UNIVERSIDADE FEDERAL DO PAMPA

Yury Alencar Lima

Teasy Framework: Uma solução para testes automatizados em aplicações web

Alegrete - RS
2021

Yury Alencar Lima

Teasy Framework: Uma solução para testes automatizados em aplicações web

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Software da Universidade Federal do Pampa como requisito parcial para a obtenção do título de Bacharel em Engenharia de Software.

Orientador: Prof. PhD Elder de Macedo Rodrigues

Coorientador: Prof. PhD Rafael Alves Paes de Oliveira

Alegrete - RS
2021

SERVIÇO PÚBLICO FEDERAL

MINISTÉRIO DA EDUCAÇÃO

Universidade Federal do Pampa

YURY ALENCAR LIMA

TEASY FRAMEWORK: UMA SOLUÇÃO PARA TESTES AUTOMATIZADOS EM APLICAÇÕES WEB

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia de Software da Universidade Federal do Pampa, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Trabalho de Conclusão de Curso defendido e aprovado em: 08, de março de 2021.

Banca examinadora:

_____

Prof. Dr. Elder de Macedo Rodrigues

Orientador

Unipampa

_____

Prof. Dr. Rafael Alves Paes de Oliveira

Co-Orientador

UTFPR

Prof. Dr. Maicon Bernardino da Silveira

Unipampa

Prof. Dr. Fábio Paulo Basso

Unipampa

Assinado eletronicamente por **FABIO PAULO BASSO**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em 08/03/2021, às 11:25, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **Rafael Alves Paes de Oliveira**, **Usuário Externo**, em 08/03/2021, às 11:41, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **ELDER DE MACEDO RODRIGUES**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em 09/03/2021, às 09:45, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **MAICON BERNARDINO DA SILVEIRA**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em 09/03/2021, às 09:49, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

A autenticidade deste documento pode ser conferida no site https://sei.unipampa.edu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **0478023** e o código CRC **C4F268BC**.

Universidade Federal do Pampa, Campus Alegrete
Av. Tiarajú, 810 – Bairro: Ibirapuitã – Alegrete – RS CEP: 97.546-550

Telefone: (55) 3422-8400

Este trabalho é dedicado à todos que fizeram parte da minha trajetória,
me ajudando tanto com conhecimento quanto com apoio nos momentos difíceis.

AGRADECIMENTOS

Este trabalho é resultado de anos de pesquisa, então primeiramente agradeço a todos os envolvidos que me auxiliaram de alguma forma. Especialmente aos discentes Juliana Mareco e Samuel Modesto que fizeram parte diretamente da execução e elaboração da solução apresentada. Além dos discentes citados, agradeço aos docentes Elder Rodrigues, Maicon Bernardino e Fábio Basso, que sempre estiveram presentes solucionando possíveis dúvidas e possibilitando este trabalho ser concretizado. Também agradeço ao professor Rafael de Oliveira por aceitar o convite e fazer parte deste trabalho, apresentando novos pontos de vista para a solução. Além de todos citados agradeço a doutoranda Ildevana Rodrigues pelo apoio na etapa de validação, com soluções e processos que simplificaram esta parte do estudo.

Além dos envolvidos diretamente com a pesquisa agradeço meu pai Gilmar que sempre me apoiou durante a graduação, que mesmo de longe teve um papel crucial para a entrega deste trabalho, e servia de motivação nos tempos difíceis. Agradeço também a minha namorada Bruna Wellen que mesmo nos momentos de cansaço, sempre tinha alguma palavra de motivação e carinho para me fazer seguir e concretizar este trabalho. Por fim, mas não menos importante agradeço a todos os professores, alunos e amigos envolvidos durante a graduação todos tiveram uma grande influência tanto profissional quanto pessoal.

“Você nunca sabe que resultados virão da sua ação.
Mas se você não fizer nada, não existirão resultados.“
(Mahatma Gandhi)

RESUMO

O teste funcional é a solução mais utilizada dentro do ciclo de desenvolvimento para detectar *bugs* nos sistemas. Entre as abordagens existentes para a realização dos testes, a automação pode reduzir o esforço a médio e longo prazo. Isto acontece decorrente a possibilidade de reexecução do teste sem o auxílio de um testador. Com base nisto, a adoção da automatização pode reduzir o tempo gasto e os custos relacionados aos testes de regressão dentro de um processo de desenvolvimento, além de aumentar a confiabilidade e a qualidade do *software*. Entretanto, quando ocorrem atrasos no desenvolvimento ou problemas durante alguma etapa anterior, o tempo para a criação dos testes é impactado ocasionando automações com problemas ou até mesmo a verificação manual das funcionalidades. A partir disto foram desenvolvidas ferramentas e linguagens específicas de domínio com o intuito de inserir os testes em outras etapas como, por exemplo, a etapa de análise, o que possibilita a automação antes ou durante o desenvolvimento. Apesar disso, o cenário ágil e mudança constante dos requisitos impacta nesta prática, além da dificuldade da manutenção e armazenamento dos testes automatizados, tendo em vista que na próxima versão do sistema um teste antigo pode ficar obsoleto por causa de modificações nos requisitos. Baseado nisto foi investigado dentro da literatura através de um mapeamento sistemático e um *snowballing* todas as linguagens específicas de domínio para testes funcionais, com o intuito de coletar os benefícios, limitações e tendências de cada linguagem. Após a análise dos resultados, foi definida a proposta do Teasy, um *framework* para testes funcionais com o foco na criação uma linguagem de domínio específico baseada nos benefícios, tendências e redução dos problemas relatados nos estudos. O *Teasy Framework* é composto por uma estrutura de testes escaláveis (*Teasy Structure*), um gerador de sequências (*Teasy Generator*) e uma linguagem específica de domínio para testes em aplicações *web* (*Teasy Language*). A *Teasy Language* tem o objetivo de prover o máximo de reúso, geração automática dos testes e facilidade no uso. Devido a comparações realizadas entre os tipos de linguagens a *Teasy Language* foi definida como projecional, o que aumenta sua produtividade e reduz o tempo necessário para a especificação do sistema a ser testado. Com o intuito de avaliar o *framework* foram realizados um quasi-experimento e uma avaliação de usabilidade com especialistas. Assim, foi possível comparar a representatividade, eficácia e usabilidade do *framework*.

Palavras-chave: Teste funcional. Framework de testes. Linguagem de domínio específico. Automação de testes. Qualidade de software.

ABSTRACT

Functional testing is the most used solution within the development cycle to detect bugs in systems. Among the existing approaches to conducting tests, automation can reduce effort in the medium and long term, this is due to the possibility of re-running the test without the aid of a tester. Based on this, the adoption of automation can reduce the time and costs related to regression testing within a development process, in addition to increasing the reliability and quality of software. However, when development delays or problems occur during a previous step, the time for creating the tests is impacted, causing automation with problems or even the manual verification of functionalities. From this, domain specific tools and languages were developed in order to insert the tests in other stages, such as, for example, the analysis stage, which allows automation before or during development. In spite of this, the agile scenario and constant changing requirements have an impact on this practice, in addition to the difficulty of maintaining and storing automated tests, considering that in the next version of the system an old test may become obsolete due to changes in requirements. Based on this, it was investigated within the literature through systematic mapping and snowballing all domain specific languages for functional tests, in order to collect the benefits, limitations and trends of each language. After analyzing the results, the Teasy proposal was defined, a framework for functional tests with a focus on creating a specific domain language based on the benefits, trends and reduction of the problems reported in the studies. The Teasy Framework is composed of a scalable test structure (Teasy Structure), a sequence generator (Teasy Generator) and a domain specific language for testing applications web (Teasy Language). Teasy Language aims to provide maximum reuse, automatic test generation and ease of use. Due to comparisons made between the types of languages, Teasy Language was defined as design, which increases its productivity and reduces the time required for the specification of the system to be tested. In order to evaluate the framework, a quasi-experiment and usability evaluation with specialists were carried out. Thus, it was possible to compare the representativeness, effectiveness and usability of the framework.

Keywords: Functional testing. Testing framework. Domain-specific language. Testing automation. Software quality.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

CE Critério de Exclusão

CI Critério de Inclusão

CQ Critério de Qualidade

ID Identificador

M Métrica

N Não

N/A Não se aplica

P Parcial

QP Questão de Pesquisa

R Requisito

S Sim

LISTA DE SIGLAS

DSL *Domain-Specific Language*

EMF Eclipse Modeling Framework

FSM Máquina de Estados Finitos

GMF Graphical Modeling Framework

GPL Linguagem de Propósito Geral

HTML HyperText Markup Language

IDE Integrated Development Environment

IHC Interação Humano Computador

JSON *JavaScript Object Notation*

LW *Language Workbench*

MBT *Model-based Testing*

MPS *Meta Programming System*

OCL *Object Constraint Language*

PICOC Population Intervention Comparison Outcome and Context

SMS Mapeamento Sistemático de Literatura

SPA *Single Page Application*

SUT Sistema Sob Teste

W3C World Wide Web Consortium

XML Extensible Markup Language

SUMÁRIO

1 INTRODUÇÃO

A demanda por *software* continua crescendo devido a concorrência de mercado e as novas *startups* estão cada vez mais preocupadas com a entrega rápida dos seus produtos (SOUZA et al., 2017). A necessidade de uma entrega rápida pode impactar diretamente na qualidade do produto, tendo em vista que os curtos prazos afetam geralmente a etapa de verificação do *software*. A verificação de um produto de *software* frequentemente é realizada através dos testes de software, que podem averiguar diversos aspectos do sistema como segurança, desempenho, funcionalidade entre outros (DELAMARO; JINO; MALDONADO, 2013). Dentre estes aspectos, o funcional é o que determina se o sistema está fazendo o que ele deveria fazer de acordo com o informado pelo cliente, e habitualmente é o aspecto mais testado de um produto de *software* (DELAMARO; JINO; MALDONADO, 2013).

O teste funcional é o responsável por analisar as funcionalidades de um sistema e pode ser realizado de maneira manual ou automatizada. A automação é vista como a melhor alternativa, pois além de possibilitar o teste de regressão, pode garantir que os testes sejam realizados de maneira regular e consistente, auxiliando na rápida detecção de erros (MOREIRA et al., 2017). Entretanto, a implementação dos *scripts* automatizados é realizada de modo manual e, principalmente, a manutenção destes *scripts* são caras. Ocasionando assim um problema em processos de desenvolvimento ágil, considerando que as adaptações frequentes podem afetar os testes de regressão que deveriam estar sempre disponíveis (TÖRSEL, 2013).

O *Model-based Testing* (MBT) pode ser uma possibilidade para mitigar o problema, pois, é uma abordagem para geração automatizada de testes baseados em modelos predefinidos, melhorando assim a eficiência no design dos casos de teste. Assim, basta alterar somente o modelo e todos os casos de teste podem ser gerados novamente, melhorando a manutenabilidade dos casos de teste (UTTING; LEGEARD, 2010). Uma alternativa para o uso da abordagem MBT é através de uma *Domain-Specific Language* (DSL). As DSLs são linguagens que tem como objetivo representar um domínio específico, aumentando a produtividade e engajamento dos envolvidos, possibilitando ou não a geração de artefatos desde que criadas para esta finalidade (FOWLER, 2010).

Diante disto, este estudo tem como intuito propor um *framework* para testes funcionais composto por: uma DSL, uma estrutura escalável para os projetos de teste, facilitando a manutenibilidade do código final e também um gerador de sequências, a fim de aumentar a cobertura e reduzir o esforço necessário para realizar a etapa de qualidade em aplicações reais. Além disso o *framework* é voltado para o uso em diversas abordagens de desenvolvimento, mas essa proposta tem como foco aplicações *Web*, entretanto poderá também ser utilizada em outros tipos de aplicações sem a geração de código.

1.1 Motivação

No cenário competitivo de desenvolvimento de *software* e de metodologias ágeis, o foco das empresas está na implementação e contato com o cliente (SOUZA et al., 2017). Entretanto, em alguns casos não é possível manter um contato contínuo durante o desenvolvimento para realizar as validações necessárias. Assim, a qualidade do produto nestes cenários pode ser impactada tendo em vista que o tempo destinado aos testes geralmente é reduzido.

Com o intuito de manter o foco no produto sem perder a qualidade em praticamente todos os cenários possíveis, as empresas carecem de uma maneira simplificada de realizar as verificações dos seus produtos de *software* exigindo um menor esforço e reduzindo o tempo necessário. Assim, com a intenção de fornecer uma solução para esta problemática a Teasy foi idealizada. Possuindo como foco a geração de casos e *scripts* de testes automatizados para o sistema descrito através da sua sintaxe.

1.2 Objetivo

O objetivo geral do *framework* Teasy é "Aumentar a qualidade de um produto de *software* exigindo um menor esforço e tempo". Além deste objetivo geral este trabalho e a linguagem possuem outros cinco objetivos específicos.

Objetivo Específico 1: Realizar um mapeamento dos benefícios, problemas e tendências entre as DSLs existentes. Possibilitando a elaboração de uma solução que atenda as necessidades tanto da indústria quanto da academia.

Objetivo Específico 2: Possibilitar o reúso de componentes dentro da linguagem, reduzindo o esforço e tempo necessários para realizar a descrição do sistema a ser testado.

Objetivo Específico 3: Realizar a geração automática de casos de teste para sistemas *web* e também a geração de *scripts* de teste de maneira automatizada com base na descrição do sistema em um gerador *web*.

Objetivo Específico 4: Prover uma estrutura escalável e de alta manutenabilidade dos artefatos de teste, dado que não são necessários cuidados para manter cada caso ou *script* de teste, somente a descrição do sistema.

Objetivo Específico 5: Avaliar a *Teasy Framework* com acadêmicos e profissionais da indústria com o propósito de coletar informações sobre a praticidade no uso e comparação com outras soluções já existentes.

1.3 Metodologia

Com o objetivo de desenvolver uma DSL para teste funcional que permitisse o menor esforço possível sem perder a qualidade dos testes gerados, tornou-se necessária a execução de um Mapeamento Sistemático de Literatura (SMS), como este estudo geral-

mente é realizado por mais de um revisor, contou-se com a colaboração da aluna Juliana Mareco. Este SMS teve como principal objetivo detectar todas as linguagens para testes funcionais existentes juntamente com seus principais benefícios, limitações e tendências. Após a execução deste mapeamento foi detectada uma ameaça a validade que poderia impactar na análise dos estudos, sendo esta a possibilidade de existirem outras linguagens em outras bibliotecas digitais não utilizadas no processo de busca dos estudos.

Com base nisto também foi executado um *Snowballing*, utilizando como estudos iniciais os artigos selecionados após a finalização do SMS. Assim, mitigando a ameaça encontrada e possibilitando uma análise mais completa dos estudos. O resultado da análise originou tanto em um conjunto de linguagens com seus principais benefícios, limitações e tendências quanto em um conjunto de ferramentas que podem ser utilizadas para a implementação de uma DSL.

