloud Computing (CLOUD)**. [S.l.: s.n.], 2013. p. 970–971.

DEGRAVE, K. “Cross-browser Testing Using BrowserStack”. [On-line]. 2013. Accessed: 21-Jul-2014. Disponível em: <<http://java.dzone.com/articles/cross-browser-testing-using>>.

DELAMARO, M.; MALDONADO, J.; JINO, M. Introdução ao teste de software. In: . 2. ed. [S.l.]: Elsevier, 2016.

DELAMARO M., M. E.; CHAIN, M. L.; VINCENZI, A. M. R. Técnicas e ferramentas de teste de software. In: PUC-RIO, R. de J. E. (Ed.). [S.l.]: Atualizações em Informática 2010 (JAI 2010), 2010. p. 55–110.

DELANGE, J.; MCHALE, J.; HUDAK, J.; NICHOLS, W.; NAM, M.-Y. **Evaluating and Mitigating the Impact of Complexity in Software Models**. Pittsburgh, PA, 2015. Disponível em: <<http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=448083>>.

DEMILLO, R. **Software testing and evaluation**. [S.l.]: Benjamin/Cummings Pub. Co., 1978.

- DILLON, T.; WU, C.; CHANG, E. Cloud Computing: Issues and Challenges. In: **2010 24th IEEE International Conference on Advanced Information Networking and Applications (AINA)**. [S.l.: s.n.], 2010. p. 27–33.
- DURAO, F.; CARVALHO, J. F.; FONSEKA, A.; GARCIA, V. C. A systematic review on cloud computing. **J. Supercomput.**, Kluwer Academic Publishers, Hingham, MA, USA, v. 68, n. 3, p. 1321–1346, jun. 2014. Disponível em: <<http://dx.doi.org/10.1007/s11227-014-1089-x>>.
- EDMONDS, A.; METSCH, T.; PAPASPYROU, A.; RICHARDSON, A. Toward an open cloud standard. **IEEE Internet Computing**, v. 16, n. 4, p. 15–25, July 2012.
- FALESSI, D.; JURISTO, N.; WOHLIN, C.; TURHAN, B.; MÜNCH, J.; JEDLITSCHKA, A.; OIVO, M. Empirical software engineering experts on the use of students and professionals in experiments. **Empirical Software Engineering**, p. 1–38, 2017. Disponível em: <<http://dx.doi.org/10.1007/s10664-017-9523-3>>.
- FERRY, N.; ROSSINI, A.; CHAUVEL, F.; MORIN, B.; SOLBERG, A. Towards Model-Driven Provisioning, Deployment, Monitoring, and Adaptation of Multi-cloud Systems. In: **Proceedings of the 2013 IEEE Sixth International Conference on Cloud Computing**. Washington, DC, USA: IEEE Computer Society, 2013. (CLOUD '13, v. 1), p. 887–894. Disponível em: <<http://dx.doi.org/10.1109/CLOUD.2013.133>>.
- FERRY, N.; SONG, H.; ROSSINI, A.; CHAUVEL, F.; SOLBERG, A. Cloud mf: Applying mde to tame the complexity of managing multi-cloud applications. In: **Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing**. Washington, DC, USA: IEEE Computer Society, 2014. (UCC '14), p. 269–277. Disponível em: <<http://dx.doi.org/10.1109/UCC.2014.36>>.
- FORUM, O. G. **“Open Cloud Computing Interface (OCCI).”** [On-line]. 2014. Último acesso: 26/06/2014. Disponível em: <<http://occi-wg.org/>>
- FUSCO, C. **“Este será o seu computador”.** [On-line]. 2009. Último acesso: 20/07/2014. Disponível em: <<http://exame.abril.com.br/revista-exame/edicoes/0941/noticias/este-sera-seucomputador-449751>>.
- GAISBAUER, S.; KIRSCHNICK, J.; EDWARDS, N.; ROLIA, J. VATS: Virtualized-Aware Automated Test Service. In: **Fifth International Conference on Quantitative Evaluation of Systems, 2008. QEST '08**. [S.l.: s.n.], 2008. p. 93–102.
- GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. **Design Patterns: Elements of Reusable Object-oriented Software**. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995.
- GANDHI, P.; HAUGEN, N.; HILL, M.; WATT, R. Creating a living specification using fit documents. In: **Agile Conference, 2005. Proceedings**. [S.l.: s.n.], 2005. p. 253–258.
- GAO, J.; BAI, X.; TSAI, W.-T. Cloud Testing- Issues, Challenges, Needs and Practice. In: **Software Engineering : An International Journal (SEIJ)**. September 2011, 2011. v. 1, n. 1, p. 9–23. Disponível em: <<http://seij.dce.edu/Paper%201.pdf>>.
- GAO, J.; BAI, X.; TSAI, W.-T.; UEHARA, T. Testing as a Service (TaaS) on Clouds. In: **Service Oriented System Engineering (SOSE), 2013 IEEE 7th International Symposium on**. [S.l.: s.n.], 2013. p. 212–223.

GAO, J.; BAI, X.; TSAI, W. T.; UEHARA, T. Mobile application testing: A tutorial. **Computer**, v. 47, n. 2, p. 46–55, Feb 2014.

GOCHENOUR, P. **Documentação do Saucelabs [On-line]**. 2016. Último acesso: 07/04/2017. Disponível em: <<https://wiki.saucelabs.com/display/DOCS/Java+Test+Setup+Example>>.

GOGGRID. **Complex Infrastructure Made Easy - Scaling Your Internet Business [On-line]**. 2010. Último acesso: 26/06/2014. Disponível em: <https://www.cloudsleuth.net/c/document_library/get_file?uuid=535be2b1-0ff3-47f8-bdb7-650245975a68&groupId=100135>.

GOVINDARAJAN, A.; LAKSHMANAN. Overview of Cloud miscs. In: ANTONOPOULOS, N.; GILLAM, L. (Ed.). **Cloud Computing**. Springer London, 2010, (Computer Communications and Networks). p. 77–89. Disponível em: <http://dx.doi.org/10.1007/978-1-84996-241-4_5>.

GOYAL, G. **Selenium Components**. 2013. Disponível em: <<http://seleniumview.blogspot.com.br/2013/10/selenium-components-and-their.html>>.

GRAHAM, D.; VEENENDAAL, E. V.; EVANS, I.; BLACK, R. **Foundations of Software Testing: ISTQB Certification**. [S.l.]: Intl Thomson Business Pr, 2008.

GROZEV, N.; BUYYA, R. Inter-cloud architectures and application brokering: Taxonomy and survey. **Softw. Pract. Exper.**, John Wiley & Sons, Inc., New York, NY, USA, v. 44, n. 3, p. 369–390, mar. 2014. Disponível em: <<http://dx.doi.org/10.1002/spe.2168>>.

_____. Multi-cloud provisioning and load distribution for three-tier applications. **ACM Trans. Auton. Adapt. Syst.**, ACM, New York, NY, USA, v. 9, n. 3, p. 13:1–13:21, out. 2014. Disponível em: <<http://doi.acm.org/10.1145/2662112>>.

HAMBLING, B.; GOATHEM, P. van. **User Acceptance Testing: A step-by-step Guide**. [S.l.]: BCS Learning & Development Limited, 2013.

HAMDAQA, M.; TAHVILDARI, L. Prison break: A generic schema matching solution to the cloud vendor lock-in problem. In: **Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA), 2014 IEEE 8th International Symposium on the**. [S.l.: s.n.], 2014. p. 37–46.

HANSSEN, G.; HAUGSET, B. Automated Acceptance Testing Using Fit. In: **42nd Hawaii International Conference on System Sciences, 2009. HICSS '09**. [S.l.: s.n.], 2009. p. 1–8.

HARROLD, M. J.; JONES, J. A.; LI, T.; LIANG, D.; ORSO, A.; PENNINGS, M.; SINHA, S.; SPOON, S. A.; GUJARATHI, A. Regression Test Selection for Java Software. **SIGPLAN Not.**, ACM, New York, NY, USA, v. 36, n. 11, p. 312–326, out. 2001. Disponível em: <<http://doi.acm.org/10.1145/504311.504305>>.

HARSH, P.; DUDOUET, F.; CASCELLA, R. G.; JEGOU, Y.; MORIN, C. Using open standards for interoperability issues, solutions, and challenges facing cloud computing. In: **2012 8th international conference on network and service management (cnsm) and 2012 workshop on systems virtualization management (svm)**. [S.l.: s.n.], 2012. p. 435–440.

HAUGSET, B.; HANSSEN, G. K. Automated acceptance testing: A literature review and an industrial case study. In: **Agile 2008 Conference**. [S.l.: s.n.], 2008. p. 27–38.

- HERBST, N. R.; KOUNEV, S.; REUSSNER, R. H. Elasticity in cloud computing: What it is, and what it is not. In: **10th International Conference on Autonomic Computing, ICAC'13, San Jose, CA, USA, June 26-28, 2013**. [s.n.], 2013. p. 23–27. Disponível em: <<https://www.usenix.org/conference/icac13/technical-sessions/presentation/herbst>>.
- HOFMANN, P.; WOODS, D. Cloud Computing: The Limits of Public Clouds for Business Applications. **Internet Computing, IEEE**, v. 14, n. 6, p. 90–93, Nov 2010.
- HOLMES, A.; KELLOGG, M. Automating functional tests using selenium. In: **Agile Conference, 2006**. [S.l.: s.n.], 2006. p. 6 pp.–275.
- HÖST, M.; REGNELL, B.; WOHLIN, C. Using students as subjects—a comparative study of students and professionals in lead-time impact assessment. **Empirical Software Engineering**, v. 5, n. 3, p. 201–214, 2000. Disponível em: <<http://dx.doi.org/10.1023/A:1026586415054>>.
- HSIEH, S. J.; YUAN, S. M.; LUO, G. H.; CHEN, H. W. A flexible public cloud based testing service for heterogeneous testing targets. In: **Network Operations and Management Symposium (APNOMS), 2014 16th Asia-Pacific**. [S.l.: s.n.], 2014. p. 1–3.
- HU, K.; LEI, L.; TSAI, W.-T. Multi-tenant verification-as-a-service (vaas) in a cloud. **Simulation Modelling Practice and Theory**, v. 60, p. 122 – 143, 2016. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S1569190X15001227>>.
- HUANG, S.; LI, Z. jie; LIU, Y.; ZHU, J. Regression Testing as a Service. In: **SRII Global Conference (SRII), 2011 Annual**. [S.l.: s.n.], 2011. p. 315–324.
- HUMBLE, J.; FARLEY, D. **Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation**. Pearson Education, 2010. (Addison-Wesley Signature Series (Fowler)). Disponível em: <<https://books.google.com.br/books?id=6ADDuzere-YC>>.
- IEEE. **IEEE misc for software verification and validation plans**. [S.l.], 1986. Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=26585>.
- _____. **IEEE misc glossary of Software Engineering terminology**. 1990. Padrão 620.12.
- _____. **IEEE misc for Software and System Test Documentation 829-2008**. 2008. Disponível em: <<http://miscs.ieee.org/findstds/misc/829-2008.html>>.
- İNÇKI, K.; ARI, I.; SOZER., H. A Survey of Software Testing in the Cloud. In: **2012 IEEE Sixth International Conference on Software Security and Reliability Companion (SERE-C)**. [S.l.: s.n.], 2012. p. 18–23.
- ISO25010. **NBR ISO/IEC 25010 Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE)**. 2011. System and software quality models.
- ISO25066. **ISO/IEC FDIS 25066:2016, Systems and software engineering – Software product Quality Requirements and Evaluation SQuaRE – Common Industry Format CIF for usability: Evaluation report**. [S.l.], 2016.
- JAMSHIDI, P.; AHMAD, A.; PAHL, C. Cloud migration research: A systematic review. **IEEE Trans. Cloud Comput.**, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 1, n. 2, p. 142–157, jul. 2013. Disponível em: <<http://dx.doi.org/10.1109/TCC.2013.10>>.

JORGENSEN, M. **“What We Do and Don’t Know about Software Development Effort Estimation”**. 2014. Disponível em: <<https://www.infoq.com/articles/software-development-effort-estimation>>.

JURISTO, N.; MORENO, A. M. **Basics of Software Engineering Experimentation**. 1st. ed. [S.l.]: Springer Publishing Company, Incorporated, 2010.

KARATAS, F.; BOURIMI, M.; BARTH, T.; KESDOGAN, D.; GIMENEZ, R.; SCHWITTEK, W.; PLANAGUMA, M. Towards secure and at-runtime tailorable customer-driven public cloud deployment. In: **Pervasive Computing and Communications Workshops (PERCOM Workshops), 2012 IEEE International Conference on**. [S.l.: s.n.], 2012. p. 124–130.

KATHERINE, A.; ALAGARSAMY, K. Article: Software Testing in Cloud Platform: A Survey. **International Journal of Computer Applications**, v. 46, n. 6, p. 21–25, May 2012. Published by Foundation of Computer Science, New York, USA.

KIM, W.; KIM, S. D.; LEE, E.; LEE, S. Adoption Issues for Cloud Computing. In: **Proceedings of the 11th International Conference on Information Integration and Web-based Applications & Services**. New York, NY, USA: ACM, 2009. (iiWAS ’09, v. 1), p. 3–6. Disponível em: <<http://doi.acm.org/10.1145/1806338.1806341>>.

KIRAN, M.; FRIESEN, A.; SIMONS, A. J. H.; SCHWACH, W. K. R. Model-based testing in cloud brokerage scenarios. In: _____. **Service-Oriented Computing – ICSOC 2013 Workshops: CCSA, CSB, PASCEB, SWESE, WESOA, and PhD Symposium, Berlin, Germany, December 2-5, 2013. Revised Selected Papers**. Cham: Springer International Publishing, 2014. p. 192–208. Disponível em: <http://dx.doi.org/10.1007/978-3-319-06859-6_17>.

KITCHENHAM, B.; CHARTERS, S. **Guidelines for performing Systematic Literature Reviews in Software Engineering**. [S.l.], 2007.

KOLB, S.; LENHARD, J.; WIRTZ, G. Application migration effort in the cloud - the case of cloud platforms. In: **2015 IEEE 8th International Conference on Cloud Computing**. [S.l.: s.n.], 2015. p. 41–48.

KOSMATOV, N.; WILLIAMS, N.; BOTELLA, B.; ROGER, M. Structural unit testing as a service with pathcrawler-online.com. In: **Service Oriented System Engineering (SOSE), 2013 IEEE 7th International Symposium on**. [S.l.: s.n.], 2013. p. 435–440.

KRISHNA, R.; JAYAKRISHNAN, R. Impact of Cloud Services on Software Development Life Cycle. In: MAHMOOD, Z.; SAEED, S. (Ed.). **Software Engineering Frameworks for the Cloud Computing Paradigm**. Springer London, 2013, (Computer Communications and Networks). p. 79–99. Disponível em: <http://dx.doi.org/10.1007/978-1-4471-5031-2_4>.

KRISHNAMOORTHY, V. **Testing in The Cloud Using SauceLabs [On-line]**. 2012. Último acesso: 19/07/2014. Disponível em: <<http://www.sitepoint.com/testing-in-the-cloud-using-saucelabs/>>.