Utilizando como base as linguagens e ferramentas encontradas, a *Teasy Language* foi definida e implementada utilizando o *Meta Programming System* (MPS) (JETBRAINS, 2014) como *Language Workbench* (LW) para o desenvolvimento. Ao final desta atividade foi desenvolvido também um gerador de sequências de teste com base nos artefatos de saída gerados pela *Teasy Language*, viabilizando assim, o uso do MBT através da especificação do sistema realizada na linguagem. Com o objetivo de todo o código final gerado tanto pela *Teasy Language* quanto pelo gerador (*Teasy Generator*) foi idealizada a *Teasy Structure* uma estrutura escalável para a execução dos *scripts* de teste executáveis.

A fim de verificar e validar o *Teasy Framework* foi idealizado um *quasi*-experimento e um *survey* com especialistas em usabilidade. Dessa forma foi possível avaliar se o *framework* é representativo para o domínio e se é eficaz para criar testes funcionais para aplicações *web* de acordo com os objetivos específicos, além de apresentar também quais limitações a solução apresenta no momento. O *survey* teve foco na usabilidade da *Teasy Language*, assim viabilizando uma análise sobre da linguagem de acordo com as heurísticas de usabilidade propostas por (NIELSEN; MOLICH, 1990).

1.4 Principais Contribuições

Este trabalho possui como principal contribuição o Teasy, um *framework* de testes funcionais de *software* com o foco na redução do esforço e tempo destinados a implementação e manutenção dos *scripts* de teste automatizados para aplicações *web*. Mitigando assim problemas relacionados a definição do sistema e geração automática de casos de teste. Outra contribuição é o mapeamento e análise das DSLs existentes para testes funcionais e quais seus principais benefícios, limitações e tendências.

Outra contribuição é a avaliação do *framework*, composta pelo *quasi*-experimento e o *survey* realizado com os especialistas em usabilidade. Viabilizando assim, uma análise sobre o impacto que a solução pode ter tanto na indústria quanto na academia. Por fim essa avaliação pode apresentar os benefícios e quais limitações necessitam ser corrigidas

em trabalhos futuros.

1.5 Organização do Trabalho

A sequência deste estudo foi estruturada da seguinte forma:

- O Capítulo 2 possui toda a fundamentação teórica necessária para a concretização do estudo, apresentando conceitos de testes de *software* e sobre linguagens de domínio específico, além das lições do capítulo.
- O Capítulo 3 possui todos os trabalhos relacionados ao estudo juntamente com os detalhes do SMS e do *Snowballing*, além das lições do capítulo.
- O Capítulo 4 apresenta os detalhes relacionados ao *Teasy Framework*, além das lições do capítulo.
- Os Capítulos 5 e 6 apresentam as validações realizadas no *framework* em conjunto com as lições de cada capítulo.
- O Capítulo 7 apresenta as considerações finais do estudo.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados os principais conceitos relacionados ao tema deste estudo, são eles: Testes funcionais, Teste Baseado em Modelos, *Domain-Specific Languages* (DSLs) e o *Meta Programming System* (MPS).

2.1 Testes de *Software*

Nesta seção serão apresentados todos os conceitos relacionados à testes de *software* utilizados para a concepção e implementação do *Teasy Framework*, sendo eles: testes funcionais e testes baseado em modelos.

2.1.1 Testes Funcionais

Os testes de funcionais tem como objetivo encontrar *bugs* dentro de um produto de software (DELAMARO; JINO; MALDONADO, 2013). Esta prática aumenta a confiabilidade relacionada ao produto a ser entregue e auxilia na verificação do *software* em relação aos requisitos do cliente (RIOS et al., 2007). Nos testes funcionais, um produto de *software* é considerado uma caixa-preta, ou seja, não são verificados os códigos-fonte e sim seu comportamento (DELAMARO; JINO; MALDONADO, 2013). Este comportamento é derivado dos requisitos e verificado através dos casos de teste, que são um conjunto de condições para a realização dos testes, possuindo como principais elementos as entradas, ações e resultados esperados (MYERS; SANDLER; BADGETT, 2011). Assim, possibilitando a detecção de *bugs* funcionais antes de o produto ser entregue, reduzindo os custos relacionados à correções após a entrega do *software*, sendo estes os mais altos do ciclo de vida do produto (DELAMARO; JINO; MALDONADO, 2013).

2.1.2 Teste Baseado em Modelos

O Teste Baseado em Modelo ou *Model-based Testing* (MBT), é uma abordagem para a criação de testes, que tem como objetivo aumentar o nível de abstração no desenvolvimento dos testes (UTTING; LEGEARD, 2010). Provendo assim maior entendimento do produto de *software* a ser testado e melhorando a especificação. Nesta abordagem é definido um modelo do produto de *software* que possui nele como deve ser o seu comportamento e dados relacionados. A partir destas informações são gerados casos de teste e *scripts* de testes automatizados. O MBT evita ambiguidades relacionadas à documentação textual, pode reduzir custos tendo em vista que os *bugs* podem ser detectados antes mesmo do produto ser desenvolvido já que necessita somente do modelo relacionado ao *software* e possibilita não programadores a elaborar testes automatizados devido à possibilidade de geração automática de *scripts* de teste (UTTING; LEGEARD, 2010).

2.2 Linguagem de Domínio Específico

Ao contrário das Linguagens de Propósito Gerais (GPLs) como JavaScript (NEGRINO; SMITH, 2001), Java (GOSLING; HOLMES; ARNOLD, 2005), dentre outras, as DSLs são linguagens de computador que possuem o objetivo de fornecer suporte e solucionar problemáticas relacionadas a um determinado domínio (MERNIK; HEERING; SLOANE, 2005). As diferenças de uma GPL e uma DSL não são nítidas, tendo em vista que a linguagem pode possuir apenas alguns recursos específicos para o domínio, permitindo ser amplamente utilizada (FOWLER, 2010). Como uma DSL possui como principal objetivo resolver um problema de um respectivo domínio, geralmente a DSL não é capaz de resolver as adversidades que ocorram fora deste cenário, sendo considerada parte da Engenharia de Domínio (FOWLER, 2010). Com base nisto, o seu nível de abstração comumente é maior que uma GPL, facilitando o uso por parte dos especialistas do domínio e os aproximando da resolução dos problemas (ŽIVANOV; RAKIĆ; HAJDUKOVIĆ, 2008).

Mesmo que uma DSL possa aumentar a produtividade e engajamento dos especialistas de domínio, sua implementação não é uma atividade trivial. Além disto, o perfil dos especialistas que irão utilizar a linguagem podem influenciar no seu desenvolvimento e tipo. Com base nisto existem algumas maneiras de se implementar uma DSL que influenciam na definição dos seus tipos (FOWLER, 2010). Abaixo são apresentados os tipos de DSLs.

2.2.1 Linguagens Internas

As Linguagens Internas ou Linguagens Embarcadas são definidas e implementadas dentro de outra linguagem existente que geralmente é uma GPL (FOWLER, 2006). Como elas possuem relação com a GPL na qual foram criadas os especialistas do domínio que vão utilizá-las tem vantagem caso possuam um conhecimento prévio relacionado a respectiva GPL. Um exemplo deste cenário poderia ser uma linguagem “XUnit” para testes unitários que foi desenvolvida em Java, os programadores que possuem conhecimento em Java tendem a ter mais facilidade de adesão e compreensão da linguagem “XUnit”.

2.2.2 Linguagens Externas

As Linguagens Externas são DSLs que são independentes de outras linguagens ou GPLs, elas contêm seu próprio analisador e gerador, quando possuírem (FOWLER, 2010). Assim, o seu principal benefício é que o desenvolvedor pode elaborar uma sintaxe livre e especializada para o respectivo domínio, podendo inclusive aumentar o nível de abstração para um determinado contexto.

2.2.3 *Languages Workbenches*

As *Languages Workbenches* (LWs) são ferramentas que apoiam a implementação de DSLs fornecendo um ambiente de desenvolvimento (FOWLER, 2005). Este ambiente fornece um esquema destinado para definição de um modelo semântico, edição da DSL e realização de uma semântica comportamental por meio da interpretação e geração de código caso necessário. Resultando assim em uma maior facilidade e redução dos custos relacionados ao desenvolvimento de uma DSL (FOWLER, 2005). De acordo com o domínio a DSL ainda pode ser textual, gráfica, projecional ou até mesmo tabular, a escolha de qual tipo utilizar vai depender do respectivo domínio.

Partindo do domínio, é necessário escolher qual LW escolher, tendo em vista que existem soluções tanto pagas quanto *Open source*. Dentre os LWs gratuitos é possível citar o XText(BETTINI, 2016) utilizado para implementar linguagens textuais, Sirius (SIRIUS, 2019) que possibilita a criação de linguagens gráficas e por fim o MPS (JETBRAINS, 2019) que permite a criação de linguagens projecionais.

2.2.4 *Meta Programming System*

O MPS é uma LW *Open source* com o objetivo de projetar DSLs, sendo uma das mais consolidadas atualmente que faz uso do recurso *Projectional Editing* (FOWLER, 2010). Este recurso permite que os desenvolvedores elaborem seus próprios editores com o intuito de projetar a linguagem especificada proporcionando uma projeção que seja textual e gráfica ao mesmo tempo, se necessário (JETBRAINS, 2019). O MPS é desenvolvido pela JetBrains e inclui uma linguagem para auxiliar o desenvolvimento, denominada MPS *BaseLanguage*, que é baseada na linguagem de programação Java. Entretanto, com o tempo foram acrescidas novas linguagens que podem também ser utilizadas para a definição de uma linguagem como Extensible Markup Language (XML), C e JavaScript. Com o intuito de elaborar uma DSL dentro do MPS é necessário estabelecer os conceitos da linguagem e relacioná-los, após esta etapa é possível determinar um editor para tais conceitos e por fim estipular um gerador de artefatos caso necessário (JETBRAINS, 2019). O MPS também é baseado em árvore de sintaxe abstrata, ou seja, cada atributo necessita ser escolhido através das opções fornecidas para o usuário, assim o elemento selecionado passa a ser projetado (JETBRAINS, 2019). Além disto, a LW ainda provê um conjunto de personalizações para o editor no qual é possível simular o comportamento de Integrated Development Environments (IDEs) convencionais (JETBRAINS, 2019).

2.3 Lições do Capítulo

Com base nas informações apresentadas neste capítulo é possível verificar a importância do teste funcional de *software*, tanto para a redução de custos quanto para o aumento da confiabilidade do produto. A utilização de modelos para a definição destes

testes podem ser uma alternativa para facilitar e reduzir o tempo necessário para a criação dos testes automatizados, além de centralizar as informações em um único modelo. Caso este modelo seja representado através de uma DSL é possível uma maior abstração e sintaxe voltada para testadores aumentando sua compreensão e produtividade. O MPS é uma alternativa para definir esta DSL, possuindo a versatilidade na projeção, provendo a possibilidade de uso da representação textual e gráfica, se necessário, proporcionando também a geração automática de artefatos. Apresentando assim, uma justificativa e viabilizando o desenvolvimento da Teasy.

3 MAPEAMENTO SISTEMÁTICO DA LITERATURA

Com o objetivo de mapear os trabalhos relacionados da área e as necessidades do cenário atual foi realizado um Mapeamento Sistemático de Literatura (SMS). Este estudo teve como principal foco identificar as *Domain-Specific Languages* (DSLs) para testes funcionais existentes, bem como quais seus maiores benefícios, limitações e tendências do mercado. Com base neste SMS, foi detectada uma ameaça à validade relacionada a possibilidade de existirem outras linguagens que não estivessem presentes nas bibliotecas digitais buscadas, resultando assim na necessidade de execução de um *Snowballing*. Os demais detalhes relacionados ao SMS e o *Snowballing* estão descritos neste capítulo.

3.1 *Systematic Mapping Study*

O mapeamento é uma maneira de identificar, avaliar e interpretar as pesquisas relevantes para determinadas questões de pesquisa. Tendo em vista que o SMS busca evidenciar o que existe relacionado a um tema, este estudo geralmente é executado por mais de um revisor. Assim, contou-se com a colaboração da aluna Juliana Mareco para a execução deste estudo. Este SMS tem como objetivo detectar DSLs para testes funcionais de *software* e verificar todas as desvantagens, benefícios e tendências do cenário atual.

3.1.1 Planejamento

Um mapeamento sistemático tem como objetivo analisar os estudos relacionados a um determinado tópico de pesquisa. Assim, com o objetivo de realizar este mapeamento foi utilizado como base a abordagem proposta por (PETERSEN et al., 2008). Nesta abordagem o SMS é subdividido em três etapas sendo elas o planejamento, condução e a análise. Esta seção apresenta a primeira etapa do estudo, ou seja, o planejamento. Esta etapa tem como principal objetivo a identificação da necessidade do estudo, sendo esta a motivação para executar o SMS. Com o intuito de atingir este objetivo é necessária a criação de um protocolo que deve ser seguido garantindo assim a possibilidade de replicação do estudo.

3.1.1.1 Protocolo

Com o intuito de possibilitar a replicabilidade do estudo, o protocolo é composto por questões de pesquisa, estratégias para a condução, critérios de inclusão e exclusão, e critérios de qualidade. Cada uma destas informações são apresentadas abaixo e impactam diretamente no resultado final do estudo.

3.1.1.1.1 Escopo e Objetivo

De modo a evidenciar o principal foco do estudo é definido o seu escopo e objetivo. Neste caso o mapeamento sistemático em questão destina-se a mapear as DSLs para testes funcionais de *software* e seus recursos, desvantagens, benefícios e tendências. Possibilitando assim verificar os maiores déficits e benefícios do cenário atual e propor uma nova linguagem específica de domínio para testes funcionais de *software*, com a finalidade de atenuar as desvantagens encontradas e unir a maioria dos benefícios e tendências.

3.1.1.1.2 Estrutura das Questões de Pesquisa

As questões de pesquisa são as informações que após a análise dos estudos se deseja possuir. A fim de estruturá-las e elaborar as *strings* de busca com clareza foi proposto por (KITCHENHAM; CHARTERS, 2007) o uso do Population Intervention Comparison Outcome and Context (PICOC), que as organiza em população, intervenção, comparação, resultado e contexto. A seguir são apresentadas as respostas definidas para o PICOC neste estudo.

- **População**: Todos os estudos publicados que propõem ou utilizam uma DSL para teste funcional de *software*.
- **Intervenção**: As DSLs para testes funcionais de *software* e suas funcionalidades, limitações e benefícios.
- **Comparação**: Este item do PICOC não se aplica a este estudo, pois o objetivo é detectar limitações para mitigar e reunir o maior número de benefícios encontrados, além da verificação das possíveis tendências da área.
- **Saída**: Um conjunto de DSLs para testes funcionais de *software*, juntamente com suas desvantagens, benefícios e tendências.
- **Contexto**: Este estudo considera todos os contextos, sendo eles acadêmicos ou industriais.

3.1.1.1.3 Questões de Pesquisa

As questões de pesquisa definem as informações a serem identificadas nos estudos, os dados a serem extraídos e analisados. Dessa forma, neste estudo foram definidas as seguintes Questões de Pesquisa (QPs):

- **QP1**: Os artefatos gerados a partir da DSL (por exemplo, *scripts* e modelos gerados) são aplicados a quais domínios de *software*? Esta questão tem como intuito detectar os domínios para os quais as DSLs estão sendo desenvolvidas.

- **QP2**: Quais são as funcionalidades destas DSLs? Esta questão tem o objetivo de verificar quais funcionalidades são mais implementadas dentre as linguagens propostas.
- **QP3**: Quais tecnologias e notações foram usadas para o desenvolvimento das DSLs para testes funcionais? Com base nesta resposta será possível verificar quais tecnologias são mais utilizadas e recomendadas para o desenvolvimento de uma nova linguagem para testes funcionais de *software*.
- **QP4**: Quais técnicas/abordagens/métodos a DSL usa para garantir a cobertura dos testes, no caso de possuir a geração do *script*? Esta pergunta tem a finalidade de verificar quais são os métodos mais utilizados para garantir a cobertura do sistema a ser testado.
- **QP5**: Como os dados de entrada do Sistema Sob Teste (SUT) são representados nos modelos? Assim, é possível verificar como e quais informações são inseridas nas linguagens para possibilitar a geração e/ou especificação dos sistemas sob teste.
- **QP6**: Quais são os benefícios, desvantagens e tendências relacionadas ao uso da DSL para testes funcionais? Esta pergunta possibilita a análise da linguagem e em qual contexto é mais indicada e quais limitações devem ser solucionadas em uma nova linguagem para facilitar sua adesão.

3.1.1.1.4 Estratégia de Busca

Com o objetivo de realizar a busca foram utilizadas bibliotecas digitais, através dos mecanismos de buscas fornecidos por cada biblioteca. Estes mecanismos utilizam palavras-chave e maneiras de restringir estudos da área da computação. As bibliotecas digitais utilizadas no SMS foram: ACM Digital Library[1], Engineering Village (Compendex)[2], IEEE Xplore[3], ScienceDirect[4], SCOPUS[5] e SpringerLink[6].

As palavras-chave utilizadas para a elaboração da *string* de busca genérica podem ser visualizadas através da Tabela 1.

Com a finalidade de transformar as palavras-chave e seus sinônimos em uma *string* de busca, foram usados operadores booleanos sendo eles o "OR" para unificar os sinônimos e o "AND" para associar as palavras-chave. O resultado desta operação pode ser visto através da Figura 1. Entretanto, a *string* de busca apresentada não pode ser utilizada em todas as bibliotecas digitais. Tendo em vista que algumas bibliotecas possuem restrições diferentes. Assim, resultando em *strings* de buscas específicas para cada biblioteca digital.

[1] ACM: http://www.dl.acm.org
[2] Compendex: http://www.engineeringvillage
[3] IEEE Xplore: http://www.ieeexplore.ieee.org
[4] ScienceDirect: http://www.sciencedirect.com
[5] SCOPUS: http://www.scopus.com
[6] SpringerLink: http://www.link.springer.com

Tabela 1 – Palavras-Chave e Sinônimos

Palavra-Chave	**Sinônimos**
Domain-Specific Language	domain specific language
	domain-specific-language
	domain-specific modeling language
	domain-specific-modeling-language
	domain-specific-modeling language
	DSL
	DSML
Functional testing	black box test
	black-box test
	black box testing
	black-box testing
	functional test
	regression test
	regression testing
	acceptance test
	acceptance testing

Figura 1 – *String* de Busca Base

(Domain-Specific Language **OR** "domain specific language" **OR** "domain-specific-language" **OR** "domain-specific modeling language"**OR** "domain-specific-modeling-language" **OR** DSL **OR** DSML) **AND** ("Functional testing"**OR** "black box test"**OR** "black-box test"**OR** "black box testing"**OR** "black-box testing"**OR** "functional test"**OR** "regression test"**OR** "regression testing"**OR** "acceptance test"**OR** "acceptance testing")

3.1.1.1.5 Critérios de Inclusão e Exclusão

Os Critérios de Inclusão (CIs) e Critérios de Exclusão (CEs) são utilizados para definir se um respectivo estudo colabora com a resposta de uma ou mais questões de pesquisa definidas. Assim, para analisar os estudos encontrados através das *strings* de busca foram definidos os seguintes critérios.

- **CI1**: Estudos que apresentem explicitamente o uso ou a proposta de uma DSL para testes funcionais de *software*.
- **CE1**: Estudos escritos em outra linguagem diferente do inglês.
- **CE2**: Estudos que não apresentam explicitamente o uso ou a proposta de uma DSL para testes funcionais de *software*.

3.1.1.1.6 Critérios de Qualidade

Os Critérios de Qualidade (CQs) tem o objetivo verificar o nível de contribuição dos estudos selecionados em relação ao foco do mapeamento. Os CQs não possuem o intuito de avaliar a contribuição do estudo para a área de Engenharia de *Software*, somente para o objetivo do SMS. Assim, foram definidos os seguintes CQs:

CQ1. O estudo apresenta as principais funcionalidades da DSL?

Respostas - **Sim (S)**: O estudo apresenta todas as funcionalidades presentes na linguagem proposta ou utilizada; **Parcial (P)**: O estudo apresenta somente algumas das funcionalidades relacionadas à DSL proposta ou utilizada; **Não (N)**: O estudo não apresenta as funcionalidades da DSL.

CQ2. O estudo apresenta para quais domínios os artefatos, como *scripts* gerados pela DSL, podem ser aplicados?

Respostas - **S**: O estudo apresenta todos os domínios que os artefatos gerados possam ser aplicados; **P**: O estudo apresenta somente alguns usos dos artefatos gerados; **N**: O estudo não apresenta o domínio para qual os artefatos gerados possam ser utilizados.

CQ3. O estudo apresenta como os dados do SUT são inseridos e representados nos modelos?

Respostas - **S**: O estudo apresenta como todos os dados são inseridos e representados nos modelos; **P**: O estudo apresenta parcialmente em relação como os dados são inseridos os representados nos modelos; **N**: O estudo não apresenta como os dados são inseridos os representados nos modelos.

CQ4. O estudo apresenta benefícios, desvantagens e tendências relacionadas ao uso da DSL para testes funcionais?

Respostas - **S**: O estudo apresenta os benefícios, malefícios e tendências; **P**: O estudo apresenta somente parte dos benefícios, malefícios e tendências; **N**: O estudo não apresenta os benefícios, malefícios e tendências.

3.1.1.1.7 Processo de Seleção

A fim de selecionar os estudos relevantes para responder às questões de pesquisa apresentadas em 3.1.1.1.3 o processo de seleção foi dividido em sete etapas, onde parte destas etapas foram realizadas por dois pesquisadores. As etapas e os pesquisadores envolvidos nelas, são descritos a seguir:

Etapa 1: Busca nas bibliotecas digitais: para a concretização desta etapa foram criadas *strings* de busca para cada biblioteca digital utilizando as palavras-chave e seus respectivos sinônimos. A partir destas *strings* foram encontrados os estudos em cada base.

Etapa 2: Eliminação dos estudos duplicados: nesta etapa, foram necessários dois pesquisadores que realizaram uma pré-análise dos estudos, com o objetivo de eliminar os artigos duplicados.

Etapa 3: Triagem por Título e *Abstract*: nesta etapa os dois pesquisadores realizaram a leitura dos títulos e *abstracts* dos estudos, aplicando o critério de inclusão CI1 e os critérios de exclusão CE1 e CE2.

Etapa 4: Triagem por Inclusão e Exclusão: nesta etapa, um pesquisador realizou a leitura completa dos estudos selecionados na etapa anterior aplicando os critérios de inclusão e exclusão.

Etapa 5: Aplicação dos Critérios de Qualidade: nesta etapa, um pesquisador realizou a leitura completa dos estudos selecionados na etapa anterior aplicando os critérios de qualidade com a finalidade de *rankear* os estudos.

Etapa 6: Extração dos Dados: nesta etapa foi realizada a extração de dados baseada nas questões de pesquisa através de um formulário de extração, apresentado em 3.1.1.1.8.

Etapa 7: Análise dos Dados: nesta etapa foi realizada a análise dos dados coletados através da extração de dados.

3.1.1.1.8 Formulário de Extração de Dados

O formulário de extração de dados possui um papel essencial para extrair os dados relevantes dos estudos selecionados, possibilitando responder às questões de pesquisa. Baseado nisto, os respectivos dados foram extraídos de cada estudo:

- Título
- Autor(es)
- Ano da publicação
- O objetivo do estudo está claro? Sim/Não
- Existe alguma ameaça ao estudo? Se sim, quais ?
- Nome da DSL
- O estudo usa ou propõe uma DSL?
- Quais são as funcionalidades da DSL?
- Qual o domínio de aplicação foco da DSL?
- Quais tecnologias e/ou notações foram usadas para desenvolver a DSL?
- Quais são as técnicas/abordagens/métodos utilizados para garantir a cobertura do sistema através dos casos de teste?
- Como os dados relacionados ao SUT são representados nos modelos?
- Quais são os benefícios no uso da DSL?
- Quais são as limitações no uso da DSL?
- Quais são as tendências no uso da DSL?

3.1.1.1.9 Processo de Análise

Com o intuito de identificar as DSLs existentes, os estudos foram divididos entre os que utilizam e os que propõem as linguagens, onde cada linguagem foi tabelada em conjunto com suas funcionalidades, benefícios, desvantagens e tendências. Além disto, as tecnologias identificadas para o desenvolvimento de cada DSL também foram mapeadas, juntamente com as técnicas, abordagens e métodos para garantir a cobertura do sistema através dos caso de teste. Com isto, tornou-se possível a identificação das tendências e limitações das DSLs atuais.

3.1.2 Condução

A condução foi executada como descrito no processo de seleção presente em 3.1.1.1.7. Nesta seção é apresentado detalhadamente cada etapa realizada na condução deste SMS.

3.1.2.1 *Strings* de Busca Específicas

Como apresentado na seção 3.1.1.1.4, cada biblioteca digital necessita de uma *string* de busca específica para possibilitar a busca dos estudos. Assim, antes de realizar a pesquisa dos estudos foram elaboradas *strings* de busca específicas. A *string* de busca base apresentada na Figura 1 foi adaptada para cada biblioteca digital com o intuito de manter a similaridade entre as *strings*. O resultado relacionado à esta adaptação pode ser verificado através da Tabela 2.

Tabela 2 – Strings de Busca Utilizadas em Cada Biblioteca Digital

Biblioteca Digital	*String* de Busca
ACM DL	((dsl OR dsml OR "domain specific language"OR "domain-specific language"OR "domain-specific-language"OR "domain specific model* language"OR "domain-specific model* language"OR "domain specific model*-language"OR "domain-specific-model*-language") AND ("black box test*"OR "black-box test*"OR "functional test*"OR "regression test*"OR "acceptance test*"))
IEEE Xplore	((dsl OR dsml OR "domain specific language"OR "domain specific model* language") AND ("functional test*"OR "black box test*"OR "regression test*"OR "acceptance test*"))
Science Direct	(("domain specific language"OR "domain specific model language"OR "DSL "OR "DSML") AND ("black box test"OR "functional test"OR "regression test"OR "acceptance test"))

Tabela 2 continuação da página anterior

SCOPUS	TITLE-ABS-KEY((dsl OR dsml OR "domain specific language"OR "domain-specific language"OR "domain-specific-language"OR "domain specific model* language"OR "domain-specific model* language"OR "domain specific model*-language"OR "domain-specific-model*-language") AND ("black box test*"OR "black-box test*"OR "functional test*"OR "regression test*"OR "acceptance test*")) AND (LIMIT-TO (SUBJAREA, "COMP")) AND (LIMIT-TO(LANGUAGE, "English"))
Springer Link	((dsl OR dsml OR "domain specific language"OR "domain-specific language"OR "domain-specific-language"OR "domain specific model* language"OR "domain-specific model* language"OR "domain specific model*-language"OR "domain-specific-model*-language") AND ("black box test*"OR "black-box test*"OR "functional test*"OR "regression test*"OR "acceptance test*"))
Compendex	("domain-specific language"OR "domain specific language"OR "domain-specific-language"OR "domain specific modeling language"OR "domain-specific modeling language"OR "domain-specific-modeling language"OR "domain-specific-modeling-language"OR "DSL "OR "DSML") AND ("functional testing"OR "black box test"OR "black-box test"OR "black box testing"OR "black-box testing"OR "functional test"OR "regression test"OR "regression testing"OR "acceptance test"OR "acceptance testing")

3.1.2.2 Busca nas Bases de Dados

Após o desenvolvimento das *strings* de buscas específicas, elas foram utilizadas para realizar as buscas em suas respectivas bibliotecas digitais. Algumas bibliotecas, o resultado destas buscas passaram por filtragem, como na Compendex que foi filtrado e na SpringerLink. Na SpringerLink foram realizados filtros por idioma, categoria e foi removido os estudos que apresentavam apenas uma prévia. Além disto, na Science Direct foram possíveis apenas buscas utilizando o título, resumo e palavras-chave. O resultado final representando a quantidade de estudos encontrados após todas as buscas e filtros pode ser visualizado na Tabela 3.

3.1.2.3 Seleção dos Estudos

A seleção dos estudos seguiu as etapas definidas no processo de seleção apresentado na Seção 3.1.1.1.7. A primeira etapa foi apresentada na Seção 3.1.2.2, resultando em 1077 estudos encontrados. Partindo deste resultado foi realizada a remoção dos estudos

Tabela 3 – Quantidade de Estudos Encontrados

Biblioteca Digital	**Total**
ACM DL	16
IEEE Xplore	442
Science Direct	225
SCOPUS	33
Springer Link	335
Compendex	26
Total de Estudos	**1077**

duplicados, sendo esta a segunda etapa do processo de seleção, retirando assim todos os estudos repetidos, independente da biblioteca digital. Esta etapa resultou em 972 estudos para aplicar os Critério de Inclusão (CI) e Critério de Exclusão (CE).

A terceira etapa foi realizada por dois pesquisadores, realizando a leitura do título e *abstract* de cada um dos estudos aplicando os CI e CE. Ao final da aplicação dos critérios foram aceitos 46 estudos. Estes artigos aceitos passaram por uma quarta etapa onde foi realizada a leitura completa destes estudos, resultando em 31 artigos aceitos.