KROPP, M.; SCHWAIGER, W. Reverse Generation and Refactoring of Fit Acceptance Tests for Legacy Code. In: **Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications**. New York, NY, USA: ACM, 2009. (OOPSLA ’09, v. 1), p. 659–664. Disponível em: <<http://doi.acm.org/10.1145/1639950.1639961>>.

- LANA, C. A.; SOUZA, N. M.; DELAMARO, M. E.; NAKAGAWA, E. Y.; OQUENDO, F.; MALDONADO, J. C. Systems of systems (sos) development: Initiatives, trends and challenges. In: **XLII Conferencia Latinoamericana de Informatica**. [S.l.: s.n.], 2016.
- LEUNG, H.; WONG, P. A study of user acceptance tests. **Software Quality Journal**, Kluwer Academic Publishers, v. 6, n. 2, p. 137–149, 1997. Disponível em: <<http://dx.doi.org/10.1023/A%3A1018503800709>>.
- LEWIS, G. “**The Role of miscs in Cloud-Computing Interoperability**”. [S.l.], 2012. Disponível em: <<http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=28017>>.
- LI, R.; ABENDROTH, D.; LIN, X.; GUO, Y.; BAEK, H.-W.; EIDE, E.; RICCI, R.; MERWE, J. Van der. Potassium: Penetration testing as a service. In: **Proceedings of the Sixth ACM Symposium on Cloud Computing**. New York, NY, USA: ACM, 2015. (SoCC '15), p. 30–42. Disponível em: <<http://doi.acm.org/10.1145/2806777.2806935>>.
- LOUTAS, N.; KAMATERI, E.; BOSI, F.; TARABANIS, K. Cloud Computing Interoperability: The State of Play. In: **2011 IEEE Third International Conference on Cloud Computing Technology and Science (CloudCom)**. [S.l.: s.n.], 2011. p. 752–757.
- LOUTAS, N.; PERISTERAS, V.; BOURAS, T.; KAMATERI, E.; ZEGINIS, D.; TARABANIS, K. Towards a Reference Architecture for Semantically Interoperable Clouds. In: **2010 IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom)**. [S.l.: s.n.], 2010. p. 143–150.
- MALDONADO, J. C. **Crterios potenciais usos: uma contribuio ao teste estrutural de Software**. Tese (Doutorado) — DCA/FEE/UNICAMP, Campinas, SP, 1991.
- MARINOS, A.; BRISCOE, G. Community Cloud Computing. In: **Proceedings of the 1st International Conference on Cloud Computing**. Berlin, Heidelberg: Springer-Verlag, 2009. (Cloud-Com '09, v. 1), p. 472–484. Disponível em: <http://dx.doi.org/10.1007/978-3-642-10665-1_43>.
- MARKETING, S. **Key findings from the World Quality Report: Increased focus on quality assurance, performance testing [On-line]**. 2014. ltimo acesso: 07/03/2017. Disponível em: <<https://www.soasta.com/blog/key-findings-world-quality-report-increased-focus-quality-assurance-performance-testing/>>.
- MARTINO, B. D.; CRETELLA, G.; ESPOSITO, A.; SPERANDEO, R. G. Semantic representation of cloud services: A case study for microsoft windows azure. In: **Intelligent Networking and Collaborative Systems (INCoS), 2014 International Conference on**. [S.l.: s.n.], 2014. p. 647–652.
- MELL, P.; GRANCE, T. **The NIST Definition of Cloud Computing**. <http://csrc.nist.gov/groups/SNS/cloud-computing/cloud-def-v15.doc>: [s.n.], 2009.
- MICHAEL, C. C.; MCGRAW, G.; SCHATZ, M. A. Generating Software Test Data by Evolution. **IEEE Trans. Softw. Eng.**, IEEE Press, Piscataway, NJ, USA, v. 27, n. 12, p. 1085–1110, dez. 2001. Disponível em: <<http://dx.doi.org/10.1109/32.988709>>.
- MILLER, R.; COLLINS, C. Acceptance testing. In: **Proceedings of the XP Universe Conference**. [S.l.: s.n.], 2001.

MIRANDA, J.; GUILLÉN, J.; MURILLO, J. M.; CANAL, C. Enough about standardization, let's build cloud applications. In: **Proceedings of the WICSA/ECSA 2012 Companion Volume**. New York, NY, USA: ACM, 2012. (WICSA/ECSA '12), p. 74–77. Disponível em: <<http://doi.acm.org/10.1145/2361999.2362011>>.

MIRANDA, J.; GUILLÉN, J.; MURILLO, J. M.; CANAL, C. Assisting cloud service migration using software adaptation techniques. In: **2013 IEEE Sixth International Conference on Cloud Computing**. [S.l.: s.n.], 2013. p. 573–580.

MUNCH, J. Cloud - based software engineering. In: **Proceedings of the Seminar**. University of Helsinki, Faculty of Science, Department of Computer Science., 2013. Disponível em: <https://tuhat.halvi.helsinki.fi/portal/files/28513674/cbse13_proceedings.pdf>.

MYERS, G. J.; SANDLER, C.; BADGETT, T. **The Art of Software Testing**. [S.l.]: John Wiley & Sons, 2004.

NGUYEN, V.; BOEHM, B.; DANPHITSANUPHAN, P. Assessing and estimating corrective, enhance, and reductive maintenance tasks: A controlled experiment. In: **2009 16th Asia-Pacific Software Engineering Conference**. [S.l.: s.n.], 2009. p. 381–388.

NOGUEIRA, E.; MOREIRA, A.; LUCRÉDIO, D.; GARCIA, V.; FORTES, R. Issues on developing interoperable cloud applications: definitions, concepts, approaches, requirements, characteristics and evaluation models. **Journal of Software Engineering Research and Development**, v. 4, n. 1, p. 7, 2016. Disponível em: <<http://dx.doi.org/10.1186/s40411-016-0033-6>>.

NOLL, J.; PIROTTA, E.; SOLIS, C.; WANG, X. Automated Acceptance Testing vs. Quality: A Case Study of an Open Source Project. In: **The IRISH Software Engineering Research Centre**. [S.l.: s.n.], 2011.

OLIVEIRA, J.; LOPES, D.; ABDELOUAHAB, Z.; CLARO, D.; HAMMOUDI, S. Model driven testing for cloud computing. In: _____. **Innovations and Advances in Computing, Informatics, Systems Sciences, Networking and Engineering**. Cham: Springer International Publishing, 2015. p. 297–304. Disponível em: <http://dx.doi.org/10.1007/978-3-319-06773-5_40>.

OLIVEIRA, R. R. d.; MARTINS, R. M.; SIMAO, A. d. S. Impact of the vendor lock-in problem on testing as a service (taas). In: **2017 IEEE International Conference on Cloud Engineering (IC2E)**. [S.l.: s.n.], 2017. p. 190–196.

OPARA-MARTINS, J.; SAHANDI, R.; TIAN, F. Critical review of vendor lock-in and its impact on adoption of cloud computing. In: **Information Society (i-Society), 2014 International Conference on**. [S.l.: s.n.], 2014. p. 92–97.

ORSO, A.; SILVA, S. Open issues and research directions in object oriented testing. In: **In Proceedings of the 4th International Conference on "Achieving Quality in Software: Software Quality in the Communication Society" (AQUIS'98)**. [S.l.: s.n.], 1998.

PAHL, C.; XIONG, H.; WALSHE, R. A comparison of on-premise to cloud migration approaches. In: **Service-Oriented and Cloud Computing - Second European Conference, ESOC 2013, Málaga, Spain, September 11-13, 2013. Proceedings**. [s.n.], 2013. p. 212–226. Disponível em: <http://dx.doi.org/10.1007/978-3-642-40651-5_18>.

- PETCU, D. Portability and Interoperability between Clouds: Challenges and Case Study. In: ABRAMOWICZ, W.; LLORENTE, I.; SURRIDGE, M.; ZISMAN, A.; VAYSSIÈRE, J. (Ed.). **Towards a Service-Based Internet**. Springer Berlin Heidelberg, 2011, (Lecture Notes in Computer Science, v. 6994). p. 62–74. Disponível em: <http://dx.doi.org/10.1007/978-3-642-24755-2_6>.
- PETCU, D.; MACARIU, G.; PANICA, S.; CRICIUN, C. Portable Cloud applications-From Theory to Practice. **Future Gener. Comput. Syst.**, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 29, n. 6, p. 1417–1430, ago. 2013. Disponível em: <<http://dx.doi.org/10.1016/j.future.2012.01.009>>.
- PORTABLEAPPS.COM. **About PortableApps.com**. 2017. Disponível em: <<https://portableapps.com/about>>.
- PRAKASH, N. “Cloud testing: attracting demand”. 2010. Disponível em: <<http://www.expresscomputeronline.com/20100201/trend01.shtml>>.
- PRESSMAN, R. **Software Engineering: A Practitioner’s Approach**. 7. ed. New York, NY, USA: McGraw-Hill, Inc., 2009.
- PRIYANKA; CHANA, I.; RANA, A. Empirical Evaluation of Cloud-based Testing Techniques: A Systematic Review. **SIGSOFT Softw. Eng. Notes**, ACM, New York, NY, USA, v. 37, n. 3, p. 1–9, maio 2012. Disponível em: <<http://doi.acm.org/10.1145/180921.2180938>>.
- RAFIQUE, A.; WALRAVEN, S.; LAGAISSÉ, B.; DESAIR, T.; JOOSEN, W. Towards portability and interoperability support in middleware for hybrid clouds. In: **Computer Communications Workshops (INFOCOM WKSHPS), 2014 IEEE Conference on**. [S.l.: s.n.], 2014. p. 7–12.
- RANABAHU, A.; SHETH, A. Semantics Centric Solutions for Application and Data Portability in Cloud Computing. In: **2010 IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom)**. [S.l.: s.n.], 2010. p. 234–241.
- RAZAVIAN, S.; KHANI, H.; YAZDANI, N.; GHASSEMI, F. An analysis of vendor lock-in problem in cloud storage. In: **2013 3th International eConference on Computer and Knowledge Engineering (ICCKE)**. [S.l.: s.n.], 2013. p. 331–335.
- RICCA, F.; TORCHIANO, M.; PENTA, M. D.; CECCATO, M.; TONELLA, P. Using Acceptance Tests As a Support for Clarifying Requirements: A Series of Experiments. **Inf. Softw. Technol.**, Butterworth-Heinemann, Newton, MA, USA, v. 51, n. 2, p. 270–283, fev. 2009. Disponível em: <<http://dx.doi.org/10.1016/j.infsof.2008.01.007>>.
- RIMAL, B.; CHOI, E.; LUMB, I. A Taxonomy and Survey of Cloud Computing Systems. In: **Fifth International Joint Conference on INC, IMS and IDC, 2009. NCM ’09**. [S.l.: s.n.], 2009. p. 44–51.
- RIUNGU-KALLIOSAARI, L. **Empirical study on the adoption, use and effects of cloud-based testing**. Tese (Doutorado) — Lappeenranta University of Technology, 2014. Disponível em: <<http://www.doria.fi/bitstream/handle/10024/96973/isbn9789522655769.pdf?sequence=2>>.
- ROCHWERGER, B.; BREITGAND, D.; LEVY, E.; GALIS, A.; NAGIN, K.; LLORENTE, I.; MONTERO, R.; WOLFSTHAL, Y.; ELMROTH, E.; CACERES, J.; BEN-YEHUDA, M.; EMMERICH, W.; GALAN, F. The Reservoir model and architecture for open federated cloud computing. **IBM Journal of Research and Development**, v. 53, n. 4, p. 4:1–4:11, July 2009.

RODERO-MERINO, L.; VAQUERO, L. M.; GIL, V.; GALÁN, F.; FONTÁN, J.; MONTERO, R. S.; LLORENTE, I. M. From infrastructure delivery to service management in clouds. **Future Generation Computer Systems**, v. 26, n. 8, p. 1226 – 1240, 2010. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S0167739X10000294>>.

ROGERS, R. Acceptance Testing vs. Unit Testing: A Developer's Perspective. In: ZANNIER, C.; ERDOGMUS, H.; LINDSTROM, L. (Ed.). **Extreme Programming and Agile Methods - XP/Agile Universe 2004**. Springer Berlin Heidelberg, 2004, (Lecture Notes in Computer Science, v. 3134). p. 22–31. Disponível em: <http://dx.doi.org/10.1007/978-3-540-27777-4_3>.

SAMS, P. **Selenium Essentials**. [S.l.]: Packt Publishing, 2015. (Community experience distilled).

SATZGER, B.; HUMMER, W.; INZINGER, C.; LEITNER, P.; DUSTDAR, S. Winds of Change: From Vendor Lock-In to the Meta Cloud. **Internet Computing, IEEE**, v. 17, n. 1, p. 69–73, Jan 2013.

SAUCELABS. “**Docs: Sauce Connect**”. [On-line]. 2014. Accessed: 27-Mar-2017. Disponível em: <<https://docs.saucelabs.com/reference/sauce-connect/>>.

_____. “**Docs: Sauce Connect**”. [On-line]. 2014. Accessed: 21-Jul-2014. Disponível em: <<https://docs.saucelabs.com/reference/sauce-connect/>>.

_____. “**Docs: Sauce Connect**”. [On-line]. 2016. Accessed: 21-Jan-2016. Disponível em: <<https://docs.saucelabs.com/reference/sauce-connect/>>.

SELENIUM. **Page Object Design Pattern** [On-line]. 2017. Último acesso: 05/03/2017. Disponível em: <http://www.seleniumhq.org/docs/06_test_design_considerations.jsp>.

SELENIUMHQ. **SeleniumHQ Browser Automation** [On-line]. 2014. Last access: 02/07/2014. Disponível em: <http://docs.seleniumhq.org/docs/01_introducing_selenium.jsp#introducing-selenium>.

_____. **SeleniumHQ Browser Automation** [On-line]. 2017. Último acesso: 02/07/2017. Disponível em: <http://docs.seleniumhq.org/docs/01_introducing_selenium.jsp#introducing-selenium>.

SENIOR, L. C.; CRISPIN, L. **Testing in the Fast Lane: Automating Acceptance Testing in an Extreme Programming Environment**. 2001.

SHAH, K. **Survey on cloud based testing tools**. Dissertação (Master of Science Thesis) — Tampere University of Technology, February 2014. Disponível em: <<https://dspace.cc.tut.fi/dpub/bitstream/handle/123456789/22070/shah.pdf?sequence=1>>.

SHAWISH, A.; SALAMA, M. Cloud computing: Paradigms and technologies. In: XHAFIA, F.; BESSIS, N. (Ed.). **Inter-cooperative Collective Intelligence: Techniques and Applications**. Springer Berlin Heidelberg, 2014, (Studies in Computational Intelligence, v. 495). p. 39–67. Disponível em: <http://dx.doi.org/10.1007/978-3-642-35016-0_2>.

SILVA, G.; ROSE, L.; CALINESCU, R. A Systematic Review of Cloud Lock-In Solutions. In: **2013 IEEE 5th International Conference on Cloud Computing Technology and Science (CloudCom)**. [S.l.: s.n.], 2013. v. 2, p. 363–368.

_____. Towards a Model-Driven Solution to the Vendor Lock-In Problem in Cloud Computing. In: **2013 IEEE 5th International Conference on Cloud Computing Technology and Science (CloudCom)**. [S.l.: s.n.], 2013. v. 1, p. 711–716.