O resultado da etapa de seleção pode ser visualizado através da Figura 2. A quinta e a sexta etapa não resultam na remoção de estudos e são descritas nas subseções a seguir.

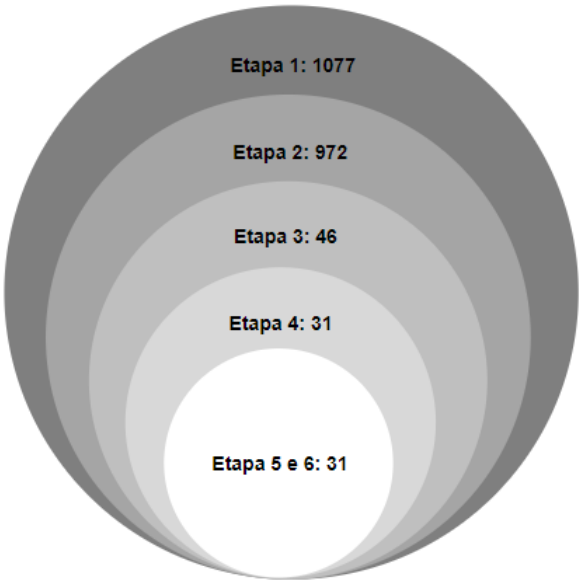


Figura 2 – Diagrama em Camadas dos Estudos Selecionados

3.1.2.4 Aplicação de Critérios de Qualidade

A aplicação dos Critérios de Qualidade (CQs) tem o intuito de verificar quais os estudos que mais contribuem para o propósito do Mapeamento Sistemático de Literatura (SMS). A Tabela 4 apresenta as informações relacionadas às pontuações de cada estudo. Cada estudo na tabela possui um Identificador (ID) juntamente com sua respectiva refe-

rência. As colunas CQ1, CQ2, CQ3 e CQ4 apresentam a pontuação do estudo baseada em cada um dos CQs. Na coluna "Score Geral" o estudo será classificado em Pouco Relevante, Relevante ou Muito Relevante para este trabalho. Enquanto a coluna "Score" mostra a pontuação total de cada estudo, sendo esta obtida a partir da soma da pontuação de cada critério. Os CQs foram utilizados como uma verificação da relevância desse estudo para atender ao objetivo do SMS, desta maneira, nenhum estudo foi removido nesta etapa.

Tabela 4 – Aplicação dos Critérios de Qualidade (SMS)

ID	Referência	CQ1	CQ2	CQ3	CQ4	Score Geral	Score
E1	(FELDERER; JESCHKO, 2018)	S	S	S	P	Muito Relevante	3.5
E2	(HARGASSNER et al., 2008)	S	S	S	P	Muito Relevante	3.5
E3	(TÖRSEL, 2013)	S	S	S	S	Muito Relevante	4.0
E4	(IBER et al., 2015b)	S	S	S	P	Muito Relevante	3.5
E5	(DWARAKANATH et al., 2017)	S	S	S	S	Muito Relevante	4.0
E6	(GAFUROV; HURUM; MARKMAN, 2018)	S	S	S	S	Muito Relevante	4.0
E7	(DUCLOS et al., 2013)	S	S	S	P	Muito Relevante	3.5
E8	(HÄSER; FELDERER; BREU, 2016a)	S	S	P	P	Relevante	3.0
E9	(SANZ et al., 2015)	S	S	S	P	Muito Relevante	3.5
E10	(CONTAN; MICLEA; DEHELEAN, 2017)	S	S	S	P	Muito Relevante	3.5
E11	(ZHOU; YIN, 2014)	S	S	S	S	Muito Relevante	4.0
E12	(MAKEDONSKI et al., 2016)	S	S	S	S	Muito Relevante	4.0
E13	(SIPPL et al., 2016)	P	S	P	S	Relevante	3.0
E14	(JORGE et al., 2018)	S	S	S	S	Muito Relevante	4.0
E15	(KING et al., 2014)	P	S	S	P	Relevante	3.0
E16	(CHATLEY; AYRES; WHITE, 2010)	S	S	S	S	Muito Relevante	4.0
E17	(ARTHO et al., 2013)	S	S	S	S	Muito Relevante	4.0
E18	(NABUCO; PAIVA, 2014)	S	S	S	P	Muito Relevante	3.5
E19	(ALEGROTH; BACHE; BACHE, 2015)	S	S	P	P	Relevante	3.0
E20	(MILLER; KUMAR; SINGHAL, 2009)	P	S	P	N	Pouco Relevante	2.0
E21	(KOS; MERNIK; KOSAR, 2016)	S	S	S	S	Muito Relevante	4.0
E22	(MAKEDONSKI et al., 2019)	S	S	S	S	Muito Relevante	4.0
E23	(HÄSER; FELDERER; BREU, 2014)	S	N	S	P	Relevante	2.5
E24	(HERBOLD et al., 2015)	S	S	S	P	Muito Relevante	3.5
E25	(LUNA; ROSSI; GARRIGÓS, 2011)	S	S	S	S	Muito Relevante	4.0
E26	(LUNA et al., 2010)	S	S	S	P	Muito Relevante	3.5
E27	(HOLMES et al., 2018)	S	S	S	P	Muito Relevante	3.5
E28	(IBER et al., 2015a)	S	S	S	S	Muito Relevante	4.0
E29	(PRASETYA et al., 2011)	S	S	S	P	Muito Relevante	3.5
E30	(ELODIE et al., 2018)	S	S	S	P	Muito Relevante	3.5
E31	(TALBY, 2009)	S	N	S	P	Relevante	2.5

3.1.3 Análise dos Estudos

Com a finalização da aplicação dos critérios de qualidade, os dados de cada estudo foram extraídos utilizando o formulário de extração exibidos na Seção 3.1.1.1.8. Assim, nesta subseção serão apresentados os resultados relacionados as respostas das questões de pesquisas apontadas na Seção 3.1.1.1.3. Com base na extração de dados dos estudos foi gerado um gráfico de estudo por ano, que pode ser visto através da Figura 3. Analisando esta figura é possível verificar que apesar de alguns altos e baixos, os estudos que apresentam o uso ou proposta de uma DSL para teste funcional podem ser considerados um tópico atual de pesquisa em crescente.

A seguir, são apresentadas as respostas de cada questão de pesquisa.

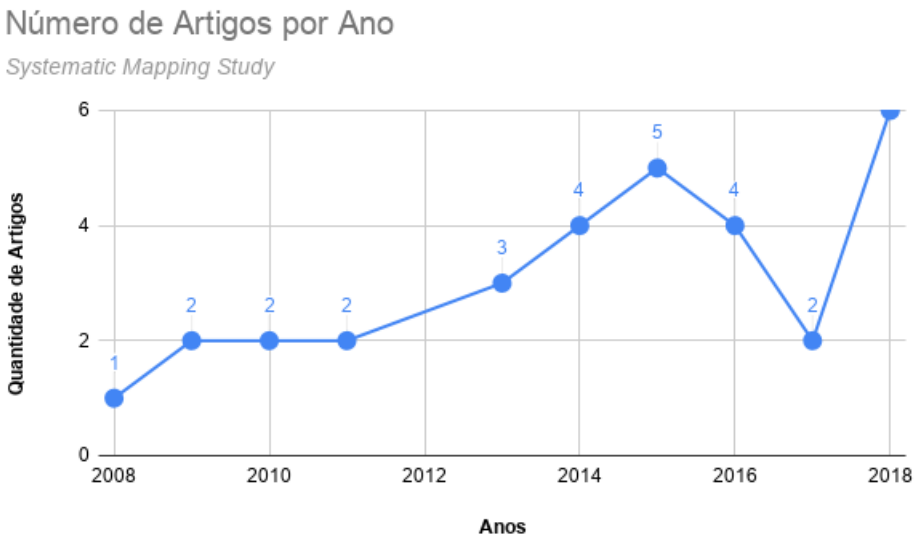


Figura 3 – Quantidade de Artigos por Ano (SMS)

QP1: *Os artefatos gerados a partir da DSL (por exemplo, scripts e modelos gerados) são aplicados a quais domínios de software?*

Após a análise realizada através da extração de dados foi possível observar que a maioria das DSLs existentes e a maior parte dos estudos apresentam como foco principal as aplicações *web*, onde foram encontrados treze estudos relacionados (E01, E10, E03, E05, E06, E08, E09, E11, E15, E16, E18, E25 e E26). Dentre as DSLs citadas para teste funcional de *software* de aplicações *web* é possível citar o Gherkin, DSL1, DSL2, Legend, ATA-DSL, PARADIGM e WebSpec, os estudos E03, E09, E11 e E16 não apresentaram um nome para a respectiva linguagem, ou seja, não de aplicam (N/A). Assim, o domínio de *web* é o que mais estão apresentando tentativas de aplicação e desenvolvimento de solução.

Seguindo a análise, em outros domínios foram encontrados dois estudos que apresentaram soluções para os sistemas críticos através da DSL Ubtl (E04 e E28) e Sistemas de Telecomunicações, Automotivos, Aeroespacial e Saúde por meio da linguagem TDL (E12, E22). Aplicações desenvolvidas em C++, ADAS, Sistemas Embarcados, Sistemas Direcionados a Eventos, Aplicações baseadas em Java, Aplicativos Convergentes, Java ou Delphi, Aplicações Orientadas a Serviço, Java ou Python, *Web Services*, todos estes domínios foram representados apenas por um estudo cada, sendo eles respectivamente E07, E13, E14, E17, E19, E20, E21, E24, E27, E29. Representando o interesse de diversas áreas pelo uso de DSLs para testes funcionais.

O estudo E2 apresenta uma DSL para testes *mobile*, entretanto é um estudo do ano de 2008. Decorrente a evolução rápida dos Sistemas Operacionais *mobile*, atualmente esta linguagem pode não oferecer suporte à maioria dos *smartphones* atuais. Além disto, os estudos E23 e E31 não apresentaram o nome e o domínio da linguagem, não possibilitando uma análise sobre estas DSLs. Todos os dados relacionados à esta questão de pesquisa podem ser visualizados através da Tabela 5.

Tabela 5 – Domínio foco de cada DSL (SMS)

IDs	Nome da DSL	Domínio
E01, E10	Gherkin	Aplicações *Web* e pode ser utilizado em outros Domínios.
E02	TBL (*Testbed Language*)	Aplicações *Mobile*.
E03	N/A	Aplicações *Web*.
E04, E28	Ubtl (*UTP-based Test Language*)	Sistemas Críticos.
E05	ATA DSL	Aplicações *Web*.
E06	N/A	Aplicações *Web*.
E07	N/A	Aplicações C++.
E08	DSL1 e DSL2	Aplicações *Web*.
E09	N/A	Aplicações *Web*.
E11	N/A	Aplicações *Web*.
E12, E22	*Test Description Language* (TDL)	Sistemas de Telecomunicações, Automotiva, Aeroespacial e Saúde.
E13	N/A	ADAS (*Advanced Driver Assistance Systems*).
E14	Claret	Sistemas Embarcados.
E15	Legend	Aplicações *Web*.
E16	N/A	Aplicações *Web*.
E17	N/A	Sistemas direcionado a Eventos.
E18	PARADIGM	Aplicações *Web*.
E19	N/A	Aplicações baseadas em Java.
E20	Photonese	Aplicativos Convergentes.
E21	Sequencer	Java e Delphi.
E23	N/A	N/A.
E24	MIDAS	Aplicações Orientadas a Serviço.
E25, E26	WebSpec	Aplicações *Web*.
E27	TSTL (*Template Scripting Testing Language*)	Aplicações Python e Java.
E29	CTy	*Web Services*.
E30	Yest	ERPs e Aplicativos de Negócio.
E31	N/A	N/A.

QP2: *Quais são as funcionalidades destas DSLs ?*

A partir da análise de cada uma destas linguagens foi possível detectar uma série de funcionalidades. A DSL Gherkin (E01, E10) tem como principal funcionalidade a conversão de texto criado em linguagem natural em testes executáveis, além disto possibilita a parametrização dos dados de teste. A linguagem TBL (E02) fornece a possibilidade de criação de casos de teste partindo do uso de instruções pré-definidas. O estudo E03 não apresenta o nome da sua respectiva DSL, entretanto a linguagem proposta possibilita a parametrização de dados através de XML e conversão do modelo do sistema em um formato no qual a ferramenta NuSMV aceita, possibilitando assim uma verificação

do modelo. A linguagem Ubtl (E04, E28), apresenta a possibilidade de transformar sua especificação em modelos UML. A ATA DSL (E05) possibilita a geração de *scripts* de teste em Java utilizando o *Selenium WebDriver* para a comunicação com o navegador e ainda possibilita a edição do código de teste em tempo de execução.

O estudo E06 apresenta uma linguagem que possibilita a criação de testes automatizados com ou sem parametrização. Enquanto o artigo E08 apresenta uma solução para detectar vazamentos de memória dentro de uma aplicação em C++, possibilitando sua especificação direto no código-fonte da aplicação sem afetar o seu funcionamento. As linguagens denominadas DSL1 e DSL2, foram implementadas como exemplo de um processo de desenvolvimento de DSLs, dessa forma estas linguagens são idênticas onde possuem como diferença o uso de *plugins* para aumentar a produtividade durante a especificação de casos de teste em linguagem natural. No artigo E09, a linguagem apresentada possibilita a geração de *scripts* de teste, parametrização de dados e realizar a criação de cenários de teste.

Outra linguagem é apresentada no estudo E11, onde a DSL tem como principal funcionalidade escrever *scripts* de teste de forma declarativa e gerar posteriormente versões executáveis. O estudo E12 e E22 apresentam o TDL uma linguagem na qual o comportamento do SUT é descrito e a partir dele é possível realizar a geração e automatização dos casos de teste. O E13 apresenta uma linguagem de criação de casos de teste com base em um catálogo de situações. Enquanto isto, a linguagem Claret (E14) possibilita a descrição dos casos de teste, juntamente com suas pré e pós condições, além de gerar uma documentação relacionada. Os casos de teste criados no Claret também podem ser exportados através de um formato XML aceito pela ferramenta TestLink (BITTENCOURT, 2013).

Como outras DSLs já citadas a Legend (E15) executa e depura testes automatizados, além de possibilitar a parametrização dos dados de teste. A DSL apresentada no artigo E16 possui como principal funcionalidade a criação de *scripts* de teste. Enquanto a do E17 é única linguagem que da suporte a especificação não determinística, tanto de comportamento quanto especificação do SUT. A DSL PARADIGM (E18) se mostrou como uma das mais completas possibilitando a execução dos casos de teste, geração de relatórios, modelagem das interações do usuário com o sistema através de padrões de interação e integração com o Sikuli (YEH; CHANG; MILLER, 2009) para detecção de elementos na página.