SILVA, G. C.; ROSE, L. M.; CALINESCU, R. Towards a model-driven solution to the vendor lock-in problem in cloud computing. In: **2013 IEEE 5th International Conference on Cloud Computing Technology and Science**. [S.l.: s.n.], 2013. v. 1, p. 711–716.

SILVA, M. A. A. d.; ARDAGNA, D.; FERRY, N.; PÉREZ, J. F. Model-driven design of cloud applications with quality-of-service guarantees: The modacLOUDS approach, micas tutorial. In: **Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), 2014 16th International Symposium on**. [S.l.: s.n.], 2014. p. 3–10.

SNELL, J.; GLOVER, T. **Portability and interoperability - Similarities and differences explained [On-line]**. 2003. Último acesso: 11/07/2014. Disponível em: <<http://www.ibm.com/developerworks/library/ws-port/>>.

SOCIETY, I. C.; BOURQUE, P.; FAIRLEY, R. E. **Guide to the Software Engineering Body of Knowledge (SWEBOOK(R)): Version 3.0**. 3rd. ed. Los Alamitos, CA, USA: IEEE Computer Society Press, 2014.

SOMMERVILLE, I. **Software Engineering (8th Edition)**. [S.l.]: Pearson Addison Wesley, 2007.

_____. **Software Engineering**. 9th. ed. USA: Addison-Wesley Publishing Company, 2010.

SONG, H.; ZHANG, X.; FERRY, N.; CHAUVEL, F.; SOLBERG, A.; HUANG, G. Modeling adaptation policies as domain-specific constraints. In: _____. **Model-Driven Engineering Languages and Systems: 17th International Conference, MODELS 2014, Valencia, Spain, September 28 – October 3, 2014. Proceedings**. Cham: Springer International Publishing, 2014. p. 269–285. Disponível em: <http://dx.doi.org/10.1007/978-3-319-11653-2_17>.

STAPLES, M.; NIAZI, M. Experiences using systematic review guidelines. **J. Syst. Softw.**, Elsevier Science Inc., New York, NY, USA, v. 80, n. 9, p. 1425–1437, set. 2007. Disponível em: <<http://dx.doi.org/10.1016/j.jss.2006.09.046>>.

STAROV, O.; VILKOMIR, S.; GORBENKO, A.; KHARCHENKO, V. Testing-as-a-service for mobile applications: State-of-the-art survey. In: ZAMOJSKI, W.; SUGIER, J. (Ed.). **Dependability Problems of Complex Information Systems**. Cham: Springer International Publishing, 2015. p. 55–71. Disponível em: <http://dx.doi.org/10.1007/978-3-319-08964-5_4>.

SUN, H.; WANG, X.; YAN, M.; TANG, Y.; LIU, X. Service4all: A scalable paas platform for service-oriented software. **Int. J. Web Grid Serv.**, Inderscience Publishers, Inderscience Publishers, Geneva, SWITZERLAND, v. 11, n. 2, p. 229–245, abr. 2015. Disponível em: <<http://dx.doi.org/10.1504/IJWGS.2015.068901>>.

SUN, K.; LI, Y. Effort estimation in cloud migration process. In: **2013 IEEE Seventh International Symposium on Service-Oriented System Engineering**. [S.l.: s.n.], 2013. p. 84–91.

_____. Effort estimation in cloud migration process. In: **Service Oriented System Engineering (SOSE), 2013 IEEE 7th International Symposium on**. [S.l.: s.n.], 2013. p. 84–91.

SURHONE, L.; TENNOE, M.; HENSSONOW, S. **Write Once, Run Anywhere**. [S.l.]: VDM Publishing, 2011.

- TAURION, C. **Cloud Computing - Computação em Nuvem**. [S.l.]: BRASPORT, 2009.
- TECHTARGET. **What is interoperability?** [*On-line*]. 2006. Accessed: 07-Sep-2016. Disponível em: <<http://searchsoa.techtarget.com/definition/interoperability>>.
- TESTINGBOT. **‘Cross Browser Selenium Testing on browsers and mobile with TestingBot’**. [*On-line*]. 2015. Accessed: 27-Mar-2017. Disponível em: <<https://testingbot.com/>>.
- _____. **“Run cross browser Selenium tests on your local network”**. [*On-line*]. 2016. Accessed: 22-Jan-2016. Disponível em: <<https://testingbot.com/support/other/tunnel>>.
- _____. **Documentação do Testingbot**[*On-line*]. 2017. Último acesso: 10/05/2017. Disponível em: <<https://testingbot.com/support/getting-started/java.html>>.
- TOOSI, A. N.; CALHEIROS, R. N.; BUYYA, R. Interconnected cloud computing environments: Challenges, taxonomy, and survey. **ACM Comput. Surv.**, ACM, New York, NY, USA, v. 47, n. 1, p. 7:1–7:47, maio 2014. Disponível em: <<http://doi.acm.org/10.1145/2593512>>.
- TOSIC, M.; SESKAR, I. Resource specification and intelligent user interaction for federated testbeds using semantic web technologies. **Computer Networks**, v. 63, p. 84 – 100, 2014. Special issue on Future Internet Testbeds - Part {II}. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S1389128614000231>>.
- TRAN, V.; KEUNG, J.; LIU, A.; FEKETE, A. Application migration to cloud: A taxonomy of critical factors. In: **Proceedings of the 2Nd International Workshop on Software Engineering for Cloud Computing**. New York, NY, USA: ACM, 2011. (SECLOUD '11), p. 22–28. Disponível em: <<http://doi.acm.org/10.1145/1985500.1985505>>.
- TSAI, W.-T.; LI, Q.; COLBOURN, C. J.; BAI, X. Adaptive fault detection for testing tenant applications in multi-tenancy saas systems. In: **IEEE. Cloud Engineering (IC2E), 2013 IEEE International Conference on**. [S.l.], 2013. p. 183–192.
- TUNG, Y.-H.; LIN, C.-C.; SHAN, H.-L. Test as a service: A framework for web security taas service in cloud environment. In: **Service Oriented System Engineering (SOSE), 2014 IEEE 8th International Symposium on**. [S.l.: s.n.], 2014. p. 212–217.
- URBONAS, R. **Automated User Interface Testing** [*On-line*]. 2013. Último acesso: 15/07/2014. Disponível em: <<http://www.devbridge.com/articles/automated-user-interface-testing/>>.
- VILLANES, I. K.; COSTA, E. A. B.; DIAS-NETO, A. C. Automated mobile testing as a service (am-taas). In: **Proceedings of the 2015 IEEE World Congress on Services**. Washington, DC, USA: IEEE Computer Society, 2015. (SERVICES '15), p. 79–86. Disponível em: <<http://dx.doi.org/10.1109/SERVICES.2015.20>>.
- VITORMERIAT. **Modelos de Serviço na Nuvem: IaaS, PaaS e SaaS** [*On-line*]. 2011. Último acesso: 27/06/2014. Disponível em: <<http://vitormeriat.com.br/2011/07/08/modelos-de-servio-na-nuvem-iaas-paas-e-saas/>>.
- VMWARE. **“vCloud”**. [*On-line*]. 2014. Último acesso: 26/06/2014. Disponível em: <<http://vcloud.vmware.com/>>.
- WEISSLEDER, S. **Test Models and Coverage Criteria for Automatic Model-Based Test Generation with UML State Machines**. Tese (Doutorado) — Humboldt-University Berlin, Germany, 2010.

- WOHLIN, C.; HÖST, M.; HENNINGSSON, K. Empirical research methods in web and software engineering. In: _____. **Web Engineering**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006. p. 409–430. Disponível em: <http://dx.doi.org/10.1007/3-540-28218-1_13>.
- WOHLIN, C.; RUNESON, P.; HÖST, M.; OHLSSON, M. C.; REGNELL, B.; WESSLÉN, A. **Experimentation in software engineering: an introduction**. Norwell, MA, USA: Kluwer Academic Publishers, 2000.
- YAN, M.; SUN, H.; WANG, X.; LIU, X. Building a TaaS Platform for Web Service Load Testing. In: **2012 IEEE International Conference on Cluster Computing (CLUSTER)**. [S.l.: s.n.], 2012. p. 576–579.
- _____. WS-TaaS: A Testing as a Service Platform for Web Service Load Testing. In: **2012 IEEE 18th International Conference on Parallel and Distributed Systems (ICPADS)**. [S.l.: s.n.], 2012. p. 456–463.
- YI-CHEN, W.; YI-KUN, W. The research on software test pattern. In: **2011 International Conference on Future Computer Sciences and Application**. [S.l.: s.n.], 2011. p. 109–113.
- YU, L.; TSAI, W.-T.; CHEN, X.; LIU, L.; ZHAO, Y.; TANG, L.; ZHAO, W. Testing as a Service over Cloud. In: **Service Oriented System Engineering (SOSE), 2010 Fifth IEEE International Symposium on**. [S.l.: s.n.], 2010. p. 181–188.
- YU, L.; ZHAO, W.; DI, X.; KONG, C.; ZHAO, W.; WANG, Q.; ZHU, J. Towards Call for Testing: An Application to User Acceptance Testing of Web Applications. In: **33rd Annual IEEE International Computer Software and Applications Conference, 2009. COMPSAC '09**. [S.l.: s.n.], 2009. v. 1, p. 166–171.
- ZHANG, Q.; CHENG, L.; BOUTABA, R. Cloud computing: state-of-the-art and research challenges. **Journal of Internet Services and Applications**, Springer-Verlag, v. 1, n. 1, p. 7–18, 2010. Disponível em: <<http://dx.doi.org/10.1007/s13174-010-0007-6>>.
- ZHANG, Z.; WU, C.; CHEUNG, D. W. A Survey on Cloud Interoperability: Taxonomies, miscs, and Practice. **SIGMETRICS Perform. Eval. Rev.**, ACM, New York, NY, USA, v. 40, n. 4, p. 13–22, abr. 2013. Disponível em: <<http://doi.acm.org/10.1145/2479942.2479945>>.

REVISÃO SISTEMÁTICA SOBRE TESTE COMO SERVIÇO (TAAS)

A.1 A Research Roadmap of Vendor Lock-in Solutions to Testing as a Service (TaaS)

Conforme destacado por Riungu-Kalliosaari (2014), existe hoje uma grande necessidade de estratégias que permitem ampliar a adoção de testes baseados em nuvem, entre as quais pode-se destacar: métodos, ferramentas, gerenciamento e avaliação de serviços de teste na computação em nuvem. Os autores destacam também a importância de avaliar os benefícios e as limitações desse novo modelo de serviço em projetos e processos reais, levando em consideração a segurança, entre outros aspectos.

Devido à escassa quantidade de estudos primários sobre o teste na nuvem de uma forma geral, que sejam relevantes e tenham qualidade. Os estudos existentes não são rigorosos ou pouco conclusivos e principalmente devido à falta de evidência empírica na área de serviço de teste, muitas questões devem ser investigadas na perspectiva da migração entre os provedores de TaaS (PRIYANKA; CHANA; RANA, 2012). Torna-se necessária uma revisão sistemática da literatura para **identificar, classificar e comparar as principais soluções existentes para solucionar o problema de *vendor lock-in* no contexto de Teste como Serviço (TaaS)** e que contribuam com o avanço do estado da arte, direcionando novas pesquisas na área e, principalmente, facilitando a ampla adoção do modelo de serviço TaaS na computação em nuvem.

Uma revisão sistemática foi conduzida para encontrar as principais soluções para o problema de *vendor lock-in* no TaaS, conforme as diretrizes estabelecidas por Kitchenham e Charters (2007). Assim, as principais contribuições dessa atividade são:

- Análise, classificação e comparação das principais soluções adotadas para contornar o

problema de *vendor lock-in* no contexto TaaS;

- Análise das estratégias e soluções que poderiam ser reaproveitadas no contexto de TaaS a partir de outros modelos de serviço;
- Identificação de lacunas para futuras pesquisas sobre o TaaS, considerando principalmente as abordagens de Federação, *Multi-Cloud* e *Intercloud*.

Neste Apêndice será apresentada a seguinte revisão sistemática submetida para o Journal Information & Software Technology.

OLIVEIRA, R. R. de; PINTO, V. H. S. C.; LUZ, H. J. F. da; SIMÃO, A. da S. A research roadmap of vendor lock-in solutions to testing as a service (TaaS). Information & Software Technology, 2017. (Journal)

A.2 A Research Roadmap of Vendor Lock-in Solutions to Testing as a Service (TaaS)

A.2.1 Abstract

Context: Context: Testing as a Service (TaaS) offers an attractive combination of pay-per-use and cost reduction through the Internet. However, the lack of standardization in the writing, configuration, execution and management of tests causes the *vendor lock-in* problem, which limits the effective use of such new technologies and hampers the widespread adoption of TaaS. As studies conducted are neither rigorous, nor conclusive and due to the lack of empirical evidence, many issues must be investigated from the vendor lock-in in the context of TaaS.

Objective: This study identifies, analyzes and classifies the main current solutions to vendor *lock-in* in the context of TaaS and identifies gaps and directions for future research.

Method: A systematic review of 874 primary studies that describe the state-of-the-art of the main solutions was performed and 93 of such studies were selected and used for a thorough examination.

Results: The analysis of the current solutions for migration among TaaS platforms provided a knowledge base for the industrial community and interested researchers.

Conclusions: Testing as a Service is at an early stage and requires further empirical evidence of the existing solutions that must be automated for the migration among TaaS platforms, according to the steps established in the testing process in the cloud. Finally, the results indicate the existence of a gap regarding both application testing and testing of the Federation, multi-cloud and Intercloud environments. Therefore, further research is required for a widespread adoption of TaaS.

Keyword: Testing as a Service, TaaS, Cloud Computing, Vendor Lock-in, Migration and Systematic Review of Literature.

A.2.2 Introduction

The cloud computing paradigm has caused several changes to occur in the development of methodologies and testing of software products and services. The number of applications developed in the cloud has significantly increased in recent years (KRISHNA; JAYAKRISHNAN, 2013), because of the way computing resources are provided and used.

Cloud computing has opened new horizons for an efficient and effective software testing and Testing as a Service (TaaS) has been considered a new business and service model. One of the main objectives of TaaS in a cloud environment is the testing of certain software according to the customer's demand (pay-as-you-test), so that the quality of software can be ensured through the Internet (GAO; BAI; TSAI, 2011).

Cloud-based tests directly impact on the reduction of infrastructure costs, tool licenses, configuration and maintenance of test environments (SHAWISH; SALAMA, 2014). However, the major obstacles for a widespread adoption of TaaS are related mainly to the lack of effort metrics and the high costs involved in the migration process among providers (TRAN *et al.*, 2011; SUN; LI, 2013b).

Differences among the implementations of test services provided by suppliers also limit clients towards migrating data and applications at the different stages of the testing process in the cloud and cause users to be locked in a particular platform. The so-called lock-in problem is widely known in the literature and has been identified by the US government as a major obstacle for the adoption of cloud computing (ARMBRUST *et al.*, 2010; BADGER *et al.*, 2014). Due to high costs, legal restrictions and technical incompatibilities, customers are not allowed to switch from one provider to another.

Portability, interoperability and integration among cloud providers enable the vendor lock-in problem to be circumvented and competition among service providers to increase in cloud computing (HARSH *et al.*, 2012). Portability refers to the ability of an application to be carried out or adapted among different platforms at lower costs and efforts (RAFIQUE *et al.*, 2014). Interoperability refers to the ability of different heterogeneous systems to run, communicate and interact and integration is related to the combination of components or software systems for the achievement of a comprehensive solution (HARSH *et al.*, 2012; CLOUD... , 2009).

According to Riungu-Kalliosaari (RIUNGU-KALLIOSAARI, 2014), strategies, as methods, tools, management and evaluation of testing services in cloud computing are required for a wider adoption of cloud-based testing. The authors also pointed out the importance of the assessment of the benefits and limitations of a new service model in real projects and processes, taking into account safety, among other aspects.

Due to the limited number of relevant and high-quality primary studies on cloud testing (PRIYANKA; CHANA; RANA, 2012), a systematic review of the literature is fundamental for the identification, classification and comparison of the existing approaches, so that the lock-in problem in the context of Testing as a Service (TaaS) can be solved. Such approaches are also expected to contribute to advances in the state-of-the-art, directions of research in the area and the wide adoption of TaaS in cloud computing.

A systematic review was conducted under the guidelines established by Kitchenham et al. (KITCHENHAM; CHARTERS, 2007) and its contributions involve:

- Analysis, classification and comparison of the main solutions adopted for the circumvention of the lock-in problem in the TaaS context;
- Analysis of strategies and solutions to be used in the context of TaaS by other service models;
- Definition of categories from those solutions for the characterisation of research trends in the TaaS lock-in;
- Identification of gaps for future research on TaaS; and
- Ideas for new solutions to the lock-in in TaaS that involve mainly the Federation, Multi-Cloud and Intercloud approaches.

The article is organized as follows: Section A.2.3 discusses the main concepts involved in the study; Section A.2.4 describes the stages of the Systematic Review process; Section A.2.5 reports ideas for new solutions to the lock-in in TaaS in an efficient and effective way; Section A.2.6 addresses the threats to validity; Section A.2.7 describes the main studies on solutions to the lock-in problem in cloud computing; finally, Section A.2.8 provides the conclusions and suggests some future work.

A.2.3 Testing as a Service

Testing as a Service (TaaS) is a new business model for software verification and validation in a cloud computing environment. The consumer sends a requisition test to a TaaS provider through the Internet. The provider runs the tests and their results are returned to the customer (GAO; BAI; TSAI, 2011; GAO *et al.*, 2013).

Cloud testing involves the stages of writing, execution, configuration and management of tests. The model enables reductions in the pay-per-use costs when the cloud infrastructure is used. However, the lack of standardization in testing activities causes the lock-in problem, i.e., the consumer imprisonment in a specific provider. Such dependency can hamper the exchange of

providers due to the high maintenance costs for the migration of artifacts involved in each stage of the testing process (GAO; BAI; TSAI, 2011; GAO *et al.*, 2013).

Figure 46 shows an overview of TaaS usage in a functional testing context, specifically in automated User Acceptance Testing (UAT). Three entities, namely TaaS Customer, TaaS Provider and Web Application are involved. Test scripts are locally written by a TaaS customer. A Socket is open through the TCP protocol and the Selenium commands are sent to the provider (Saucelabs¹, for example) (SAUCELABS, 2014b). The provider runs the tests in a cloud infrastructure in the web application and returns the results to the consumer via REST API, from which information, such as browser version, operating system, logs, screenshots and videos of the tests can be obtained. All interaction and communication between the customer and the vendor occurs through Selenium Remote WebDriver and REST API (SELENIUMHQ, 2014).

Figura 46 – Overview of TaaS usage through automated testing

Portability in a TaaS context refers to the ability of software engineers to write a test once and run it in multiple TaaS platforms with no changes in it. Interoperability in a TaaS context refers to a standard protocol (or an abstraction layer) that enables heterogeneous TaaS providers to collaborate with each other in a transparent way to their clients. This is possible only if their REST APIs follow a common specification (GOVINDARAJAN; LAKSHMANAN, 2010; PETCU, 2011; SNELL; GLOVER, 2003).

The vendor lock-in problem is caused by the lack of portability and interoperability that prevents consumers from migrating to another vendor if the rate or testing service availability problems increase (ARMBRUST *et al.*, 2009; ARMBRUST *et al.*, 2010). The lock-in factor affects several service-delivery models, including TaaS. Therefore, we conducted a systematic review focusing on vendor lock-in solutions to TaaS, which is described in the following section.

¹ Saucelabs: <www.saucelabs.com>

A.2.4 Systematic Review Process

This research was based on guidelines established by Kitchenham et al. (KITCHENHAM; CHARTERS, 2007) for the Systematic Review of Literature (SRL), which provides a systematic search strategy through a reliable, accurate and auditable methodology for the detection of as much relevant literature as possible. Therefore, all available research can be identified, evaluated and interpreted in a particular issue area or phenomenon of interest. Figure 47 shows an overview of the process used in this study. The SRL process was divided into three main stages, namely planning, conducting and reporting. Our SRL was planned and conducted between November 2015 and April 2016 and the process of analysis and presentation of results occurred between May and August 2016. A repository was also created with all artifacts generated and available for download². The following subsections describe each step of the SRL process.

Figura 47 – Phases of the Systematic Review (Adapted from Lana et al. (LANA *et al.*, 2016))

A.2.4.1 Phase 1 - Planning

In this phase, the protocol with the objectives and search criteria, as well as the research questions and search string were set. The following subsections describe the main topics of the protocol developed by the authors and validated by the research supervisor, according to the guidelines of Kitchenham et al. (KITCHENHAM; CHARTERS, 2007).

The systematic review was aimed at identifying, analyzing and categorizing the main types of solutions to the lock-in problem in the Testing as a Service (TaaS) context. The main features, state-of-the-art and research opportunities on the theme were also presented.

A.2.4.1.1 Research Questions

Staples and Niazi (STAPLES; NIAZI, 2007) recommend limiting the scope of an SRL and choosing clear and exact research questions. Therefore, the following two research questions were raised:

² Dataset of the systematic review: <<http://goo.gl/QvyV80>>

- **RQ₁** - What main solutions to the vendor lock-in problem in the Testing as a Service (TaaS) context have been reported in the literature? What are their main features?
- **RQ₂** - How could the solutions to the vendor lock-in problem from other service models be reused in the context of TaaS?

A.2.4.1.2 Search String and Strategy

The search strategy was defined from the research questions and based on keywords “Testing as a Service”, “Cloud” and “Lock-in”. A search string was elaborated from such keywords, as shown in Figure 48, for the obtaining of as many primary studies related to terms “Testing as a Service” as possible, and their initials, i.e. “taas”. They were simultaneously associated with the vendor lock-in problem or cloud computing in general.

(*“testing as a service” or taas*) **and**
(*cloud or “cloud computing” or lock-in or “vendor lock”*)

Figura 48 – Search String

Only primary studies published between 1997 and the beginning of 2016 were considered. Among the articles published in this period, those whose complete texts were available and according to the relevance of the results returned by the search string. 1997 was chosen because it was the year in which term cloud computing was first used in the scientific literature (DURAO *et al.*, 2014).

The search string was adapted and used for ACM Digital Library³, IEEE Xplorer⁴, Scopus⁵, Springer Link⁶, ScienceDirect⁷, Compendex⁸ and Google Scholar⁹ databases. The documentation with the full records of the calibration process of the search string has been made available¹⁰. The seven databases were chosen because they have been widely used and scientifically accepted by the Software Engineering community in systematic review processes.

A.2.4.1.3 Control papers selected

Some studies were used for checking if the search string had been properly calibrated and returned primary reference studies, as well as relevant ones. The key papers used are Gao et al. (GAO; BAI; TSAI, 2011), Bai et al. (BAI *et al.*, 2011), Yu et al. (YU *et al.*, 2010), Candea

³ ACM Digital Library: <portal.acm.org>

⁴ IEEE Xplorer: <ieeexplore.ieee.org>

⁵ Scopus: <www.scopus.com>

⁶ Springer Link: <www.springerlink.com>

⁷ ScienceDirect: <www.sciencedirect.com>

⁸ Compendex: <www.engineeringvillage.com>

⁹ Google Scholar: <scholar.google.com.br>

¹⁰ History of the Search String: <http://bit.ly/1UJ97j1>

et al. (CANDEA; BUCUR; ZAMFIR, 2010) and Ciortea et al. (CIORTEA *et al.*, 2010). They were selected according to the number of citations. Although the chosen criterion is subjective, it helped researchers identify the main studies that should be returned by the search string and probably the most important ones in the field.

A.2.4.1.4 Inclusion and Exclusion criteria

Another important element of the systematic review planning is the definition of the inclusion (IC) and exclusion criteria (EC), detailed below. According to such criteria, primary studies relevant to the research questions are included and studies that do not satisfy them are excluded. The following inclusion criteria were defined for meeting each research question:

- **IC₁** - The primary study describes characteristics of vendor lock-in and the way they affect customers;
- **IC₂** - The primary study shows at least one approach for the solution of the vendor lock-in problem, e.g. architecture, environment, tool, framework, processes, techniques, APIs, methods, standards/norms and others;
- **IC₃** - The primary study addresses at least one difficulty in the use and adoption/migration regarding TaaS;
- **IC₄** - The primary study reports or proposes a new technique, strategy or portable and interoperable approach that mitigates the vendor lock-in problem for cloud application testing;
- **IC₅** - The primary study describes experiments and case studies on Testing as a Service (TaaS) in cloud computing or provides any solution to the lock-in problem.

Exclusion criteria:

- **EC₁** - The primary study is not related to any subject of this review;
- **EC₂** - The quality of the primary study is below 1.5 point;
- **EC₃** - The primary study is not written in English;
- **EC₄** - The publication year precedes 1997;
- **EC₅** - The primary study has not been made available by the author or has not been published;
- **EC₆** - The research is at an earlier version of a more complete study on the same topic;
- **EC₇** - Titles, abstracts and keywords do not fit TaaS;

- **EC₈** - The paper is duplicated;
- **EC₉** - The study presents an approach and research strategy regarding TaaS, however, it is a short paper of at most 5 pages and does not provide enough information;
- **EC₁₀** - The study is a systematic review, survey, mapping or any other secondary study.

A.2.4.1.5 Paper Quality Assurance

The procedures for a qualitative assessment of the studies were adopted according to the strategy defined by Jamshidi et al. (JAMSHIDI; AHMAD; PAHL, 2013), where G and S represent the general criteria with maximum score 1 or 25% of the weight and the specific criteria with maximum score 3 or 75% of the weight, as shown by equation (A.1). Such values were adopted, as the specific evaluation is more important than the general criteria. The maximum score achieved by a primary study was $G + S = 4$, therefore, studies whose score was lower than 1.5 were excluded. The criteria are described in Subsection A.2.4.2.2.

$$QualityScore = \left[\frac{\sum_{i=1}^5 G_i}{5} + \left(\frac{\sum_{i=1}^5 S_i}{5} \times 3 \right) \right] \quad (A.1)$$

A.2.4.1.6 Selection of Primary Studies

Below are the procedures for the selection of primary studies:

1. **Preselection:** exclusion based on title, keywords and abstract;
2. **Selection:** exclusion based on the full reading of the text;
3. **Quality Assessment:** evaluation of the quality of the selected papers.

A.2.4.1.7 Data Extraction Process

In this step, all necessary information was collected from primary studies for the research questions and according to the quality assessment criteria established in Subsection A.2.4.2.2 (KITCHENHAM; CHARTERS, 2007). The process was divided into three stages, as follows:

- Preparation of the electronic form¹¹ on Google Forms¹² for data extraction;
- Registration and storage of information for the research questions;

¹¹ Form Data Extraction: <<http://bit.ly/1JZgfka>>

¹² Google Forms: <<https://www.google.com/forms/about/>>

- Use of Jabref¹³ tool for the management of references.

The data were extracted independently by three PhD students - 2 from the Software Engineering area and 1 from the area of Distributed Systems - along with the help of a specialist responsible for the evaluation of the protocol, validation of the included and excluded studies and definition of the control, quality evaluation, protocol review and validation of the systematic review as a whole. The data were compared and disagreements were resolved.

After the extraction of the data, the quality of the item selected was assessed and, if approved, was included.

A.2.4.2 **Phase 2 - Conducting**

In this phase, as illustrated in Figure 47, the identification and selection of primary studies, as well as the quality assessment, data extraction and synthesis of data were performed. The following subsections describe the general analysis of the included studies and detail the quality assessment process.

Figure 49 shows the distribution of papers in the conduction process. 874 primary studies were returned after the application of the search string to each database. Google Scholar provided the largest number of primary studies (261). The ACM Digital Library, IEEE Xplorer, Scopus, SpringerLink, Science Direct and Compendex bases returned 102, 90, 160, 125, 85 and 51 studies, respectively. 234 duplicate studies were removed, which resulted in 640 studies for the second phase through the reading of the title, keywords and abstract. After the application of the inclusion/exclusion criteria, 90 studies were selected for a complete reading. Snowballing, i.e., the process of analysis of reference lists of each relevant study (KITCHENHAM; CHARTERS, 2007) was applied concomitantly and 3 studies were added, which resulted in 93 articles completely read. The inclusion/exclusion criteria were applied again in the selection stage along with the quality assessment and 23 other studies were excluded. At the end of the process, only 70 papers had been selected. Due to space limitations, the list of selected papers will not be shown, although it is available¹⁴.

Figure 50 shows the distribution of the papers selected from each search database. The bases with the largest number of selected studies were Xplorer IEEE (39 studies, or 56%), Google Scholar and SpringerLink, with 11 (16%) and 5 (7%) studies, respectively. The remaining bases, including primary studies obtained through snowballing, accounted for 12 articles (21%).

A.2.4.2.1 **General analysis of the Included Studies**

The primary studies were analyzed according to distribution per year, solution types, publication per country and evaluation of quality.

¹³ Jabref: <<http://jabref.sourceforge.net/>>

¹⁴ Complete list of selected studies: <<http://goo.gl/mE2kKs>>

Figura 49 – Distribution of papers in the conduction phase

Figura 50 – Distribution of papers selected by database

From 2009, the number of studies on functional tests has increased to 18 (25.7%). However, only 9 (12.9%) address non-functional tests in the cloud environment. In contrast, only 3 (4%) model-based testing and 1 (1%) primary study addressed structural tests in the cloud environment.

Figure 51 shows the countries where most research in the area has been developed. The United States leads the ranking with 19% of included studies, followed by China and Germany, with 10%, and the United Kingdom with 7%.

Different types of solutions were categorized by the primary studies and the most widely adopted by the industry and academia are MDE, DSL or solutions based on models, migration processes, abstraction layers and approaches involving some type of architecture, framework, APIs and platforms. From the 70 studies included, 64, i.e., 91.42%, have been published in the past five years, which shows the growing interest towards the solution of the vendor lock-in problem.

Figure 52 shows the disposal of the papers selected from book chapters, theses and

Figura 51 – Publication per country of the authors

magazines, symposiums, workshops, conferences and journals. Most primary studies have been published in events, which indicates the TaaS area is at an early stage and requires further research.

Figura 52 – Number of selected papers per workshops, journals, symposiums, conferences, book chapters, theses and magazines.