Com a finalidade de testar aplicações Java a linguagem apresentada no estudo E19 viabiliza o agrupamento de ações com o objetivo de facilitar o entendimento dos códigos de teste. Por outro lado a Photonese (E20) possibilita somente a criação de *scripts* de teste. Com as funcionalidades de realizar asserções, definição de valores e gerar relatórios de teste o Sequencer (E21) é apresentado como uma solução para aplicações em Delphi ou Java. O estudo E23 também não apresenta o nome da sua respectiva linguagem entretanto,

são informadas as seguintes funcionalidades: Descrição de fluxos de teste juntamente com as suas respectivas pré e pós condições e exportação de relatórios de atividade. Como a única alternativa para testes funcionais em aplicações orientadas a serviço o MIDAS (E24) possui a geração de *scripts*, definição de um planejamento, *design* e dados de teste como principais funcionalidades.

O WebSpec (E25, E26) é uma linguagem gráfica que permite a criação de diagramas, simulações utilizando o *Selenium WebDriver* e JUnit e realizar associações entre os elementos do diagrama com textos em HyperText Markup Language (HTML) representando o sistema *web*. O TSTL (E27) apresenta a possibilidade de se realizar construção de SQLs e geração de testes aleatórios. A DSL CTy (E29) possibilita ao testador criar oráculos de teste, definir partições de equivalência, realizar testes combinatórios e utilizar métodos nativos do Haskell (THOMPSON, 2011). A linguagem Yest (E30) viabiliza a geração de testes e definição de uma tabela de decisão. Por fim, a DSL apresentada no estudo E31 possibilita realizar a validação de casos de teste.

QP3: *Quais tecnologias e notações foram usadas para o desenvolvimento das DSLs para testes funcionais?*

Alguns estudos não apresentam as tecnologias e notações utilizadas para o desenvolvimento das DSLs. Porém, das tecnologia apresentadas a mais usada para o desenvolvimento das linguagens foi o Xtext (BETTINI, 2016) citada em sete trabalhos (E01, E03, E04, E05, E12, E22 e E28). Devido a Xtext ser uma *Language Workbench* (LW) para a definição de linguagens textuais, é possível salientar uma maior quantidade de DSLs deste tipo. Um ponto importante é que como o Xtext apenas define a linguagem, outros estudos fizeram o uso do Xtend (BETTINI, 2016) para realizar a geração de texto (E03, E04, E05 e E28). O Eclipse Modeling Framework (EMF) e a LW Graphical Modeling Framework (GMF) também foram utilizadas dentre as linguagens gráficas (A22, A25 e A26). Além destas tecnologias, linguagens baseadas em Java também foram encontradas (A16 e A04). C# (E06), AspectC++ (E07), *Object Constraint Language* (OCL) (E12), EBNF (E14), Scala (E17), Ruby (E20), Haskell (E29) e Metamodelos (E21) foram alternativas de tecnologias utilizadas para o desenvolvimento de *Domain-Specific Languages* (DSLs) para testes funcionais de *software*.

QP4: *Quais técnicas/abordagens/métodos a DSL usa para garantir a cobertura dos testes, no caso de possuir a geração do script?*

As linguagens encontradas na sua grande maioria não possui o foco na cobertura do sistema através da geração de testes com base em métodos específicos para esta finalidade. O principal objetivo das DSLs eram a redução da complexidade dos testes para viabilizar o entendimento dos usuários do domínio. Algumas linguagens como a apresentada em E03 apresentam soluções para garantir a cobertura, esta DSL em específico faz o uso de uma ferramenta externa denominada NuSMV (CIMATTI et al., 1999) que realiza a verificação de modelos. Outras como a Ubtl (E04) apresentam uma solução que depende

de um especialista do domínio para descrever todos os possíveis cenários. O E13 consiste em dados de simulação e com base nos dados fornecidos é garantida a cobertura. A DSL Claret (E14) utiliza a ferramenta LTS-BT (CARTAXO et al., 2008) para realizar a geração e garantir a cobertura do sistema. A Legend (E16) apresenta uma solução mais simples que é a possibilidade de realizar uma rastreabilidade entre os *scripts* e os requisitos. Enquanto a PARADIGM (E18) faz o uso de algoritmos de busca em profundidade entre os caminhos especificados durante a modelagem do comportamento do sistema. O Sequencer (E21) apresenta, como alternativa para a garantia da cobertura, a transformação dos casos de teste estipulados em modelos de execução do sistema. A DSL MIDAS utiliza de Redes de Petri (LEE-KWANG; FAVREL; BAPTISTE, 1987) para garantia da cobertura da aplicação. A linguagem TSTL (E27) faz o uso do SPIN (HOLZMANN, 2004) que também é uma ferramenta de verificação de modelos para garantir a cobertura. Por fim, a linguagem CTy (E29) apresenta como critério de cobertura a execução de cada classe especificada, no mínimo uma única vez, enquanto a DSL Yest (E30) faz o uso de tabelas de decisão para realizar esta verificação.

QP5: *Como os dados de entrada do SUT são representados nos modelos?*

Os dados de entrada relacionados ao Sistema Sob Teste (SUT) dentre as DSLs encontradas geralmente são dados parametrizados que complementam os cenários ou fluxos da aplicação, expressos pela linguagem. Mesmo nas DSLs onde não são expressados os fluxos ou cenários, como é o caso da linguagem apresentada no estudo E03 os dados do SUT são informados através de parametrização. Caso a DSL não possua a funcionalidade relacionada à parametrização os dados são informados diretamente no *script* ou na especificação (E13).

QP6: *Quais são os benefícios, desvantagens e tendências relacionadas ao uso do DSL para testes funcionais?*

De acordo com a análise dos estudos, é possível salientar alguns benefícios apresentados pela adoção das DSLs para testes funcionais de *software*, como: 1) Aumentar a legibilidade dos *scripts* de teste. 2) Incluir especialistas do domínio durante a etapa de testes. 3) Diminuir a complexidade dos códigos de teste e reduzir o tempo necessário na etapa de testes. 4) Evitar erros relacionados à ortografia ou mau entendimento dos requisitos. 5) Desenvolvimento direcionado à testes. 6) Desenvolvimento dos testes em paralelo ao da aplicação. 7) Facilitar a manutenibilidade dos *scripts* de teste. 8) Melhorar a cobertura dos testes da aplicação. 9) Baixa curva de aprendizado. 10) Possibilidade de descrição dos testes em linguagem natural.

Apesar de todos os benefícios apresentados, ainda existem problemas relacionados às DSLs para testes funcionais de *software*, como: 1) Algumas DSLs não possuem regras de validação. 2) O tamanho dos modelos podem ser limitados, tendo em vista que a verificação de algumas linguagens são feitas por ferramentas externas. 3) Dificuldade de verificar o *script* de teste gerado, isto é importante para analisar se o erro é relacionado ao

script automatizado mal implementado ou o sistema que possui defeitos. 4) As linguagens que garantem a cobertura não possibilitam a definição de caminhos específicos dentro da DSL. 5) Algumas DSLs não apresentam uma boa reusabilidade de seus componentes, aumentando assim o esforço necessário para a modelagem do sistema. 6) Algumas linguagens necessitam que o usuário possua conhecimentos em métodos formais como Máquina de Estados Finitos (FSM).

Com base nos problemas citados no cenário de DSLs para testes funcionais, os estudos mapeados apresentaram algumas tendências: 1) Realização de avaliações empíricas nas linguagens, pois muitas vezes as DSLs não são avaliadas, não possibilitando o seu uso no cenário real de desenvolvimento. 2) Utilização de outra linguagem de *script* de apoio para suprir os problemas das DSLs. 3) Inclusão de caminhos específicos nas linguagens que tem como finalidade garantir a cobertura do sistema. 4) Desenvolvimento todo guiado por modelos. 5) Geração de casos e dados de teste. 6) Suporte às mudanças relacionadas ao SUT, permitindo uma atualização simplificada da especificação. 7) Atender a diversos domínios. 8) Elaborar *guidelines* para o uso destas linguagens, a fim de facilitar o uso por terceiros.

3.1.4 Ameaças à Validade

Um ponto importante a ser destacado, é que podem existir outras *Domain-Specific Languages* (DSLs) não mapeadas, tendo em vista que nem todos os pesquisadores conhecem o respectivo conceito. Dessa maneira, é possível que existam outras soluções que se encaixem como DSLs. Outro ponto importante é a possibilidade das análises e categorização dos estudos serem impactadas de acordo com nossos interesses, a fim de mitigar isto foram utilizados os critérios de inclusão e exclusão durante a etapa de seleção, o que resultou em estudos conflitantes entre os envolvidos, que foram resolvidos através de um terceiro pesquisador. Outra ameaça é que os termos definidos foram baseados em nossas experiências e investigações dentro do cenário de testes funcionais de *software*, dessa forma não podemos afirmar que todos os sinônimos possíveis foram utilizados. Um fator que também pode impactar no resultado são as *strings* de busca, decorrente a algumas bases apresentarem limitações ou falta de mecanismos para otimizar a busca.

Uma ameaça à validade interna seria o viés dos pesquisadores referente às DSLs funcionais de *software*. Um fator a ser salientado seria a limitação a somente testes funcionais, eliminando estudos que utilizam outros tipos de teste que podem auxiliar nos testes funcionais, com isto pode existir alguma solução não investigada. Entretanto, estes estudos deveriam ser adaptados para o contexto de testes funcionais, todos os estudos que apresentavam o uso ou proposta de uma DSL para testes funcionais foram aceitos.

Como ameaça a validade de construção é possível salientar a exclusão de algum estudo que apresentasse tanto o uso quanto uma proposta de DSL funcional para testes de *software*, para isto foram usadas regras de busca e fases bem definidas de exclusão

e inclusão com diferentes filtros. Além disto, a fim de mitigar esta problemática outros integrantes foram envolvidos no processo. Como ameaça externa a validade é possível apresentar que não foram levados em consideração outros possíveis mapeamentos na área de *Domain-Specific Languages* (DSLs), pois possuem objetivos diferentes. Por fim, existe a ameaça de existirem outras DSLs para testes funcionais de *software* que não estão presentes nas bibliotecas digitais envolvidas na busca.

3.2 *Snowballing*

Tendo em vista a possibilidade de encontrar novas propostas ou usos de DSLs que não estivessem nas bibliotecas digitais pesquisadas, assim viu-se a necessidade de execução de um *Snowballing* para encontrar estes estudos. Esta seção apresenta mais detalhes relacionados ao protocolo utilizado para a realização do *Snowballing*.

3.2.1 Escopo e Objetivo

O escopo e objetivo do *Snoballing* (WOHLIN, 2014) foi baseado no apresentado na Seção 3.1.1.1.1. Sendo o objetivo detectar novas DSLs para testes funcionais não mapeadas através do SMS. Possibilitando assim verificar maiores déficits e benefícios do cenário atual, para a proposta de uma nova linguagem específica de domínio para testes funcionais de *software*, a fim de atenuar as limitações encontradas e unir a maioria dos benefícios e tendências.

3.2.2 Estudos Iniciais

A fim de executar o *Snowballing* é necessário um conjunto de estudos iniciais. Estes estudos iniciais por recomendação devem ser representativos para a área a ser pesquisada. Assim, o melhor conjunto que tínhamos conhecimento eram os 31 estudos resultantes da etapa de condução do SMS.

3.2.3 Estratégia de Busca

Tendo em vista que o *Snowballing* não leva em consideração as bibliotecas digitais e sim os estudos iniciais, não é necessário ter *strings* de busca. Assim, a busca por novos estudos foi dividida em duas etapas o *Backwards* e o *Forwards*. Na etapa de *Backwards* foram verificadas todas as citações utilizadas por cada estudo, caso um destes estudos presentes nas citações fosse incluído suas citações também eram verificadas e assim sucessivamente. Após a finalização desta etapa, foi realizado o *Forwards* destes estudos, verificando os estudos que citaram estes artigos, caso um destes estudos fosse incluído suas citações e quem citou eles também eram verificados. Estas etapas se repetiam até que não fossem encontrados novos estudos.

3.2.4 Questões de Pesquisa, Critérios e Formulário de extração

As questões de pesquisa, critérios de inclusão e exclusão e o formulário de extração de dados usados no *Snowballing* foram os mesmos do SMS. Desse modo:

- As questões de pesquisa utilizadas foram apresentadas na Seção 3.1.1.1.3.
- Os critérios de inclusão e exclusão utilizados foram apresentados na Seção 3.1.1.1.5.
- Os critérios de qualidade utilizados também foram apresentados na Seção 3.1.1.1.6.
- O formulário de extração de dados utilizado foi apresentado na Seção3.1.1.1.8.

3.2.5 Processo de Seleção

A fim de selecionar os estudos relevantes para responder as questões de pesquisa, o processo de seleção foi dividido em cinco etapas. As etapas são descritas a seguir:

Etapa 1: *Backwards*: esta etapa teve como finalidade encontrar estudos que foram citados pelos artigos iniciais. Assim, foi realizada uma triagem por Título, *Abstract* e Conclusão aplicando os critérios de inclusão e exclusão em todos os estudos que foram citados uma ou mais pelos estudos iniciais. Caso algum trabalho fosse incluído, neste estudo também era realizada sua leitura completa aplicando novamente os critérios de inclusão e exclusão. Se ao final destas atividades o estudo fosse incluído e atendesse aos objetivos do *Snowballing*, o mesmo era adicionado aos estudos iniciais. Desta forma em todos os estudos novos se realizava a etapa de *Backwards* e *Forwards*. Esta etapa foi finalizada quando não se encontraram mais estudos relacionados.

Etapa 2: *Forwards*: esta etapa teve como finalidade encontrar estudos que citaram os estudos iniciais. Assim, foi realizada uma triagem por Título, *Abstract* e Conclusão aplicando os critérios de inclusão e exclusão de todos os estudos nos quais os trabalhos iniciais foram citados. Caso algum trabalho fosse incluído, neste estudo também era realizada sua leitura completa aplicando novamente os critérios de inclusão e exclusão. Se ao final destas atividades o estudo fosse incluído e atendesse aos objetivos do *Snowballing*, o mesmo era adicionado aos estudos iniciais. Desta forma em todos os estudos novos se realizava a etapa de *Backwards* e *Forwards*. Esta etapa também foi finalizada quando não se encontraram mais estudos relacionados.

Etapa 3: Aplicação dos Critérios de Qualidade (CQs): nesta etapa foi realizada a leitura completa dos estudos selecionados nas etapas anteriores aplicando os critérios de qualidade com a finalidade de classificar os estudos.

Etapa 4: Extração de Dados: nesta etapa foi realizada a extração de dados baseada nas questões de pesquisa através de um formulário de extração. Este formulário foi o mesmo utilizado no Mapeamento Sistemático de Literatura (SMS), e foi apresentado na Seção 3.1.1.1.8.

Etapa 5: Análise dos Dados: nesta etapa foi realizada a análise dos dados coletados através da extração de dados.

3.2.6 Condução do *Snowballing*

A condução foi executada como descrito no processo de seleção presente em 3.2.5. Nesta seção é apresentado detalhadamente cada etapa realizada na condução deste *Snowballing*.

3.2.6.1 Seleção dos Estudos

A seleção dos estudos seguiu as etapas definidas no processo de seleção apresentado na Seção 3.2.5. Esta seção representa a primeira e a segunda etapa. Na etapa de *Backwards* foram encontrados quatorze novos estudos. Enquanto na etapa de *Forwards* foram detectados trinta e dois novos trabalhos, resultando assim em um total de 46 estudos para aplicar os Critérios de Qualidade (CQs) e realizar a extração de dados. Os resultados desta etapa de seleção podem ser visualizados através da Tabela 6.

Tabela 6 – Quantidade de Estudos Incluídos no *Snowballing* por Etapa

Etapa	**Total**
Backwards	14
Forwards	32
Total de Estudos	**46**

3.2.6.2 Aplicação dos Critérios de Qualidade

A aplicação dos Critérios de Qualidade (CQs) como no Mapeamento Sistemático de Literatura (SMS) tem o intuito de verificar quais os estudos que mais contribuem para o propósito do *Snowballing*. A Tabela 7 apresenta as informações relacionadas às pontuações de cada estudo. Cada estudo na tabela possui um Identificador (ID) juntamente com sua respectiva referência. As colunas CQ1, CQ2, CQ3 e CQ4 apresentam a pontuação do estudo baseada em cada um dos CQs. Na coluna "Score Geral" o estudo era classificado em Pouco Relevante, Relevante ou Muito Relevante, de acordo com o objetivo deste trabalho . Enquanto a coluna "Score" mostra a pontuação total de cada estudo, sendo esta obtida a partir da soma da pontuação de cada critério. Os CQs foram utilizados como uma verificação da relevância desse estudo para atender ao objetivo do *Snowballing*, desta maneira, nenhum estudo foi removido nesta etapa.

Tabela 7 – Aplicação dos Critérios de Qualidade (*Snowballing*)

ID	Referência	CQ1	CQ2	CQ3	CQ4	Score Geral	Score
E32	(HÄSER; FELDERER; BREU, 2016b)	S	P	S	P	Relevante	3.0
E33	(TORSEL, 2011)	S	S	S	S	Muito Relevante	4.0
E34	(MICALLEF; COLOMBO, 2015)	S	S	S	S	Muito Relevante	4.0
E35	(COSTA; PAIVA; NABUCO, 2014)	S	S	S	S	Muito Relevante	4.0
E36	(MOREIRA; PAIVA; MEMON, 2013)	S	S	S	S	Muito Relevante	4.0
E37	(VILELA; PAIVA, 2014)	S	P	S	N	Relevante	2.5
E38	(MONTEIRO; PAIVA, 2013)	S	P	S	P	Relevante	3.0
E39	(MOREIRA; PAIVA, 2014a)	S	P	S	P	Relevante	3.0
E40	(HOIS; SOBERNIG; STREMBECK, 2014)	S	S	S	S	Muito Relevante	4.0
E41	(SANTIAGO et al., 2013)	S	S	S	P	Muito Relevante	3.5
E42	(KANSTRÉN; PUOLITAIVAL, 2012)	S	S	S	S	Muito Relevante	4.0
E43	(KOS et al., 2011)	S	S	P	P	Relevante	3.0
E44	(KOS et al., 2011)	P	S	P	P	Relevante	2.5
E45	(ULRICH et al., 2014)	S	P	S	P	Relevante	3.0
E46	(ARTHO et al., 2017)	S	S	N	P	Relevante	2.5
E47	(ARTHO et al., 2015b)	S	S	S	S	Muito Relevante	4.0
E48	(ARTHO et al., 2015a)	S	N	S	N	Pouco Relevante	2.0
E49	(CUNHA; SONG, 2014)	S	S	S	S	Muito Relevante	4.0
E50	(LUNA; GARRIGOS; ROSSI, 2010)	S	S	S	P	Muito Relevante	3.5
E51	(OTADUY; DÍAZ, 2017)	S	S	S	P	Muito Relevante	3.5
E52	(HÄSER; FELDERER; BREU, 2018)	P	S	P	N	Pouco Relevante	2.0
E53	(WINKLER; MEIXNER; BIFFL, 2018)	S	N	S	P	Relevante	2.5
E54	(MOREIRA et al., 2017)	S	S	S	P	Muito Relevante	3.5
E55	(MORGADO; PAIVA, 2015)	S	S	S	S	Muito Relevante	4.0
E56	(HU; ZHU; YANG, 2018)	S	S	S	S	Muito Relevante	4.0
E57	(KOS et al., 2012)	S	S	S	N	Relevante	3.0
E58	(MOREIRA; PAIVA, 2014b)	S	S	S	S	Muito Relevante	4.0
E59	(PAIVA; VILELA, 2017)	S	S	S	S	Muito Relevante	4.0
E60	(Dias; Paiva, 2017)	S	S	S	S	Muito Relevante	4.0
E61	(HAMMOUD; ZARAKET; MASRI, 2017)	S	N	S	P	Relevante	2.5
E62	(SILVA; PAIVA; SILVA, 2018b)	S	S	S	P	Muito Relevante	3.5
E63	(SILVA; PAIVA; SILVA, 2018a)	S	S	S	P	Muito Relevante	3.5
E64	(MOREIRA; PAIVA, 2014c)	S	S	S	P	Muito Relevante	3.5
E65	(OLAJUBU et al., 2017)	S	S	S	P	Muito Relevante	3.5

Tabela 7 – Aplicação dos Critérios de Qualidade (*Snowballing*)

ID	Referência	CQ1	CQ2	CQ3	CQ4	Score Geral	Score
E66	(BUSSENOT; LEBLANC; PERCEBOIS, 2016)	S	S	S	S	Muito Relevante	4.0
E67	(KESSERWAN et al., 2019)	S	S	S	P	Muito Relevante	3.5
E68	(HÄRLIN, 2016)	S	S	S	P	Muito Relevante	3.5
E69	(SILVA, 2018)	S	S	S	S	Muito Relevante	4.0
E70	(CARVALHO, 2016)	S	N	S	P	Relevante	2.5
E71	(SACRAMENTO, 2014)	S	S	N	N	Pouco Relevante	2.0
E72	(RODRIGUES, 2018)	S	S	N	P	Relevante	2.5
E73	(VILELA, 2013)	S	S	S	P	Muito Relevante	3.5
E74	(SILVA, 2017)	S	S	S	S	Muito Relevante	4.0
E75	(MACIEL, 2019)	S	S	S	S	Muito Relevante	4.0
E76	(SACRAMENTO; PAIVA, 2014)	S	S	S	S	Muito Relevante	4.0
E77	(BUSSENOT; LEBLANC; PERCEBOIS, 2018)	S	S	S	S	Muito Relevante	4.0

3.2.7 Análise dos Estudos

Com a finalização da aplicação dos critérios de qualidade nos estudos encontrados através do *Snowballing*, cada artigo teve por sua vez seus dados extraídos utilizando o formulário de extração apresentado na Seção 3.2.4. Assim, nesta seção serão apresentados os resultados relacionados as respostas das questões de pesquisas apresentadas na Seção 3.2.4. Com base na extração de dados dos estudos foi criado um gráfico de estudo por ano, que pode ser visualizado através da Figura 4. A figura representa uma oscilação na quantidade de artigos no decorrer dos anos, mas é possível analisar que o uso ou proposta de uma DSL para teste funcional são tópicos atuais de pesquisa. Isto acontece devido ao *Snowballing* ter sido executado no início de 2019, e sendo assim, é possível detectar estudos atuais relacionados.

A seguir, são apresentadas as respostas de cada questão de pesquisa.

Figura 4 – Quantidade de Artigos por Ano (*Snowballing*)

QP1: *Os artefatos gerados a partir da DSL (por exemplo, scripts e modelos gerados) são aplicados a quais domínios de software?*

Após a extração de dados e análise dos trabalhos, a maioria dos estudos ainda são para o domínio de Aplicações *Web* com cerca de 29 artigos relacionados. Mantendo assim o panorama de que é o domínio no qual as *Domain-Specific Languages* (DSLs) para testes funcionais possuem um maior foco. Apesar da detecção de novas linguagens a maioria dos estudos encontrados são relacionados ao uso das DSLs descobertas através do Mapeamentos Sistemáticos de Literatura (SMSs). Foi detectado também crescimento na quantidade de estudos relacionados ao uso das linguagens para criação de testes em aplicações *mobile*, onde três artigos apresentam este domínio, sendo um deles voltado para jogos. O que mostra o aumento na preocupação da qualidade dos aplicativos *mobile* e tentativa de propor novas soluções a esta problemática. Soluções relacionadas a sistemas de aviação também foram mais propostas e utilizadas o que mostra a possibilidade do uso de DSLs para aplicações críticas. Com o crescimento das linguagens, uma das possíveis preocupações, como também detectado no SMS, é a qualidade destas DSLs, assim foi encontrado através do *Snowballing* um estudo voltado para testes em outras DSLs. Os demais domínios encontrados são similares aos identificados através do SMS, a Tabela 8 apresenta um resumo destas informações.

Tabela 8 – Domínio foco de cada DSL (*Snowballing*)

IDs	Nome da DSL	Domínio
E32	N/A	Aplicações *Web*.
E33	N/A	Aplicações *Web*.

Tabela 8 – Domínio foco de cada DSL (*Snowballing*)

IDs	Nome da DSL	Domínio
E34, E53, E56, E68, E69	Gherkin	Aplicações *Web* e *Mobile*.
E34	Outras DSLs (Sem nome)	*E-Commerce* e Jogos Android (Demais DSLs).
E40	Natural-language Requirements - DSL	DSLs.
E41	Legend	Aplicações *Web* e *Softwares* como Serviço.
E42	OSMOTester MBT	Da suporte a abordagens MBT *online* e *offline*.
E43, E44, E57	Sequencer	Procedimentos de Medição.
E45, E67	ETSI-TDL	Sistemas Embarcados.
E46, E47, E48	ModBat	Sistemas em Rede e Aplicações Java, Scala e C.
E49	N/A	Aplicações *Web*.
E50	WebSpec	Aplicações *Web*.
E51	TestMind	Aplicações *Web*.
E52	N/A	Aplicações *Web*.
E61	GUI specification language	Plataformas Cruzadas em Java.
E62, E63, E74	TSL (Test Specification Language)	Sistemas de Informação.
E65	N/A	Sistemas de Aviação.
E66	N/A	Sistemas de Aviação e Integração de Sistemas em Rede.
E75	RSL	N/A.
E77	ATA21 e ATA42	Ambas são para Sistemas de Aviação.
E35, E36, E37, E38, E39, E54, E55, E58, E59, E60, E64, E70, E71, E72, E73, E76	PARADIGM	Aplicações *Web*, *Mobile* e *Desktop*.

QP2: *Quais são as funcionalidades destas DSLs ?*

Após a análise de cada linguagem foram verificadas novas funcionalidades. A DSL presente no estudo E32 é composta por um conjunto de conceito do domínio *Web* como "abrir", "fechar"dentre outros, que foram adicionados à uma linguagem natural para a es-

pecificação de cenários de teste. Enquanto a linguagem apresentada no estudo E33 possui a geração de oráculos de teste e a transformação de casos de teste abstratos em *scripts* executáveis. O estudo E34 apresenta o Gherkin e outras DSLs, entretanto todas possuem como funcionalidades a possibilidade de realizar a descrição dos dos cenários de teste. A linguagem mais citada é a PARADIGM com dezessete estudos, e apresenta uma série de funcionalidades como geração e execução de casos de teste, fornecimento de informações relacionadas à cobertura do sistema e possibilidade de modelagem de uma árvore de caminhos dentro da aplicação. A DSL *Natural-Language Requeriments* apresentada pelo trabalho E40 possibilita a especificação de requisitos em linguagem natural e a geração de casos de teste executáveis. Através dos estudos apresentados no *Snowballing* a linguagem Legend (E41) apresenta como principal funcionalidade a geração de *scripts* e o uso de ferramentas para edição, execução e depuração de casos de teste. A OSMOTester MBT (E42) foi detectada através do *Snowballing* e possibilita a geração de testes através de um modelo genérico, além de prover também os dados de teste.

Seguindo a análise, o Sequencer (E43, E44, E57) permite ao especialista de domínio criar seus próprios procedimentos de medição. A linguagem TDL ou ETSI-TDL tem como objetivo prover a descrição de cenários em nível de abstração mais alto, podendo ser considerado até um intermédio entre o código em *script* e a documentação. A DSL ModBat apresentada pelos estudos E46, E47 e E48 e permite a descrição do comportamento do sistema, adicionando transições, pré e pós condições e utiliza máquinas de estado finito estendidas para a geração de casos de teste. O estudo E49 apresenta uma DSL que permite realizar a especificação e geração dos testes. Enquanto o WebSpec (E50) apresentou como principal funcionalidade apenas a especificação dos requisitos. Outra linguagem detectada foi o TestMind (E51) que permite o teste colaborativo, geração de casos de teste, realização de capturas de tela e especificação dos requisitos. Por outro lado o estudo E52 apresentou uma linguagem na qual é possível realizar a especificação de cenários de teste.

Dentre as novas DSLs encontradas através do *Snowballing*, podemos citar a *GUI specification language* (E61) que apresenta uma solução para realizar captura de comportamentos posicionais e lógicos relacionados à interface da aplicação. A linguagem TSL (E62, E63, E74) apresenta diversas funcionalidades como conversão de especificação RSL para TSL, conversão de TSL para Gherkin, conversão de TSL para casos de teste refinados e geração de casos de teste. Os estudos E65 e E66 não apresentam nomes para suas respectivas DSLs. Enquanto a linguagem do estudo E65 viabiliza a especificação de requisitos em alto nível a do artigo E66 permite a geração da estrutura dos testes, uso de parte da automatização dos procedimentos e possui integração com as restrições OCL. A DSL RSL possibilita realizar a geração de testes funcionais e utiliza a sintaxe RSL (responsável por especificar requisitos de sistemas). As linguagens ATA21 e ATA42 permitem especificar comportamentos específicos dos aviões e possibilita gerar código em Python.

QP3: *Quais tecnologias e notações foram usadas para o desenvolvimento das DSLs para testes funcionais?*

Como apresentado através da análise do SMS, entre as tecnologias mais utilizadas o Xtext ainda foi o mais mencionado (E33, E34, E40, E45, E62, E65, E67, E74, E75). Uma possibilidade é a confirmação da afirmação realizada no estudo (TÖRSEL, 2013), relacionada às DSLs textuais serem mais produtivas no cenário de testes. Outras tecnologias bastante citadas foi o EMF (E50, E54, E55, E58) para definição de metamodelos e o GMF (E38, E39, E50) para a implementação de linguagens gráficas. Além do Xtend que também foi mencionado (A68), outras tecnologias como o MPS (E52, E66), ANTLR (E61), DLex (E44). Linguagens internas também foram apresentadas e desenvolvidas a partir das seguintes GPLs: C#(E41), Java(E42), Delphi(E43) e Scala(E46, E47, E48).