A.2.4.2.2 Quality Assessment

The quality assessment was based on the procedures described in subsection A.2.4.1.5. Table 26 shows the general ($G_1 - G_5$) and specific criteria ($S_1 - S_5$). Each paper selected was assessed by reviewers according to the criteria in full readings. The score was set to 1 if the paper fully addressed the criterion, 0.5 if it addressed it partially and 0 when it did not satisfy it.

Although the paper of Hsieh et al. (HSIEH *et al.*, 2014) addresses the multi-service type test and is considered short, it did not achieve the minimum score of 1.5 required by the criteria established by specific and general issues. In contrast, although (TUNG; LIN; SHAN, 2014) was considered complete according to the number of pages and proposes a framework

for TaaS security testing, it did not achieve the minimum score required. Finally, although the paper of Oliveira et al. (OLIVEIRA *et al.*, 2015) showed a five-page limitation, it achieved a significant score of 2.7 points and covered most of the issues and quality criteria. The final quality assessment of all selected primary studies is available¹⁵.

Tabela 26 – Quality Assessment Form

General items for the evaluation of quality (G)		Score		
		Yes 1.0	Part. 0.5	No 0
<i>G₁</i>	<i>Are the definition of the problem and the objectives clearly presented?</i>			
<i>G₂</i>	<i>Has the research methodology been clearly defined?</i>			
<i>G₃</i>	<i>Do the authors present the study's limitations, as well as ideas for future work?</i>			
<i>G₄</i>	<i>Are the study's contributions in agreement with the results?</i>			
<i>G₅</i>	<i>Has the idea of the study been explicitly mentioned?</i>			
Specific items of quality assessment (S)				
<i>S₁</i>	<i>Does the research clearly focus on the lock-in Testing as a Service context?</i>			
<i>S₂</i>	<i>Do the details of related research illustrate or clearly describe some approach, method or strategy for the solution of the lock-in problem?</i>			
<i>S₃</i>	<i>Does the research involve a case study or a controlled experiment (evidence) on Testing as a Service (TaaS) in cloud computing?</i>			
<i>S₄</i>	<i>Has the approach adopted been evaluated or validated by the authors?</i>			
<i>S₅</i>	<i>Are the results relevant to the research area?</i>			

A.2.4.3 Phase 3 - Presentation of Results

This section provides the answers to the research questions from the analysis of the primary studies included.

A.2.4.3.1 RQ₁

What main solutions to the vendor lock-in problem in the Testing as a Service (TaaS) context have been reported in the literature? What are their main features?

In order to map the focus of the selected studies in quantitative terms, we defined some categories according to the paper domain. A paper can be classified in more than one category. For instance, Munch (MUNCH, 2013) describes the main interoperability issues that lead to the vendor lock-in and provides a brief overview of the current standardization process and the abstraction layer approach known as meta cloud. After a full reading, the paper was classified into two categories, namely “API's, Protocols and Standards” and “Abstraction layers (Middleware)”.

Figure 53 shows the quantitative distribution of the studies from 2009 to 2015 in 7 categories. The number of articles is smaller than that used in the data extraction, as the bubble chart considered only primary studies that provided some solution to the vendor lock-in problem in TaaS, or solutions from other service models in which the strategy could be reused in the

¹⁵ Complete List of Quality Assessment: <<http://goo.gl/QvyV80>>

Figura 53 – Distribution of primary studies according to categories

context of TaaS. The next subsections provide an overview of the main primary studies organized by categories.

a) MDE, DSL or model-based solutions

MDE can contribute to software development and automatic code generation through templates with no coding. Interoperability has become an important requirement, as it enables the generation of a platform-independent code, hence, communication of heterogeneous systems over the Internet. Regarding TaaS, an approach that uses this solution can reduce the complexity in the migration process among TaaS providers.

According to Silva et al. (SILVA; ROSE; CALINESCU, 2013c), MDE enables software engineers to work through a higher-level abstraction layer, focusing on the cloud application business rules and disregarding implementation details (FERRY *et al.*, 2013; SILVA *et al.*, 2014). Moreover, productivity and quality increase and maintenance costs and development of new software features are reduced. Such a category of solutions focuses on legacy software migration to the cloud (FERRY *et al.*, 2013).

Although we did not find specific solutions that involve MDE to handle lock-in in TaaS, we believe our approach is promising, since it was used in other service models for the tackling of the issue.

b) Abstraction Layers (Middleware)

Ten papers in the SRL cited abstraction layer-related issues as a promising research direction. The main purpose of an abstraction layer, middleware or proxy refers to the migration of application features among different cloud vendors at running time in an automated way aided by tools (SATZGER *et al.*, 2013).

As consequences of vendor lock-in, we can highlight lack of portability and interoperability among cloud service providers, as aforementioned. Such requirements in applications

deployed in hybrid clouds have become more challenging due to the heterogeneity of programming languages and technologies among platforms of service providers in the cloud.

Sun et al. (SUN *et al.*, 2015) proposed a scalable middleware via a framework designed to address changes in requirements dynamically, taking into account aspects of scalability and application deployment. According to the authors, the flexible and elastic provisioning of test environments consists of underlying resources and run-time middleware. They also identified three challenges for performing the web service load test, namely (a) mass user simulation from different geographic locations, (b) flexible and scalable test environments and (c) automated load, stress and performance testing process.

Rafique et al. (RAFIQUE *et al.*, 2014) proposed a basic middleware architecture through a common API for storing and managing asynchronous tasks for applications in hybrid clouds specifically in the context of Platform as a Service (PaaS). The middleware can dynamically decide on the requests and tasks to be executed in a specific part of the hybrid cloud. As a result, similar middleware architectures could be designed for different delivery models, as TaaS. For instance, a possible agnostic solution that uses abstraction layers would be the abstraction of the complexities and technological differences among providers for migration in run-time (MUNCH, 2013). The solution would abstract any testing services from TaaS providers (e.g. functional, non-functional, structural and model-based testing).

c) Maintenance, Reengineering and migration

The migration of software testing to the cloud can overcome difficulties and complement conventional testing activities (ALI; BADR, 2015). After the migration of cloud tests specifically to a TaaS provider, a customer can decide on the switching of providers. Despite business requirements, new market demands, attractive cost-benefit relations, functionalities, and availability, the process lacks automated approaches for its facilitation. The client must often run the re-engineered program, which consumes the proprietary REST APIs of each TaaS provider. Due to the lack of standardization, this maintenance is usually costly and biased to errors.

According to Silva et al. (SILVA; ROSE; CALINESCU, 2013b), the following four critical factors directly influence the migration among different platforms of cloud vendors: process, effort, stakeholders (developers, testers and end users etc.) and reengineering or maintenance of the source code. Tran et al. (TRAN *et al.*, 2011) developed a metric that estimates the efforts required for the migration among cloud platforms through parameters, such as maintenance of the source code and installation and configuration, which directly impact on costs.

The migration of all artifacts necessary for the testing process among TaaS providers is still one of the main challenges for both researchers and testers. One of the biggest trade-offs occurs due to the use of a proxy or a tunnel in situations in which web application testing is not publicly accessible. On the one hand, the proxy can be considered an extra layer of security, which enables a secure connection between the client and the vendor's virtual machines. On

the other hand, it is highly coupled with the specific TaaS vendor, as each vendor imposes its configurations (SAUCELABS, 2016; BROWSERSTACK, 2016; TESTINGBOT, 2016).

According to Grace (LEWIS, 2012), the standardization of authentication through OpenID or OAuth-based authentication protocols enables the sharing of user data among different TaaS providers that support this pattern, which provides a solution to the locking problem-in regarding user's records, authentication, and security issues associated with the TaaS customer profile.

d) API's, Protocols, Standards and Open Standards

Application programming interfaces (API) can specify routines, standards and protocols and the current ones enable both access and implementation of services in the cloud (MARTINO *et al.*, 2014). The lack of interoperability and portability among services from different providers is due to the non standardization of such APIs and leads to the vendor lock-in problem, hence, future reliability problems regarding security or privacy and substantial increases in rates charged for services (ALLISON; TURNER; ALLEN, 2015).

Based on the major studies selected in this category, we observed if each cloud provider adopted some standard or open protocol, an application along with its data would easily migrate among different cloud providers. For example, Open Cloud Computing Interface (OCCI) is an open standard created over HTTP and REST architectural style for managing the IaaS service model and contributing to cloud integration and interoperability (EDMONDS *et al.*, 2012). The protocol also provides an interoperable interface for other service models, as PaaS and SaaS and can possibly be used for TaaS.

Other standards from Apache, as Libcloud¹⁶ and Deltacloud¹⁷ work as an abstract layer among different APIs of each cloud provider. Libcloud is a Python library that abstracts differences among specific APIs from cloud providers and enables the management of several cloud resources through a unified API (SATZGER *et al.*, 2013). Deltacloud provides an API that works as a wrapper and drivers for the connection to cloud providers. Details are abstracted from the native API implementation of each supported cloud, i.e., Deltacloud REST API, DMTF CIMI API or EC2 API.

The main benefit of a standardized API is the abstraction for the unification of differences among vendors. Therefore, both time spent on the implementation of different APIs and complexity of the source code can be reduced.

e) Inter-cloud, Federation or Multi-cloud

Inter-cloud, federation or multi-cloud concepts are under construction. Although different definitions have been used by the scientific community, the main benefits are related to the integration and aggregation of cloud services (BERMBACH; KURZE; TAI, 2013; TOOSI;

¹⁶ Apache Libcloud: <<http://libcloud.apache.org>>

¹⁷ Apache Deltacloud: <<http://deltacloud.apache.org>>

CALHEIROS; BUYYA, 2014).

According to Bernstein et al. (BERNSTEIN *et al.*, 2009), an inter-cloud can be defined as an interoperable “cloud of clouds” that communicates through open standard protocols for connecting their components, so that they interact ubiquitously, as telephones and Internet systems (TOOSI; CALHEIROS; BUYYA, 2014). Rochwerger et al. (ROCHWERGER *et al.*, 2009) defined cloud federation as a group of aggregate vendors who collaborate mutually by sharing their resources and enabling the migration of part or all application among the vendors involved taking into account aspects of security and autonomy.

Petcu (PETCU *et al.*, 2013) identified the-state-of-the-art of only the Multi-Clouds development process and argued the emergence of several types of solutions that use the multi-cloud model is beneficial, although the different approaches have not been widely adopted and accepted. Terms multiple clouds, multi-cloud or cloud-cross enable the provisioning of resources among different cloud systems, however, vendors may or may not cooperate mutually.

The federation holds a consensus on the aggregation of features among vendors, while this may not always occur in multi-cloud (GROZEV; BUYYA, 2014a). Finally, the intercloud can include one or more federated clouds and multi-clouds (GROZEV; BUYYA, 2014a). The concepts included in this category are discussed with more detail in Section A.2.5.

f) Semantic approach

The application of semantic web and meta-data to cloud computing offers several benefits (RANABAHU; SHETH, 2010). Apart from the semantic representations that contribute to the mapping among different cloud services, the concepts can also be represented in Web Ontology Language (OWL) in an agnostic form of the cloud provider. Therefore, services of functionality and communication parameters are exposed, which contributes to dynamic resource discovery, interoperable access, and cloud service composition (MARTINO *et al.*, 2014).

Martino et al. (MARTINO *et al.*, 2014) proposed a multi-platform API for the access of resources in the cloud and abstraction of the heterogeneity of technologies among cloud providers through a semantic representation of the cloud service for the establishment of a standard and reduction in the complexity of the source code. It enables the use of a single API call for the access to resources from more than one vendor.

Ranabahu and Sheth (RANABAHU; SHETH, 2010) designed a method based on the semantic partitioning of the application through a high-level specification, so as to facilitate the portability of applications and data in the cloud. Currently, the entire process uses a DSL, however, the authors argue the approach is flexible and enables the coupling of generic transformers for the generation of a specific platform code.

The use of semantic web and ontologies can also contribute to the description, composition and mapping among the services of different TaaS providers. The approach can also reduce the differences among TaaS vendors in a dynamic and standardized way and the considerable

efforts devoted to the migration among vendors.

g) Architecture, Framework and Platforms

Several approaches, such as frameworks (TUNG; LIN; SHAN, 2014; ALI; BADR, 2015; VILLANES; COSTA; DIAS-NETO, 2015), platforms (YU *et al.*, 2010; YAN *et al.*, 2012b; SUN *et al.*, 2015), APIs (EDMONDS *et al.*, 2012; RAFIQUE *et al.*, 2014) and architectures (EDMONDS *et al.*, 2012; RAFIQUE *et al.*, 2014) can be used to mitigate the vendor lock-in of TaaS.

Edmonds et al. (EDMONDS *et al.*, 2012) discussed the development and architecture of the OCCI standard, which defines a unified extensible interface while taking into account the different aspects of service provision by vendors. The modular approach of OCCI uses the Composite design standard and focuses on interoperability to enhance extensibility, flexibility and discovery of capabilities (EDMONDS *et al.*, 2012). Yu et al. (YU *et al.*, 2010) showed a 5-layer cloud TaaS architecture and an automated testing system for the platform. The architecture adopts cloud techniques for building elastic resource infrastructures and provides several test services to test consumers. Unit testing was used to validate their proposal. The platform can support testers in the configuration of test environments, including test execution and summarization of results.

Tung et al. (TUNG; LIN; SHAN, 2014) introduced a TaaS framework for web application security testing through tools for the detection of vulnerabilities. Ali and Badr (ALI; BADR, 2015) designed a performance framework that optimally allocates resources, promotes efficient scheduling and describes methods that generate performance test cases. Finally, Villanes et al. (VILLANES; COSTA; DIAS-NETO, 2015) developed a framework that offers automated testing for mobile applications towards helping developers and testers assess the quality of their short-term applications and different devices through a user-friendly web interface.

Yan et al. (YAN *et al.*, 2012a; YAN *et al.*, 2012b) presented a WS-TaaS platform that facilitates the load testing for Web Services. They also conducted an experimental study whose data revealed the platform was more effective and accurate than some testing tools. Sun et al. (SUN *et al.*, 2015) designed a cloud services platform, called Service4All, through which new modules were developed and some existing ones from WS-TaaS were extended. The authors were mainly concerned with the problem of scalability in the context of PaaS. The platform aimed at improving the productivity of service-oriented software developers. As an evaluation of the study, they showed the effectiveness of Service4All.

A.2.4.3.2 RQ_2

How could the solutions to the vendor lock-in problem from other service models be reused in the context of TaaS?

As discussed in the previous section, some solutions to the vendor lock-in problem in the TaaS context were identified. However, many strategies are targeted to other service models and

can be reused and adapted to the TaaS domain. Some possibilities are described below.

The Model-Oriented Engineering (MDE) approach could be expanded to the testing service in the cloud. MDE models could generate test cases from a TaaS provider to another automatically (HAMDAQA; TAHVILDARI, 2014) and the MDE approach can also be used for the automatic processing of all migration processes among TaaS vendors (ALLISON; TURNER; ALLEN, 2015). Domain Specific Languages (DSL) could be used for automatically turning a test code TaaS platform into another (RANABAHU; SHETH, 2010).

The CloudML and iCloudML solutions described by Ferry et al. (FERRY *et al.*, 2013) and Allison et al. (ALLISON; TURNER; ALLEN, 2015), respectively, were built on the techniques and MDE methods and could contribute towards exploiting the peculiarities of each TaaS provider in the software testing process in the cloud and optimizing the performance, availability and costs of testing and applications used in the management of the test results.