QP4: *Quais técnicas/abordagens/métodos a DSL usa para garantir a cobertura dos testes, no caso de possuir a geração do script?*

Diferentemente do SMS, no *Snowballing* foram detectados mais técnicas,abordagens e métodos para garantir a cobertura do sistema. O estudo E33 utiliza busca em largura, enquanto a DSL PARADIGM (E35, E36, E37, E38, E39, E54, E55, E58, E59, E60, E64, E70, E71, E72, E73, E76) utiliza de buscas em profundidade através dos diagramas criados na linguagem, onde cada vértice é considerado uma URL e cada transição um *link* entre as URLs. A linguagem Legend (E41) utiliza um *framework* externo denominado *Echo* para garantir a cobertura da aplicação. O OSMOTester MBT (E42) utiliza de um algoritmo de geração aleatória, balanceamento e outro de ponderamento. O ModBat (E46, E47, E48) faz o uso de máquinas de estado finito estendidas para realizar a geração das sequências. O WebSpec (E50) converte cada caminho expresso na linguagem em um caso de teste. O TestMind (E51) apresenta a inserção de um conjunto de dados, para cada resposta do sistema é realizado um *screenshot* e o oráculo de teste é informado após o testador conferir o comportamento da aplicação para cada cenário. A linguagem TSL (E62, E63 e E74) faz o uso de máquinas de estado finito para a geração de sequência enquanto a DSL apresentada no estudo E65 faz o uso de uma tabela de decisões. O estudo E66 provê o uso de uma matriz de rastreabilidade entre os *scripts* e os requisitos. O ETSI-TDL (E45 e E67) faz o uso de uma ferramenta externa denominada jUCMNav para realizar a cobertura com da aplicação com base em casos de uso. Linguagem RSL utiliza da abordagem *Switch-0* responsável por gerar o máximo de sequências válidas.

QP5: *Como os dados de entrada do SUT são representados nos modelos?*

O padrão relacionado à parametrização encontrado no Mapeamento Sistemático de Literatura (SMS) se manteve nas novas DSLs descobertas através do *Snowballing*. Além disto, outras possibilidades de representar o Sistema Sob Teste (SUT) foram mencionadas como a utilização de casos de uso e programação através de instruções predefinidas.

QP6: *Quais são os benefícios, desvantagens e tendências relacionadas ao uso do DSL para testes funcionais?*

Com base nos estudos analisados é possível destacar alguns benefícios, como: 1) Inclusão dos conceitos de domínio durante a criação e especificação de casos de teste. 2) Uma maior flexibilidade e reutilização. 3) Geração de *Scripts* de teste automatizados. 4) Geração de relatórios. 5) Redução do Esforço necessário. 6) Eliminar necessidade de documentação auxiliar. 7) Envolvimento do usuário final no desenvolvimento de testes. 8) Utilização da linguagem natural no desenvolvimento de testes. 9) Melhoria na comunicação. 10) Realizar engenharia reversa da aplicação para a especificação. 11) Garantir a cobertura do sistema.

Entretanto também foram apresentados as seguintes desvantagens relacionadas às DSLs encontradas: 1) Não funcionam com aplicações dinâmicas. 2) Existe a possibilidade de várias pessoas especificarem o mesmo caso de teste com palavras diferentes, devido à possibilidade de uso da linguagem natural. 3) O tamanho da tela pode impactar no uso de algumas linguagens. 4) Alguns ambientes de modelagem apresentaram difícil usabilidade. 5) O esforço para modelagem da aplicação em alguns casos é maior do que implementar os *scripts* de teste automatizados. 6) Não suportam diversas plataformas ou Sistemas Operacionais.

Baseado nos problemas encontrados, os estudos mapeados apresentaram algumas tendências, como: 1) Utilização de testes baseados em modelos. 2) Linguagens textuais estruturadas para mitigar o problema de ambiguidade. 3) Incluir novos elementos e algoritmos relacionados à geração de casos de teste. 4) Conseguir gravar a execução no formato da linguagem. 5) Adicionar a possibilidade de uso de técnicas de análise de valor limite, teste pareado e partição de equivalência. 6) Conseguir lidar com a evolução do modelo da aplicação. 7) Utilizar engenharia reversa para algumas conversões a fim de reduzir o esforço.

3.2.8 Ameaças à Validade

Um ponto importante a ser destacado, é que podem existir outras *Domain-Specific Languages* (DSLs) não mapeadas, tendo em vista que nem todos os pesquisadores conhecem o respectivo conceito. Dessa maneira, é possível que existam outras soluções que se encaixem como DSLs. Outro ponto importante é a possibilidade das análises e categorização dos estudos sejam impactadas de acordo com nossos interesses, a fim de mitigar isto foram utilizados os critérios de inclusão e exclusão durante o processo de seleção. Outra ameaça é que o conjunto de estudos iniciais podem não retratar os trabalhos mais representativos para nossos objetivos. Entretanto, estes estudos foram extraídos com base nos estudos selecionados através do Mapeamento Sistemático de Literatura (SMS) realizado.

Uma ameaça à validade interna seria o viés dos pesquisadores referente às DSLs funcionais de *software*. Um fator a ser salientado seria a limitação a somente testes funcionais, eliminando estudos que utilizam outros tipos de teste que podem auxiliar nos testes funcionais, com isto pode existir alguma solução não investigada. Entretanto, estes

estudos deveriam ser adaptados para o contexto de testes funcionais, todos os estudos que apresentavam o uso ou proposta de uma DSL para testes funcionais foram aceitos.

Como ameaça a validade de construção é possível salientar a exclusão de algum estudo que apresentasse tanto o uso quanto uma proposta de DSL funcional para testes de *software*, para isto foram usadas regras de busca e fases bem definidas de exclusão e inclusão com buscas recursivas entre os estudos. Como ameaça externa a validade é possível apresentar que não foram levados em consideração outros possíveis estudos além do SMS realizado. Isto aconteceu devido a não terem sido encontrados outros Mapeamentos Sistemáticos de Literatura (SMSs) ou revisões à este respectivo contexto. Por fim, existe a ameaça de existirem outras DSLs para testes funcionais de *software* que não foram referenciadas ou não referenciaram nenhum dos estudos apresentados.

3.3 Discussão e Análise dos Trabalhos Relacionados

Com base nos resultados apresentados tanto do SMS quanto do *Snowballing* foi possível afirmar que ainda existem diversas desvantagens que impactam na adoção das linguagens no cenário real de desenvolvimento. A Figura 5 representa a união das Figuras 3 e 4, partindo disto é possível analisar que apesar da oscilação, DSLs para testes funcionais de *software* são um tópico de pesquisa em crescente, tanto para a tentativa de propostas quanto adoção do uso. A possibilidade de redução da complexidade, esforço e tempo necessários são os pontos fortes desta solução.

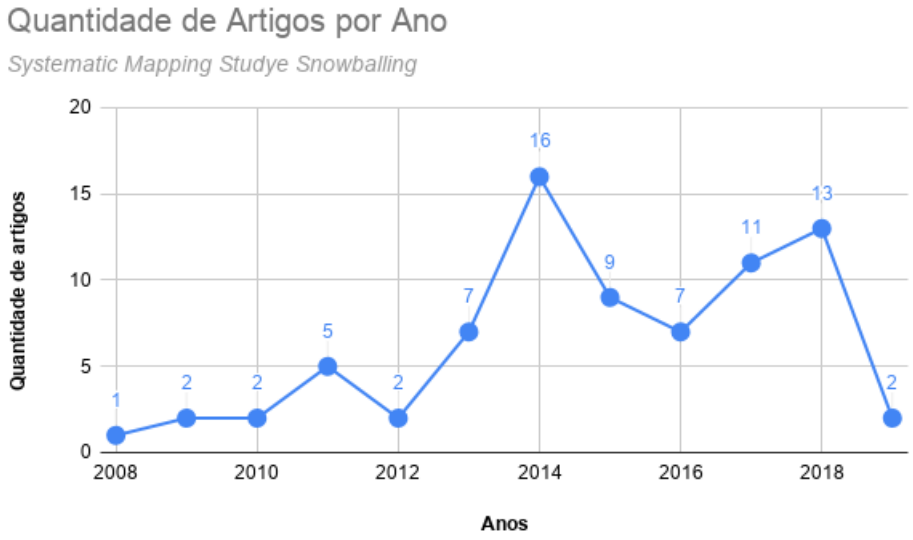


Figura 5 – Quantidade de Artigos por Ano (*Snowballing* e SMS)

As DSLs estão sendo propostas para diversos domínios diferentes inclusive sistemas críticos o que apresenta o potencial deste tipo de solução. O domínio que mais possui alternativas de linguagens é o de aplicações *web*, isto pode acontecer decorrente a constante atualização das tecnologias utilizadas. Um exemplo é a tendência das aplicações *web* serem

totalmente dinâmicas sem a necessidade de renderização sempre ao mudar de interface durante o uso do sistema. A evolução constante e a necessidade de se testar aplicações dinâmicas possibilita a Teasy um cenário ideal para sua adoção, tendo em vista que a linguagem será projetada para viabilizar atualizações constantes.

Além disto, foi encontrada uma necessidade com base no conjunto de linguagens existentes. As DSLs encontradas podem ser categorizadas em linguagens que garantem cobertura e linguagens que provêem a definição de cenários de teste de simples entendimento. Entretanto, não foi detectada uma solução textual que mitigue esta necessidade de definição tanto do sistema quanto dos cenários específicos.

3.4 Lições do Capítulo

Partindo das análises realizadas neste capítulo é possível concluir que, mesmo que existam diversas soluções, as limitações apresentadas impactam diretamente no seu uso no cenário atual. Tendo em vista a constante atualização das tecnologias e cada vez mais atualizações dinâmicas acontecendo nas aplicações *web*. A proposta da linguagem Teasy ainda é significante devido ao foco na mitigação dos problemas no cenário atual. Deste modo os requisitos, estrutura e arquitetura da Teasy foram definidos com base nos benefícios, limitações e tendências apresentadas pelos estudos. O que pode ser um fator crucial na sua adoção no cenário real de desenvolvimento.

4 TEASY *FRAMEWORK*

Este capítulo apresenta o *framework* Teasy, que é composto por uma linguagem(*Teasy Language*[1]), estrutura de execução(*Teasy Structure*[2]) e uma ferramenta responsável pela geração de sequências de teste com base nos testes criados na linguagem(*Teasy Generator*[3]). Além disto, o capítulo também contém os requisitos para a criação do *framework*, sintaxe da linguagem e exemplos de uso. Durante o desenvolvimento, uma preocupação constante era com a qualidade do código e reduzir o tempo necessário no desenvolvimento da solução, desse modo foi adotada a prática do *Pair Programming* (MCDOWELL et al., 2003) em algumas ocasiões. Um ponto importante é salientar a participação do acadêmico Samuel Modesto, durante algumas etapas de implementação da linguagem, resultando em um *framework* revisado a cada linha. Usando o *Teasy Framework* é possível programar utilizando uma linguagem projecional definindo o comportamento da aplicação sob teste. Baseado nesta definição, é possível gerar uma série de artefatos, nesta versão foram programadas as gerações de *scripts* de testes executáveis e de sequências de testes utilizando o *Teasy Generator*. Esta versão do *framework* suporta apenas aplicações *web*.

4.1 Requisitos

O Teasy *Framework* foi idealizado com o objetivo de mitigar as limitações encontradas através do SMS e *Snowballing*. Além disto outro foco da linguagem é reduzir o tempo e esforço necessário para a a criação de *scripts* de teste executáveis. Assim, viabilizando seu uso em diversos cenários de desenvolvimento. Baseado nestas informações foram criados os Requisitos (Rs) apresentados a seguir:

- **R01:** Prover o reúso de componentes dentro da linguagem, reduzindo assim o esforço necessário para a criação de testes.
- **R02:** Viabilizar a geração de artefatos de teste como *scripts* automatizados para qualquer domínio com base na aplicação definida através da Teasy. Este requisito torna possível a implementação de extensões customizáveis por terceiros de acordo com a sua respectiva necessidade.
- **R03:** Facilitar a manutenibilidade dos *scripts* de teste automatizados, tanto o gerado quanto o escrito usando a *Teasy Language*.
- **R04:** Possuir uma sintaxe simples e baseada em padrões de projeto para testes. Viabilizando o uso da linguagem por usuários com diversos níveis de conhecimento.

[1] Teasy Language: https://github.com/yuryalencar/teasy
[2] Teasy Structure: https://github.com/yuryalencar/teasystructure
[3] Teasy Generator: https://github.com/yuryalencar/teasygenerator

- **R05:** Realizar a geração automática de sequências de teste, aumentando assim a cobertura da aplicação.
- **R06:** Gerar os *scripts* de teste para uma estrutura escalável e de fácil manutenção.
- **R07:** Gerar um relatório com informações precisas e suficientes para realizar uma análise dos testes criados e gerados.

4.2 Visão Geral do *Framework*

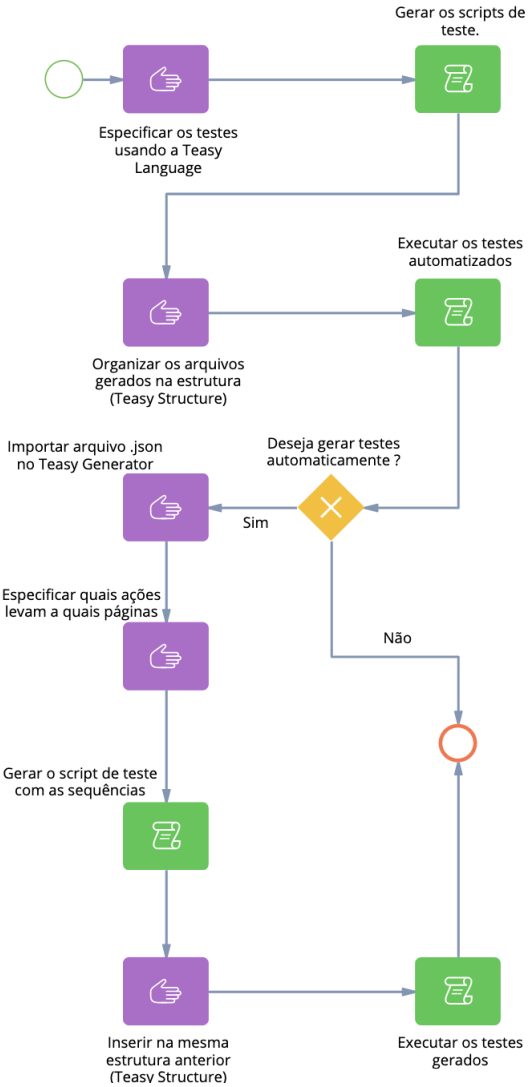
O *Teasy Framework* possui como principal foco realizar testes funcionais em aplicações *web*, entretanto é necessário saber quando usar cada parte do *framework*. Assim, esta seção tem como objetivo apresentar quais atividades e quais recursos utilizar para aproveitar o máximo que o *framework* tem a oferecer. A seguir são apresentadas as atividades sequencialmente, como ilustrado na Figura 6.