The abstraction layer solution through conversion technology, which promotes an optimal combination of cloud services, could be used as an agnostic solution of TaaS vendors for Structural, Functional, Non-Functional and model-based Testing and a support for runtime migration.

The abstraction layer solution with conversion technology, which promotes an optimal combination of cloud services with cost-effective strategies, not only supports runtime migration, but also can be used as an agnostic solution regardless of the TaaS providers or type of testing cloud technique (MUNCH, 2013; CANDEA; BUCUR; ZAMFIR, 2010; SATZGER *et al.*, 2013).

The solution proposed by Silva et al. (SILVA; ROSE; CALINESCU, 2013c) could automate migration among TaaS providers and contribute primarily to the portability and interoperability of applications and tests in the cloud.

Candea et al. (CANDEA; BUCUR; ZAMFIR, 2010) established some impacts and challenges related to the automation of testing in the cloud. One of them is the direct testing of the code in binaries, rather than source code, since most software is still proprietary, which can avoid the lock-in problem.

The standardization of authentication via OpenID or authentication protocols based on OAuth has enabled users to use data among the various TaaS providers that support those standards for the tackling of the lock-in problem regarding the user's account, authentication and security issues associated with the profile of the TaaS customer (LEWIS, 2012). Moreover, the OCCI protocol could be extended for the testing of cloud services and contribute primarily to the portability and interoperability of applications and test data (EDMONDS *et al.*, 2012).

The inter-cloud, multi-cloud and federation approaches could add the specific functionalities of each TaaS provider if suppliers came into a consensus in the future. Moreover, costs would be optimized according to the features offered by each vendor (BERMBACH *et al.*, 2011; SILVA *et al.*, 2014). The approach may use a mediation mechanism, known as cloud broker,

for the migration among TaaS providers and collection of QoS metrics for decision-making (BUYYA; RANJAN; CALHEIROS, 2010).

Ontologies can open new horizons regarding the discovery and composition of TaaS services in a dynamic and standardized way. The construction of a semantic description of services enables the conciliation and combination of different semantic representations, so that differences among distinct TaaS services can be abstracted (MARTINO *et al.*, 2014).

The modeling framework described by Ferry *et al.* (FERRY *et al.*, 2014) for cloud computing through Model Driven Engineering (MDE) could be adapted to circumvent the complexity of the heterogeneity of different TaaS platforms through a domain-specific language and support tools for multi-cloud application testing.

The solution proposed by Anastasi *et al.* (ANASTASI *et al.*, 2014) could mitigate the vendor lock-in problem in the structural, model-based or comprehensive testing, in which the instrumented code or model would be partitioned, for example, for the analysis of the source code coverage. Since symbolic execution is one of the methods used in structural test for the execution of all paths of a program and requires a large amount of such technical infrastructure, it could be applied to the context of TaaS for providing the non-covered characteristics of code portions and valuable information on the adaptation or proposal of new techniques that improve the automatic migration among TaaS providers (HSIEH *et al.*, 2014; KOSMATOV *et al.*, 2013; ALEB; KECHID, 2012).

A.2.4.3.3 Research Agenda

This section addresses the main trends of cloud-based tests for contributions and future research in the TaaS area.

1. **Empirical evidence:** few empirical studies have addressed tests in the cloud environment; only 9 (12,9%) have shown some type of controlled experiment for the evaluation or validation of their approaches;
2. **Structural test:** scarce research has been developed on the structural testing technique in the cloud environment, i.e., only one study (KOSMATOV *et al.*, 2013) has been found.
3. **Model-Based Testing:** research on the model-based testing technique involving mathematical formalism in the cloud environment is required, as only three studies have been found (KIRAN *et al.*, 2014; SONG *et al.*, 2014; HU; LEI; TSAI, 2016);
4. **Multi-tenancy tests:** the two main characteristics of multi-tenancy architecture, namely resource sharing and customization of the software according to the customer's needs have been identified. Because of the different combinations, compositions and interactions that can occur among the different components of the architecture, finding technical strategies

and test criteria that complement the approaches of conventional tests has been a challenge (HU; LEI; TSAI, 2016; LI *et al.*, 2015; TSAI *et al.*, 2013);

5. **Semantic approaches:** the need for further research on semantic approaches for combating the lock-in problem has also been identified, as only three primary studies on the subject have been found (TOSIC; SESKAR, 2014; MARTINO *et al.*, 2014; RANABAHU; SHETH, 2010);
6. **Federation Multi-cloud and Intercloud:** significant research opportunities unexploited to date regarding investigations on the application and adoption of TaaS for application testing or testing of federation environments, multi-cloud and intercloud have been identified. As an example, Kolb et al. (KOLB; LENHARD; WIRTZ, 2015) highlighted the lack of research on the testing and evaluation of performance in unified execution environments due to differences among the vendors.

A.2.5 Ideas for new solutions to lock-in of TaaS

Grozev and Buyya (GROZEV; BUYYA, 2014a) presented an architectural taxonomy of inter-cloud developments. Among the approaches for the development of inter-cloud federations, the authors highlight centralized and peer-to-peer classifications. The first relies on a central entity that manages the allocation of resources among cloud providers, whereas the second involves a direct communication among the cloud providers through a consensus. Regarding multi-cloud development, they highlight two other models, namely multi-cloud service and multi-cloud library. In the former, a broker runs and deploys an application among cloud providers, whereas the second usually uses a custom broker that provisions an across clouds application in a unified form, which facilitates the use of multiple cloud.

From those inter-cloud development models and challenges regarding the TaaS lock-in, we have adapted such classifications to discuss possible solutions, which require more research efforts. Figure 54 illustrates the approaches highlighted by the authors with adaptations to the TaaS field. Figures 54 (a) and (b) show the centralized inter-cloud federation and the peer-to-peer inter-cloud federation, respectively. The Multi-TaaS service architectures and Multi-TaaS library are displayed in (c) and (d), respectively.

A possible solution to TaaS lock-in would be the creation of a unique standard among different TaaS vendors on the market. Figures 54 (a) and (b) show the development of a Federation of Testing as a Service (FTaaS), however, it would require an open standard or protocol of interest and widely accepted by TaaS providers. FTaaS would enable the abstraction of different TaaS on the market by a common specification that describes a TaaS. The idea would be similar to CloudFoundry¹⁸, i.e., a PaaS that promotes the abstraction of different IaaS. However, the

¹⁸ CloudFoundry: <<https://www.cloudfoundry.org/>>

Figura 54 – Federation and Multi-Cloud Approaches to TaaS (Adapted from Grozev and Buyya (GROZEV; BUYYA, 2014a))

proposal is currently impracticable in the TaaS context, due to the lack of consensus among providers.

The Multi-TaaS solutions depicted in (c) and (d) enable the handling of the lock-in in two ways, i.e., through a service as RESTful, which promotes the abstraction of complexities among TaaS vendors, and through the use of a library by clients in which an abstraction layer would be created among different TaaS providers (GROZEV; BUYYA, 2014a; TOOSI; CALHEIROS; BUYYA, 2014; GROZEV; BUYYA, 2014b).

An initiative following the Multi-TaaS library approach is under development by the Software Engineering Laboratory¹⁹ of the University of São Paulo, in Brazil. The main goal is the identification of the way the lock-in impacts on the cloud testing process and aggregation of common and specific characteristics of some TaaS providers, as Saucelabs, Browserstack and Testingbot. The authors also aim at publishing a library in the central repository of Maven and making it available to the entire industrial community and interested researchers.

TaaS providers do not collaborate in a mutual way, i.e., there is no consensus on the integration and aggregation of functionalities among them. Their different functionalities are not independent, i.e., the functionality of each TaaS provider depends on the complete execution of the test and the use of the vendor's specific platform. Therefore, tests cannot be run on Saucelabs for the obtaining of the video of its execution through Browserstack. If the features were independent, costs for TaaS consumers would be optimized (GROZEV; BUYYA, 2014a; TOOSI; CALHEIROS; BUYYA, 2014).

A.2.6 Threats to Validity

Threats to validity concerning RSL refer to:

¹⁹ LabES: <<http://www.labes.icmc.usp.br/site/>>

Primary studies selection. In order to ensure an unbiased selection process, we defined research questions in advance and inclusion and exclusion criteria were devised, as aforementioned. Regarding quality assessment, each primary study was scored by reviewers and a set of the most relevant primary studies was obtained.

Missing important primary studies. The search for primary studies was conducted in several databases with search string refinement. However, some important papers may have been missed. Nevertheless, this issue was mitigated by the selection of search engines regarded as the most relevant scientific sources. The authors of the unavailable studies were contacted by e-mail, which may involve risk of bias.

Reviewers reliability. The reviewers who participated in the study are PhD candidates in Computer Science and work on cloud computing. The protocol was assessed by specialists in the same field and software testing.

Data extraction. Another threat refers to the way the data were extracted from the primary studies, since not all information fit our questions. Several sources, as technical reports and white papers were consulted, so that the validity of the process could be ensured.

A.2.7 Related Work

Haugset and Hanssen (HAUGSET; HANSEN, 2008) conducted an industrial case study and a literature review involving the automation of acceptance tests for the establishment of an overview and identification of relevant topics on the subject. An extensive knowledge base was created for professionals and interested researchers.

Jamshidi et al. (JAMSHIDI; AHMAD; PAHL, 2013) conducted a literature review for identifying gaps and future research in the area of migration from legacy systems for cloud computing through a characterization framework and comparison among primary studies. According to the authors, migration to the cloud has increased, therefore, the development of a tool that helps the automation process is fundamental. They also identified cloud systems must adapt to an architectural form and exhibit self-adaptive skills.

Priyanka et al. (PRIYANKA; CHANA; RANA, 2012) described a systematic mapping process listing approaches that involve testing frameworks and key challenges in the software testing area in the cloud environment. They also evaluated cloud-based testing techniques empirically for the categorization and classification of primary studies.

Silva et al. (SILVA; ROSE; CALINESCU, 2013a) identified research gaps and classified and compared solutions to the problem of vendor lock-in taking into consideration aspects of interoperability and portability in cloud computing regardless of a specific service model.

Starov et al. (STAROV *et al.*, 2015) reviewed the-state-of-the-art of mobile application testing in the cloud computing environment and highlighted the complexity of testing mobile

applications, as they involve different combinations of mobile platforms, hardware and other types of software. The authors highlighted the importance of an extensive testing migration to the cloud for future cost reductions and the demand for a scalable and flexible platform as a supporting tool for multidirectional tests involving different services and testing techniques in cloud computing.

Pahl et al. (PAHL; XIONG; WALSHE, 2013) also addressed the migration process and the differences between deployment models in the cloud computing environment based on three case studies in the industry conducted through interviews with experts.

Opara-Martins et al. (OPARA-MARTINS; SAHANDI; TIAN, 2014) conducted a critical review of the impact of the vendor lock-in problem on the service reliability and widespread adoption of cloud computing considering legal issues in moving cloud application data among different areas of jurisdiction that involve laws, policies and governments of different countries.

Nogueira et al. (NOGUEIRA *et al.*, 2016) conducted a bibliographical review to identify definitions, concepts and updated characteristics on interoperability in the context of cloud computing that help software engineer and researchers develop interoperable applications in the cloud.

Finally, each study established some type of solution to the lock-in problem in cloud computing. In contrast, our RSL has provided an overview of cloud testing and identified the main solutions to the lock-in of TaaS.

A.2.8 Concluding Remarks and Future Work

This paper has presented the results of an RSL that identified the state-of-the-art of solutions to the vendor lock-in in the TaaS context. Different approaches for the mitigation of the problem and the way solutions of other service models can be adapted to the TaaS context were identified, analyzed and categorized.

A clear understanding of the impact of vendor lock-in on the TaaS context and the recent advances towards solving this issue are essential for the development of research in this field. 93 primary studies were selected from conferences, symposiums, workshops and others sources, according to quality criteria and data were collected from a study set. The main contribution of this research regards the analysis of solutions and their classification into categories. The RSL can assist TaaS researchers and practitioners, once the major research trends have been mapped and some issues involving the complexity in migration among TaaS providers were discussed.

TaaS is at early stages and requires more empirical evidence of solutions for the mitigation of the lock-in problem. The study also identified the need for new research on structural testing, model-based testing, multi-tenancy testing, semantic and service-testing approaches for applications running in federated and multi-cloud clouds, as well as the environments that form an intercloud. As future work, we aim at investigating a solution that is portable and

interoperable for reducing the impact caused by the vendor lock-in on TaaS. Our multi-TaaS open source library, previously described, is under development and will be the subject of future research towards the mitigation of the vendor lock-in in the automated testing and promotion of a widespread adoption of cloud-based testing.

Acknowledgements

The authors acknowledge the Institute of Mathematics and Computer Sciences of the University of São Paulo and the Federal Institute of Education, Science and Technology of South of Minas Gerais - IFSULDEMINAS, Poços de Caldas campus for making this study possible and funding this research, Renan Martins, at ThoughtWorks²⁰ in Brazil, Roberto Naves Ungarelli, at Oceano Azul TI²¹ of Brazil for their contributions, and Angela Cristina Pregnolato Giampetro for the review and proofreading of the manuscript. They are also indebted to the reviewers of Information and Software Technology (IST) Journal for their valuable comments and suggestions.

Funding

The Institute of Mathematics and Computer Sciences of the University of São Paulo and the Federal Institute of Education, Science and Technology of South of Minas Gerais - IFSULDEMINAS, Poços de Caldas campus financed this research.

²⁰ *ThoughtWorks*: <<https://www.thoughtworks.com/pt>>

²¹ *Oceano Azul TI*: <<http://oceanoazulti.com/>>

DOCUMENTOS COMPLEMENTARES

B.1 Fluxogramas dos Experimentos

Neste apêndice, são mostrados os fluxogramas dos experimentos conduzidos no IFSUL-DEMINAS e na UFSCar. A Figura 55 mostra o fluxograma do planejamento dos experimentos e as Figuras 56 e 57 apresentam os fluxogramas das tarefas do 1º e do 2º experimento, respectivamente.

Figura 55 – Planejamento das Tarefas do Experimento

Fonte: Autoria própria

Figura 56 – Fluxograma das Tarefas do 1º Experimento

Fonte: Autoria própria

Figura 57 – Fluxograma das Tarefas do 2º Experimento

Fonte: Autoria própria

TABELAS DOS DADOS DOS EXPERIMENTOS

C.1 Tabelas de Dados dos Experimentos Controlados

Neste apêndice, são apresentados todos os dados obtidos dos experimentos controlados conduzidos no IFSULDEMINAS e na UFSCar.

Tabela 27 – Dados do 1º experimento controlado - IFSULDEMINAS

ID da Amostra	Complexidade Ciclomática	LOC	Tarefa	Período	Tratamento	Tempo Total (Minutos)	Esforço
1	1	153	3	2º	1	7	0,062
2	1	153	3	2º	1	16	0,143
3	1	153	3	2º	1	12	0,107
4	1	153	3	2º	1	29	0,259
5	1	153	4	2º	1	6	0,054
6	1	153	4	2º	1	5	0,045
7	1	153	4	2º	1	5	0,045
8	1	153	4	2º	1	22	0,196
9	1	150	3	4º	1	28	0,245
10	1	150	3	4º	1	15	0,131
11	1	150	3	4º	1	5	0,044
12	1	150	4	4º	1	8	0,070
13	1	150	4	4º	1	32	0,280
14	1	150	4	4º	1	7	0,061
15	1,09	185	3	2º	0	22	0,259
16	1,09	185	3	2º	0	14	0,165
17	1,09	183	3	2º	0	61	0,710
18	1,09	183	3	2º	0	85	0,989
19	1,09	184	4	2º	0	10	0,117
20	1,09	183	4	2º	0	7	0,081
21	1,09	182	4	2º	0	32	0,370
22	1,09	182	4	2º	0	51	0,590
23	1,09	183	3	4º	0	48	0,559
24	1,09	183	3	4º	0	64	0,745
25	1,09	182	3	4º	0	16	0,185
26	1,09	183	4	4º	0	5	0,058
27	1,09	182	4	4º	0	6	0,069
28	1,09	181	4	4º	0	8	0,092

Tabela 28 – Dados do 1º experimento controlado - UFSCar

ID da Amostra	Complexidade Ciclomática	LOC	Tarefa	Tratamento	Tempo Total (Minutos)	Esforço
1	1,05	325	4	0	12	0,027
2	1,05	324	6	0	16	0,036
3	NA	NA	4	1	3	NA
4	NA	NA	6	1	5	NA
5	1,05	329	4	0	71	0,162
6	1,05	330	6	0	10	0,023
7	NA	NA	4	0	10	NA
8	1,05	330	6	0	16	0,037
9	NA	NA	4	1	11	NA
10	NA	NA	6	1	2	NA
11	NA	NA	4	0	18	NA
12	NA	NA	6	0	18	NA
13	NA	NA	4	1	6	NA
14	1,15	336	6	1	5	0,013
15	NA	NA	4	1	8	NA
16	1,15	336	6	1	11	0,028
17	NA	NA	4	1	12	NA
18	1,15	336	6	1	8	0,020
19	NA	NA	4	0	8	NA
20	NA	NA	6	0	12	NA
21	NA	NA	4	0	8	NA
22	1,05	324	6	0	7	0,016
23	NA	NA	4	1	11	NA
24	1,15	336	6	1	13	0,033
25	NA	NA	4	0	20	NA
26	NA	NA	6	0	10	NA
27	NA	NA	4	1	15	NA
28	1,15	336	6	1	37	0,094
29	NA	NA	4	0	60	NA
30	1,05	326	6	0	5	0,011
31	NA	NA	4	0	40	NA
32	1,05	325	6	0	20	0,045
33	NA	NA	4	1	29	NA
34	1,15	336	6	1	13	0,033
35	NA	NA	4	0	22	NA
36	1,05	330	6	0	11	0,025
37	NA	NA	4	0	12	NA
38	1,05	327	6	0	13	0,029
39	NA	NA	4	0	14	NA
40	1,05	325	6	0	8	0,018
41	NA	NA	4	1	7	NA
42	NA	NA	6	1	3	NA

43	NA	NA	4	0	19	NA
44	1,05	326	6	0	60	0,136
45	NA	NA	4	1	56	NA
46	NA	NA	6	1	15	NA
47	NA	NA	4	0	26	NA
48	NA	NA	6	0	19	NA
49	1,05	327	4	0	65	0,147
50	1,05	329	6	0	49	0,112
51	1,15	336	4	1	23	0,059
52	NA	NA	6	1	26	NA
53	NA	NA	4	0	12	NA
54	1,05	326	6	0	21	0,047
55	NA	NA	4	0	14	NA
56	1,05	326	6	0	19	0,043
57	NA	NA	4	1	24	NA
58	1,15	336	6	1	25	0,064
59	NA	NA	4	1	20	NA
60	1,15	336	6	1	23	0,059
61	NA	NA	4	0	41	NA
62	1,05	334	6	0	9	0,021
63	NA	NA	4	0	21	NA
64	1,05	326	6	0	10	0,023
65	NA	NA	4	1	15	NA
66	1,15	336	6	1	17	0,043
67	NA	NA	4	0	59	NA
68	1,05	330	6	0	27	0,062
69	1,05	327	4	0	13	0,029
70	1,05	326	6	0	10	0,023
71	NA	NA	4	0	13	NA
72	1,05	326	6	0	8	0,018
73	NA	NA	4	0	23	NA
74	1,05	325	6	0	14	0,032
75	NA	NA	4	1	12	NA
76	1,15	336	6	1	28	0,071
77	1,05	329	4	0	23	0,052
78	1,05	327	6	0	4	0,009
79	NA	NA	4	0	28	NA
80	1,05	325	6	0	9	0,020
81	NA	NA	4	0	11	NA
82	1,05	329	6	0	5	0,011
83	NA	NA	4	1	33	NA
84	1,15	336	6	1	29	0,074
85	NA	NA	4	0	23	NA
86	NA	NA	6	0	16	NA
87	NA	NA	4	0	37	NA

88	1,05	327	6	0	26	0,059
89	NA	NA	4	1	17	NA
90	1,15	336	6	1	35	0,089
91	NA	NA	4	0	13	NA
92	NA	NA	6	0	11	NA
93	NA	NA	4	1	25	NA
94	1,15	336	6	1	5	0,013
95	NA	NA	4	0	50	NA
96	1,05	326	6	0	6	0,014
97	NA	NA	4	0	18	NA
98	NA	NA	6	0	33	NA
99	1,05	329	4	0	29	0,066
100	1,05	330	6	0	17	0,039
101	NA	NA	4	1	11	NA
102	NA	NA	6	1	5	NA
103	NA	NA	4	0	29	NA
104	1,05	323	6	0	28	0,063
105	NA	NA	4	0	24	NA
106	1,05	334	6	0	44	0,102
107	NA	NA	4	1	7	NA
108	1,15	336	6	1	6	0,015
109	NA	NA	4	0	12	NA
110	NA	NA	6	0	74	NA
111	NA	NA	4	1	14	NA
112	1,15	336	6	1	19	0,048
113	1,15	336	4	1	8	0,020
114	NA	NA	6	1	8	NA
115	NA	NA	4	0	38	NA
116	1,05	328	6	0	12	0,027
117	NA	NA	4	1	17	NA
118	1,15	336	6	1	32	0,082
119	NA	NA	4	0	15	NA
120	NA	NA	6	0	21	NA
121	NA	NA	4	0	33	NA
122	1,05	328	6	0	19	0,043
123	NA	NA	4	0	41	NA
124	1,05	334	6	0	9	0,021

Tabela 29 – Dados do 2º experimento controlado - UFSCar

ID da Amostra	Complexidade Ciclomática	LOC	Tarefa	Tratamento	Tempo Total (Minutos)	Esforço
1	1,13	722	2	0	42	0,226
2	1,09	1079	3	0	43	0,334
3	1,13	715	2	0	51	0,272

4	1,09	1067	3	0	38	0,292
5	NA	NA	2	1	2	NA
6	NA	NA	3	1	9	NA
7	1	19	2	1	6	0,001
8	1	24	3	1	3	0,000
9	1	19	2	1	2	0,000
10	1	24	3	1	3	0,000
11	1,15	738	2	0	72	0,404
12	1,12	1112	3	0	5	0,041
13	1	19	2	1	6	0,001
14	1	24	3	1	3	0,000
15	1	14	2	1	8	0,001
16	1	14	3	1	5	0,000
17	NA	NA	2	0	46	NA
18	NA	NA	3	0	19	NA
19	1,13	728	2	0	38	0,206
20	1,1	1091	3	0	29	0,230
21	1	19	2	1	13	0,002
22	1	24	3	1	8	0,001
23	1,13	718	2	0	44	0,236
24	1,09	1072	3	0	43	0,332
25	1,13	728	2	0	79	0,429
26	1,1	1092	3	0	10	0,079
27	1	19	2	1	10	0,001
28	1	24	3	1	17	0,003
29	1,13	727	2	0	29	0,157
30	1,1	1094	3	0	30	0,238
31	NA	NA	2	1	19	NA
32	NA	NA	3	1	13	NA
33	1,13	722	2	0	36	0,194
34	1,09	1078	3	0	54	0,419
35	1	15	2	1	15	0,001
36	1	15	3	1	5	0,000
37	1,13	735	2	0	95	0,521
38	1,1	1101	3	0	31	0,248
39	1	19	2	1	6	0,001
40	1	27	3	1	4	0,001
41	1,13	717	2	0	85	0,455
42	1,09	1073	3	0	47	0,363
43	1,15	754	2	0	55	0,315
44	1,13	1145	3	0	27	0,231
45	1,13	734	2	0	115	0,630
46	1,1	1101	3	0	33	0,264
47	1,13	733	2	0	77	0,421
48	1,1	1101	3	0	19	0,152

49	1,13	732	2	0	73	0,399
50	1,1	1100	3	0	56	0,447
51	1	18	2	1	4	0,000
52	1	24	3	1	3	0,000
53	NA	NA	2	0	27	NA
54	NA	NA	3	0	21	NA
55	1	14	2	1	10	0,001
56	1	14	3	1	3	0,000
57	1,13	731	2	0	98	0,535
58	1,1	1099	3	0	33	0,263
59	NA	NA	2	0	49	NA
60	NA	NA	3	0	45	NA
61	1,13	735	2	0	77	0,422
62	1,1	1103	3	0	34	0,272
63	1	18	2	1	10	0,001
64	1	24	3	1	3	0,000
65	1,13	726	2	0	36	0,195
66	1,1	1092	3	0	30	0,238
67	1,13	736	2	0	112	0,615
68	1,1	1097	3	0	33	0,263
69	1	19	2	1	22	0,003
70	1	24	3	1	7	0,001
71	1	18	2	1	17	0,002
72	1	24	3	1	6	0,001
73	1,13	734	2	0	93	0,509
74	1,1	1102	3	0	31	0,248
75	1	18	2	1	12	0,001
76	1	24	3	1	4	0,001
77	1,13	724	2	0	104	0,562
78	1,09	1083	3	0	45	0,351
79	1,15	737	2	0	22	0,123
80	1,12	1110	3	0	9	0,074

CÓDIGOS DAS CAPABILITIES

D.1 Códigos das Capacidades

Neste apêndice, são apresentados todos os códigos de exemplo das capacidades do Saucelabs, Browserstack e Testingbot, extraídos do livro de Sams (2015).

Código-fonte 2 – Exemplo de código de capacidades do Saucelabs (SAMS, 2015)

```
1
2 public class Saucelabs {
3
4     private WebDriver driver;
5     @Parameters({"username", "key", "browser", "browserVersion"
6         })
7     @BeforeMethod
8     public void setUp(@Optional("yourusername") String username
9         ,
10         @Optional("youraccesskey") String key,
11         @Optional("iphone") String browser,
12         @Optional("5.0") String browserVersion,
13         Method method) throws Exception {
14         // Choose the browser, version, and platform to test
15         DesiredCapabilities capabilities = new
16         DesiredCapabilities();
17         capabilities.setBrowserName (browser);
18         capabilities.setCapability ("version", browserVersion);
19         capabilities.setCapability ("platform", Platform.MAC);
20         capabilities.setCapability ("name", method.getName());
```

```

18     // Create the connection to SauceLabs to run the tests
19     this.driver = new RemoteWebDriver(
20         new URL("http://" + username + ":" + key + "@ondemand.
           saucelabs.com:80/wd/hub"), capabilities);
21 }
22 }

```

Código-fonte 3 – Exemplo de código de capacidades do Browserstack (SAMS, 2015)

```

1 public class Browserstack {
2     public static final String USERNAME = "yourusername";
3     public static final String ACCESS_KEY = "youraccesskey";
4     public static final String URL = "http://" + USERNAME + ":"
      +
5     ACCESS_KEY + "@hub.browserstack.com/wd/hub";
6     private WebDriver driver;
7     @BeforeClass
8     public void setUp() throws Exception {
9         DesiredCapabilities caps = new DesiredCapabilities();
10        caps.setCapability("browser", "Firefox");
11        caps.setCapability("browser_version", "23.0");
12        caps.setCapability("os", "Windows");
13        caps.setCapability("os_version", "XP");
14        caps.setCapability("browserstack.debug", "true");
15        //This enable Visual Logs
16        driver = new RemoteWebDriver(new URL(URL), caps);
17    }
18 }

```

Código-fonte 4 – Exemplo de código de capacidades do Testingbot (SAMS, 2015)

```

1 public class Testingbot {
2     private WebDriver driver;
3     @BeforeClass
4     public void setUp() throws Exception {
5         DesiredCapabilitiescapabilities = DesiredCapabilities.
           firefox();
6         capabilities.setCapability ("version", "24");
7         capabilities.setCapability ("platform", Platform.WINDOWS
           );
8         capabilities.setCapability ("name", "testOnCloud");

```

```
9      capabilities.setCapability ("screenshot", true);
10     capabilities.setCapability ("screenrecorder", true);
11     driver = new RemoteWebDriver(
12         new URL
13         ("http://ClientKey:ClientSecret@hub.testingbot.com:4444/
14         wd/hub"),
15         capabilities);
16 }
```

TESTES AUTOMATIZADOS DA ABORDAGEM MULTI-TAAS E CONVENCIONAL

E.1 Automatização de Testes das abordagens Multi-TaaS e Convencional

Neste apêndice, são apresentados todos os testes automatizados utilizados nos experimentos controlados. Os códigos 5 e 6 referem-se aos testes automatizados de *Create, Read, Update and Delete* (CRUD) de Universidade implementados para a abordagem Multi-TaaS e para a abordagem convencional, respectivamente. Os códigos 7 e 8 foram utilizados para a geração do relatório de execução e defeitos das abordagens Multi-TaaS e convencional, respectivamente.

A escrita dos testes automatizados ocorreu por meio do Selenium Webdriver e do *framework* JUnit, tanto na abordagem Multi-TaaS, quanto na abordagem convencional. Os testes automatizados são exatamente os mesmos para ambas as abordagens (CRUD), considerando ainda que os casos de testes e a sequência de comandos são exatamente iguais em ambas as abordagens. Além disso, antes do início da execução de cada teste, são inseridos os dados de teste diretamente no banco de dados e após a execução dos testes, os dados são removidos garantindo, assim, a independência entre os testes e mantendo a consistência do banco de dados.

Na abordagem convencional, foi utilizado um mecanismo de carga e descarga semelhante ao do grupo que utilizou a abordagem Multi-TaaS para que os testes fossem exatamente os mesmos, porém escritos de formas diferentes. A Multi-TaaS utilizou o padrão de escrita de testes automatizados, enquanto a abordagem convencional utilizou apenas a forma tradicional. É importante destacar que não foi avaliado o processo de automatização dos testes entre ambas

as abordagens, devido ao aumento da carga horária do treinamento e da complexidade final do experimento.

Código-fonte 5 – Testes de CRUD de Universidade - Abordagem Multi-TaaS

```
1 @RunWith(PluginRunner.class)
2 public class UniversidadesSystemTest extends AbstractCenario
3 {
4     @Test
5     @Carga(classe = UniversidadeCarga.class)
6     public void listarUniversidadesCadastradas() {
7         UsuarioEm(AplicacaoPage.class).deve().acessarMenu("
8             Universidades");
9         UsuarioEm(ListarUniversidadesPage.class).deve().
10            verificarQueQuantidadeUniversidades().igual(10);
11         UsuarioEm(ListarUniversidadesPage.class).deve().
12            verificarQueTextoDoRodapeDaTabelaDeUniversidades().igual("
13                10 universidades");
14    }
15
16    @Test
17    @Carga(classe = UniversidadeCarga.class)
18    public void adicionarUmaUniversidade() {
19        UsuarioEm(AplicacaoPage.class).deve().acessarMenu("
20            Universidades>Nova Universidade");
21        UsuarioEm(IncluirUniversidadePage.class).deve().
22            informarNome("Universidade Paulista").informarCidade("Sao
23                Paulo - SP").informarEndereco("Rua Jardim Cris, $^o$ 40,
24                Santo Amaro")
25            .informarCEP("13572-000").salvar();
26        UsuarioEm(ListarUniversidadesPage.class).deve().
27            verificarQueUniversidadeComNomeECidade("Universidade
28                Paulista", "Sao Paulo - SP").existe();
29    }
30
31    @Test
32    @Carga(classe = UniversidadeCarga.class)
33    public void removerUmaUniversidade() {
34        UsuarioEm(AplicacaoPage.class).deve().acessarMenu("
35            Universidades");
```

```
25     UsuarioEm(ListarUniversidadesPage.class).deve().
        removerUniversidade("Universidade Federal de Goiás (UFG)",
            "Goiânia - GO");
26     UsuarioEm(ListarUniversidadesPage.class).deve().
        verificarQueUniversidadeComNomeECidade("Universidade
            Federal de Goiás (UFG)", "Goiânia - GO").naoExiste();
27 }
28
29 @Test
30 @Carga(classe = UniversidadeCarga.class)
31 public void deveEditarUmaUniversidade() {
32     UsuarioEm(AplicacaoPage.class).deve().acessarMenu("
        Universidades");
33     UsuarioEm(ListarUniversidadesPage.class).deve().
        acessarUniversidadeComNomeECidade("Universidade Federal de
            Goiás (UFG)", "Goiânia - GO");
34     UsuarioEm(EditarUniversidadePage.class).deve().
        alterarNomePara("Universidade Federal da Bahia").
        alterarCidadePara("Salvador - BA").alterarEnderecoPara("
            Campus Universitário, $^o$ 127")
35         .alterarCEPPara("40210-730").salvar();
36     UsuarioEm(ListarUniversidadesPage.class).deve().
        verificarQueUniversidadeComNomeECidade("Universidade
            Federal de Goiás (UFG)", "Goiânia - GO").naoExiste();
37     UsuarioEm(ListarUniversidadesPage.class).deve().
        verificarQueUniversidadeComNomeECidade("Universidade
            Federal da Bahia", "Salvador - BA").existe();
38 }
39
40 }
```

Código-fonte 6 – Testes de CRUD de Universidade - Abordagem Convencional

```
1 public class UniversidadesSystemTest{
2
3     private UniversidadesPage universidadesPage;
4     private UniversidadeCarga cargaUniversidade;
5
6     @Before
7     public void inicializa() throws Exception {
```

```
8      // Cria uma instância da UniversidadePage
9      universidadesPage = new UniversidadesPage();
10     // A UniversidadeCarga é responsável pela carga e
    descarga dos dados de teste
11     cargaUniversidade = new UniversidadeCarga();
12     cargaUniversidade.realizaCarga();
13     universidadesPage.visitaListarUniversidades();
14 }
15
16 @Test
17 public void deveAdicionarUmaUniversidade() {
18     universidadesPage.novo().cadastrar("Universidade Paulista
19     ", "São Paulo - SP", "Rua Jardim Cris, nº 40, Santo
    Amaro", "13572-000");
20     assertTrue(universidadesPage.existeNaListagem("
21     Universidade Paulista", "São Paulo - SP"));
22 }
23
24 @Test
25 public void deveRemoverUmaUniversidade() {
26     universidadesPage.novo().cadastrar("Universidade Paulista
27     ", "São Paulo - SP", "Rua Jardim Cris, nº 40, Santo Amaro
    ", "13572-000");
28     assertTrue(universidadesPage.existeNaListagem("
29     Universidade Paulista", "São Paulo - SP"));
30     universidadesPage.deletaUniversidadeNaUltimaPosicao();
31     assertFalse(universidadesPage.existeNaListagem("
32     Universidade Paulista", "São Paulo - SP"));
33 }
34
35 @Test
36 public void deveListarUniversidades() {
37     int quantidadeDeUniversidades = universidadesPage.
38     getRegistrosDeUniversidades().size();
39     assertThat(quantidadeDeUniversidades, is(10));
40     assertThat(universidadesPage.
41     getTextoDoRodapeDaTabelaDeUniversidades(), is(
42     quantidadeDeUniversidades + " universidades"));
43 }
```



```
36
37  @Test
38  public void deveEditarUmaUniversidade() {
39      universidadesPage.novo().cadastrar("Universidade Paulista
40      ", "São Paulo - SP", "Rua Jardim Cris, $~o$ 40, Santo
41      Amaro", "13572-000");
42      universidadesPage.altera().para("Universidade Federal da
43      Bahia", "Salvador - BA", "Campus Universitário, $~o$ 127",
44      " 40210-730");
45      assertFalse(universidadesPage.existeNaListagem("
46      Universidade Paulista", "São Paulo - SP"));
47      assertTrue(universidadesPage.existeNaListagem("
48      Universidade Federal da Bahia", "Salvador - BA"));
49  }
50 }
```

Código-fonte 7 – Teste para geração de relatórios - Abordagem Multi-TaaS

```
1 package br.usp.icmc.test;
2
3 import org.junit.Test;
4
5 import br.usp.icmc.automation.AutomacaoServiceSingleton;
6 import br.usp.icmc.servico.AutomacaoService;
7 import br.usp.icmc.utils.ReportUtils;
8
9 public class TesteRelatorio {
10
11     @Test
12     public void geraRelatorio(){
13         AutomacaoService automacaoService =
14             AutomacaoServiceSingleton.get();
15         ReportUtils.geraArquivoHTML("Teste_Relatorio.html",
16             automacaoService.getParametrosParaGeracaoArquivo());
17     }
18 }
```

```
15 }  
16  
17 }
```

Código-fonte 8 – Teste para geração de relatórios - Abordagem Convencional

```
1 package br.usp.icmc.test;  
2  
3 import org.junit.Test;  
4  
5 import br.usp.icmc.browserstack.service.BrowserstackService;  
6 import br.usp.icmc.saucelabs.service.SaucelabsService;  
7 import br.usp.icmc.testingbot.service.TestingbotService;  
8 import br.usp.icmc.util.ReportUtils;  
9  
10 public class TesteRelatorio {  
11  
12     @Test  
13     public void testeRelatorioSaucelabs(){  
14         SaucelabsService automacaoService = new SaucelabsService  
15             ();  
16         ReportUtils.geraArquivoHTML("Teste_Relatorio_Saucelabs.  
17             html", automacaoService.getParametrosParaGeracaoArquivo())  
18             ;  
19     }  
20  
21     @Test  
22     public void testeRelatorioBrowserstack(){  
23         BrowserstackService automacaoService = new  
24             BrowserstackService();  
25         ReportUtils.geraArquivoHTML("Teste_Relatorio_Browserstack  
26             .html", automacaoService.getParametrosParaGeracaoArquivo()  
27             );  
28     }  
29  
30     @Test  
31     public void testeRelatorioTestingbot(){  
32         TestingbotService automacaoService = new  
33             TestingbotService();  
34         ReportUtils.geraArquivoHTML("Teste_Relatorio_Testingbot.  
35             html", automacaoService.getParametrosParaGeracaoArquivo())  
36             ;  
37     }  
38 }
```

```
        html", automacaoService.getParametrosParaGeracaoArquivo())  
        ;  
28    }  
29  
30 }
```

ARQUIVOS DE CONFIGURAÇÃO DA ARQUITETURA MULTI-TAAS

F.1 Arquivos de Configuração da Arquitetura Multi-TaaS

Neste apêndice, são apresentados os diferentes arquivos de configuração `taas.config` necessários para as configurações das capacidades e geração de relatórios no Saucelabs, Browserstack e Testingbot. Todas essas configurações foram utilizadas nos experimentos controlados para a abordagem Multi-TaaS.

O exemplo de Código 9 apresenta o arquivo de configuração `startup.config` responsável pela inicialização do navegador e acesso à *url* principal da aplicação web a ser testada, a partir da arquitetura Multi-TaaS. Os Códigos 10, 11 e 12 apresentam os exemplos do arquivo de configuração `taas.config` para o Saucelabs, Browserstack e Testingbot, respectivamente. O arquivo de configuração `taas.config` é responsável também por armazenar as configurações das capacidades e demais informações necessárias para a migração do serviço de teste na nuvem. Por fim, o Código 13 apresenta um exemplo de arquivo de configuração `d.config` necessário para a conexão com o banco de dados da aplicação web.

Código-fonte 9 – Arquivo de configuração `startup.config`

```
1 #-----#
2 # Esse e um comentario - comentarios iniciam com o sinal de '#'
3
4 # Acesso a aplicacao
5 startup_url_base=https://cliente-mws-usp-app.herokuapp.com/
6
7 startup=true
```

```

8 startup_class=br.usp.icmc.funcionalidade.AplicacaoPage
9 startup_method=acessar
10
11 # Tratamento de exceção - Caso ocorra algum erro
12 exception_catch=true
13 exception_catch_class=br.usp.icmc.funcionalidade.
    AplicacaoPage
14 exception_catch_method=acessar
15 #-----#

```

Código-fonte 10 – Arquivo de configuração taas.config do Saucelabs

```

1 ### SAUCELABS configuration file ###
2
3 # Provider Information
4
5 taas_service=saucelabs
6 taas_user=<user saucelabs>
7 taas_api_key=<valor-api-key>
8 taas_token=<token>
9
10 # Test Information
11
12 taas_build_name=Build_Saucelabs
13 taas_test_name= Teste Saucelabs
14 taas_operational_system=linux
15 taas_operating_system_version=
16 taas_browser_name=firefox
17 taas_browser_version=45.0
18 taas_screen_resolution=1024x768

```

Código-fonte 11 – Arquivo de configuração taas.config do Browserstack

```

1 ### BROWSERSTACK configuration file ###
2
3 # Provider Information
4
5 taas_servico=browserstack
6 taas_user=<user browserstack>
7 taas_api_key=<valor-api-key>
8 taas_token=<token>

```

```
9
10 # Test Information
11
12 taas_build_name= Build_Browserstack
13 taas_test_name= Teste Browserstack
14 taas_operational_system=Windows
15 taas_operating_system_version=10
16 taas_browser_name=IE
17 taas_browser_version=11.0
18 taas_screen_resolution=1280x1024
```

Código-fonte 12 – Arquivo de configuração taas.config do Testingbot

```
1 ### TESTINGBOT configuration file ###
2
3 # Provider Information
4
5 taas_servico=testingbot
6 taas_user=<user testingbot>
7 taas_api_key=<valor-api-key>
8 taas_token=<token>
9
10 # Test Information
11
12 taas_build_name= Build_Testingbot
13 taas_test_name= Teste Testingbot
14 taas_operational_system=yosemite
15 taas_operating_system_version=
16 taas_browser_name=chrome
17 taas_browser_version=52
18 taas_screen_resolution=2560x1600
```

Código-fonte 13 – Arquivo de configuração bd.config

```
1 bd_url=jdbc:mysql://us-cdbr-iron-east-04.cleardb.net:3306/
    heroku_b244805e8ae1bab
2 bd_user=<user>
3 bd_password=<password>
```

CONFIGURAÇÃO DAS CAPABILITIES UTILIZADAS NA ABORDAGEM CONVENCIONAL

G.1 Configuração das Capacidades utilizadas na Abordagem Convencional

Neste apêndice, são apresentados os códigos das capacidades do Saucelabs, Browserstack e Testingbot utilizadas no experimento controlado para a abordagem convencional.

Código-fonte 14 – Exemplo de código das capacidades usadas no Saucelabs

```
1 public class WebDriverSingleton {
2
3     private static WebDriver driver;
4     public static WebDriver get() {
5         // SAUCELABS
6         if (driver == null) {
7             try {
8                 DesiredCapabilities caps = new DesiredCapabilities();
9                 caps.setCapability("build", "Build_Saucelabs");
10                caps.setCapability("name", "Teste Saucelabs");
11                caps.setCapability("platform", "linux");
12                caps.setCapability("browserName", "firefox");
13                caps.setCapability("version", "45.0");
14                caps.setCapability("screenResolution", "1024x768");
15                driver = new RemoteWebDriver(new URL("http://
```

```
    robot_notifica:fc466671-e7d7-4136-90b8-  
    c34cd896cbc4@ondemand.saucelabs.com:80/wd/hub"), caps);  
16     driver.manage().timeouts().implicitlyWait(5, TimeUnit.  
        SECONDS);  
17 }  
18 catch (MalformedURLException e) {  
19     e.printStackTrace();  
20 }  
21 }  
22 return driver;  
23 }  
24 }
```

Código-fonte 15 – Exemplo de código das capacidades usadas no Browserstack

```
1 public class WebDriverSingleton {  
2     private static WebDriver driver;  
3     public static WebDriver get() {  
4         // BROWSERSTACK  
5         if (driver == null) {  
6             try {  
7                 DesiredCapabilities caps = new DesiredCapabilities();  
8                 caps.setCapability("build", "Build_Browserstack");  
9                 caps.setCapability("name", "Teste Browserstack");  
10                caps.setCapability("os", "Windows");  
11                caps.setCapability("os_version", "10");  
12                caps.setCapability("browser", "IE");  
13                caps.setCapability("browser_version", "11.0");  
14                caps.setCapability("resolution", "1280x1024");  
15                caps.setCapability("browserstack.debug", "true");  
16                driver = new RemoteWebDriver(new URL("http://robot1:  
                RNABL8QENFVX3J8Ue4LN@hub-cloud.browserstack.com/wd/hub"),  
                caps);  
17                driver.manage().timeouts().implicitlyWait(5, TimeUnit.  
                    SECONDS);  
18            }  
19            catch (MalformedURLException e) {  
20                e.printStackTrace();  
21            }  
22        }
```

```
23     return driver;
24 }
25 }
```

Código-fonte 16 – Exemplo de código das capacidades usadas no Testingbot

```
1 public class WebDriverSingleton {
2     private static WebDriver driver;
3     public static WebDriver get() {
4         // TESTINGBOT
5         if (driver == null) {
6             try {
7                 DesiredCapabilities caps = new DesiredCapabilities();
8                 caps.setCapability("build", "Build_Testingbot");
9                 caps.setCapability("name", "Teste Testingbot");
10                caps.setCapability("platform", "yosemite");
11                caps.setCapability("browserName", "chrome");
12                caps.setCapability("version", "52");
13                caps.setCapability("screen-resolution", "2560x1600");
14                caps.setCapability("screenshot", true);
15                driver = new RemoteWebDriver(new URL("http://8
d401c92209cc477e4a20165f207770e:4125
dcf1abc275fba624f7a805087cce@hub.testingbot.com/wd/hub"),
caps);
16                driver.manage().timeouts().implicitlyWait(5, TimeUnit.
SECONDS);
17            }
18            catch (MalformedURLException e) {
19                e.printStackTrace();
20            }
21        }
22        return driver;
23    }
24 }
```