- **Especificar os testes:** Utilizando a *Teasy Language* é possível especificar quais páginas existem na aplicação e quais são ações que cada página pode executar. Além disto é possível também criar testes, ou seja, fluxos de execução dentro da aplicação. A *Teasy Language* será apresentada melhor na seção 4.3 e na seção 4.6.
- **Gerar *scripts* de teste:** Utilizando como entrada um *script* em um alto nível de abstração, escrito na *Teasy Language*, é possível gerar *scripts* de testes automatizados executáveis.
- **Organizar arquivos gerados:** Utilizando a *Teasy Structure* é necessário organizar os arquivos para que todas as importações funcionem como o esperado. Mais detalhes sobre a *Teasy Structure* serão apresentados na seção 4.4 e na seção 4.6.
- **Executar os testes:** Após os arquivos serem organizados é possível executar os *scripts* de testes gerados, desde que definido algum fluxo na aplicação sob teste na *Teasy Language*.
- **Importar arquivo *JavaScript Object Notation* (JSON):** Caso o testador opte por gerar os testes automatizados basta inserir o conteúdo do arquivo com extensão *.json* no *Teasy Generator*. Mais detalhes sobre o *Teasy Generator* serão apresentados na seção 4.5 e na seção 4.6.
- **Especificar quais ações levam a quais páginas:** Utilizando o *Teasy Generator* o testador pode especificar quais ações levam a quais páginas de forma simples através de um campo de seleção por ação.
- **Gerar o *script* de teste com novas sequências:** Após especificar todas as ações e caminhos considerados importantes, basta clicar em um botão e o *Teasy Generator* irá exportar um arquivo compatível com a *Teasy Structure*.

- **Inserir na mesma estrutura os testes gerados:** Possuindo o arquivo gerado falta somente adicioná-lo na estrutura com os demais arquivos gerados pela *Teasy Language*.

- **Executar os testes gerados:** Por fim, o arquivo gerado pelo *Teasy Generator* irá se comunicar com os demais gerados anteriormente, podendo ser executado como um *script* de teste. Ao final de qualquer execução é gerado um relatório para que o testador possa acompanhar o resultado final.

Figura 6 – Visão geral do uso do *Teasy Framework*.



Fonte: Criada pelo Autor

4.3 Teasy Language

A *Teasy Language* é uma linguagem de domínio específico do tipo projecional, ou seja, ela pode fazer o uso tanto de elementos gráficos quanto textuais. Além disso, utilizando o uma linguagem projecional a lógica e a parte de interação com o usuário são implementadas separadamente, viabilizando atualizações de usabilidade e até mesmo trocando textos por símbolos sem impactar na definição da linguagem em si. Mesmo que a solução permita mesclar símbolos com textos, a *Teasy Language* projeta somente textos, esta escolha foi realizada de acordo com o estudo de (TÖRSEL, 2013), que menciona que linguagens textuais são mais produtivas em relação às gráficas no domínio de testes.

Desse modo, o editor implementado faz o uso somente de textos. A fim de implementar a *Teasy Language* foi utilizado o MPS como LW, sua escolha foi decorrente a realização de um estudo comparativo entre tecnologias de desenvolvimento de DSLs, onde o MPS apresentou superioridade nos requisitos de menor esforço e maior praticidade no uso (LIMA et al., 2019). Além disto, o uso do MPS como ambiente de desenvolvimento possibilita a utilização de atalhos e preenchimento automático de alguns elementos, aumentando a produtividade durante a programação dos testes usando a *Teasy Language*.

4.3.1 Sintaxe

Com o intuito de facilitar a compreensão da linguagem, a sintaxe foi subdividida com base nos tipos de arquivos existentes. Estes arquivos em conjunto é a instanciação do padrão de projeto para testes funcionais *Page Object Models* (RAGHAVENDRA, 2021), sendo um padrão recomendado pelos próprios desenvolvedores do Selenium (DEV, 2021).

4.3.1.1 Tipos de Arquivos

Com o intuito de implementar os testes de uma aplicação usando a *Teasy Language*, foram criados cinco tipos de arquivos, que se relacionam, garantindo o reúso e redução de esforço necessário com a duplicação de componentes, atendendo ao requisito **R01**. Todos os arquivos a seguir são obrigatórios e precisam ser implementados de modo correto para que ocorra a compilação e geração dos *scripts* de teste executáveis. Um ponto importante é que os arquivos não necessitam de uma ordem de criação, possibilitando uma maior liberdade ao usuário permitindo criar seus elementos dinamicamente de acordo com a sua necessidade. Todos os arquivos e dados necessários são apresentados a seguir .

4.3.1.1.1 *Configuration*

O arquivo *Configuration* deve ser único no projeto, e é responsável por garantir a configuração necessária para os *scripts* de teste gerados. A partir deste arquivo são geradas as configurações e os *hooks* dos *scripts* de teste. Os *hooks* são a responsáveis por

abrir/fechar o navegador e tirar *screenshots* da tela ao final dos testes, melhorando assim o relatório gerado ao final da execução e contribuindo para atender ao requisito **R07**. A seguir são apresentadas as informações necessárias para criação deste arquivo dentro da *Teasy Language*:

- ***Browser***: Esta informação é referente a qual navegador os testes gerados deverão executar, atualmente suportando o Mozilla Firefox (FIREFOX, 2008) e Google Chrome (FIREFOX, 2018).
- **Tempo de espera implícita**: Este dado é um número em segundos representando o tempo máximo de espera para que um elemento esteja no HTML da página *web*.
- **Máximo de testes a ser gerado**: Esta informação não esta sendo utilizada atualmente no gerador, mas é um número que representa a quantidade máxima de testes a ser gerado. Este valor poderá ser utilizado para calibrações futuras no gerador.
- **URL da primeira página**: Esta informação contém a URL da primeira página da aplicação sob teste. Todos os testes deverão ser desenvolvidos a partir dessa página.
- **Altura e Largura do navegador**: Esta informação representa a resolução na qual o navegador deve estar durante a execução dos testes. Desse modo viabilizando a replicação dos testes automatizados com uma resolução mais próxima do cliente final.

4.3.1.1.2 *PageRegisterConfig*

O arquivo *PageRegisterConfig* também deve ser único para cada projeto, e possui um papel fundamental para que os *scripts* de testes executáveis funcionem como o esperado. Dentro deste arquivo devem ser inseridos os nomes de todas as *Pages* criadas (Seção 4.3.1.1.4), este ponto é fundamental para que as importações funcionem dentro da TeasyStructure(Seção 4.4).

4.3.1.1.3 *Components*

O arquivo *Components* também deve ser único por projeto, nele são inseridos todos os componentes envolvidos nos testes realizados. A partir deste arquivo é possível realizar interações com os componentes, como: inserir valores, clicar, esperar algum estado e até mesmo alterar sua estrutura se necessário. A seguir são apresentadas as informações que devem ser inseridas para cadastrar um componente.

- **Nome**: Esta informação é referente ao nome do componente, não podendo conter espaços e deve ser única, ou seja, não permitindo a duplicação dos nomes entre os

componentes. Esse dado é importante, pois toda vez que for utilizar este componente este nome será referenciado.

- **Seletor**: Este dado é referente ao seletor utilizado para encontrar o componente, sendo possível escolher entre: id, name, css, class e xpath.
- **Valor do seletor**: Este valor do seletor é extraído do HTML da página, a Teasy utilizará este valor em conjunto com o seletor escolhido para detectar o componente na página.

4.3.1.1.4 *Page*

O arquivo *Page* não é único dentro do projeto, para cada página da sua aplicação é recomendado que se crie um arquivo deste tipo. Neste arquivo são inseridas as ações que podem ser realizadas através de uma página específica, como: realizar uma autenticação, realizar o cálculo de soma, dentre outras. As ações criadas neste arquivo são compostas por um conjunto de manipulações realizadas nos componentes definidos no arquivo de *Components* (Seção 4.3.1.1.3). A seguir são apresentadas todas as informações para criar uma página dentro da *TeasyLanguage*.

- **Nome**: Nome da página, não pode conter espaços e é utilizado no arquivo *PageRegisterConfig* (Seção 4.3.1.1.2) viabilizando as importações dos *scripts* executáveis gerados.
- **Ação**: Podem existir várias ações e de preferência devem ser isoladas. A seguir, a seguir são apresentadas as informações presentes em uma ação:
  - **Nome**: Nome da ação, este valor pode conter espaços e ele é utilizado nos arquivos de *Flows* (Seção 4.3.1.1.5) para referenciar as ações necessárias para criar um fluxo de testes.
  - **Manipulações**: Dentro de uma ação é possível realizar as interações com um componente, é possível fazer quinze diferentes tipos de manipulações, dentre elas é possível citar: clicar em um elemento, inserir texto, esperar algum estado do componente, etc. Em cada manipulação são inseridos o nome do componente, e a informação adicional de acordo com a respectiva manipulação. Uma exceção entre as manipulações é a possibilidade de executar *Javascript* e esperar por uma condição em *Javascript*, estas interações não necessitam de um componente, somente o código a ser executado. Um detalhe importante é que não existe limite para a quantidade de interações e nem de tipos de manipulações que podem existir dentro de uma ação, a quantidade é definida pelo próprio testador.

4.3.1.1.5 *Flows*

O arquivo *Flows* também não é único dentro do projeto, para cada conjunto de testes é recomendado que se crie um arquivo deste tipo. Neste arquivo são criadas as sequências de teste a partir da página inicial da aplicação. As ações criadas nos arquivos de *Pages* (Seção 4.3.1.1.4) são utilizadas para criar um fluxo de teste permitindo assim um maior reúso, sem a necessidade de replicação de código. A seguir são apresentadas as informações necessárias para criar um arquivo *Flows*:

- **Nome do arquivo**: É um nome sem espaços, que representa o conjunto de fluxos de teste. Este nome é utilizado para a geração de *scripts* de testes executáveis como o nome do arquivo.
- **Nome do fluxo**: Nome da sequência de testes, este nome pode conter espaços e é recomendado ser único e representativo para cada arquivo. Podem existir um ou mais fluxos dentro de um único arquivo de *Flows*.
- **Ações**: Dentro de cada fluxo, é possível ter uma ou mais ações, as ações são informadas sequencialmente definindo um fluxo de teste a ser executado. Um ponto importante é que cada ação utilizada neste arquivo deve estar presente em algum arquivo do tipo *Page*.

4.4 Teasy Structure

A *Teasy Structure* é uma estrutura composta por um conjunto de pastas e arquivos para explicitar quais arquivos vão em quais diretórios. Nesta primeira versão é necessário inserir os *scripts* de teste executáveis em seu respectivo diretório manualmente. O uso da *Teasy Structure* é crucial para que os arquivos sejam executados, pois os arquivos também são reutilizados entre si, desse modo esta estrutura permite que todas as importações funcionem como o esperado.

Devido os *scripts* de teste gerados pela *Teasy Language* utilizarem o *Robot Framework*, o seu código em conjunto com a *Teasy Structure* tende a estar mais modularizado e possuir também uma manutenção facilitada direto no código final se necessário. A seguir são apresentados os diretórios presentes na estrutura:

- **config:** Neste diretório são inseridos os arquivos de configuração gerados pela *Teasy Language*, sendo estes os arquivos: *config.robot* e o *page_register.config.robot*.
- **components:** Neste diretório são inseridos todos os componentes da aplicação sob teste. Estes componentes estão presentes no arquivo *components.robot*.
- **tests:** Neste diretório são inseridos todos os arquivos de teste (com extensão *.tests.robot*).

- **pages:** Neste diretório são inseridas todas as páginas da aplicação conforme o padrão *Page Objects*, estes arquivos possuem a extensão *.pages.robot*.

  - **commons:** Neste diretório são inseridos os arquivos utilizados em todas as pages atualmente somente o arquivo *hooks.pages.commons.robot* atende a esse requisito.

4.5 Teasy Generator

Além da possibilidade de criar seus casos de teste, executá-los com uma estrutura escalável, o *Teasy Framework* também permite a geração de sequências de teste através do *Teasy Generator*. O *Teasy Generator* é uma aplicação *web* onde é possível informar através de um arquivo JSON, quais são as páginas e as respectivas ações de uma aplicação sob teste. A partir destas informações basta escolher a primeira página da aplicação e selecionar a próxima página que cada ação redireciona. Ao selecionar uma outra página a mesma é inserida abaixo de uma lista, permitindo continuar explicitando quais páginas suas ações acarretam.

É possível que uma ação redirecione para a mesma página. Nestes casos a página é duplicada, entretanto pelo cabeçalho criado a cada inserção na lista é possível identificar o caminho percorrido até aquela página. Ou seja, qual o estado da aplicação naquele respectivo momento. Ao finalizar esta especificação basta clicar em um botão para gerar as sequências de teste baseado nos possíveis caminhos explicitados.

O *Teasy Generator* vai transformar os possíveis caminhos informados em uma estrutura de grafo (BRODER et al., 2000) e utilizar um algoritmo baseado em busca em profundidade (TARJAN, 1972) para gerar sequências que fazem sentido no contexto da aplicação. A partir destas sequências é gerado um *script* de teste executável e totalmente compatível com o código gerado pela *Teasy Language*. O código gerado também está pronto para ser utilizado com a *Teasy Stucture* respeitando todas as importações necessárias. Outros detalhes de uso e exemplos de tela serão apresentados em 4.6.3.

4.6 Exemplo de Uso

Com o objetivo de apresentar um exemplo de uso foi escolhida uma aplicação simples, sendo esta uma calculadora desenvolvida para o domínio *web*. Mesmo sendo uma simples calculadora as tecnologias envolvidas ainda são utilizadas em aplicações reais. Esta implementação pode ser encontrada através de um repositório de código aberto denominado ReactJSCalculator[4]. A seguir serão apresentados maneiras de como criar um teste para a calculadora, neste caso foi escolhida uma soma entre dois números.

[4] ReactJSCalculator: https://github.com/yuryalencar/ReactJSCalculator

4.6.1 Como Utilizar a *Teasy Language*

A *Teasy Language* controla todo o fluxo de testes, desde como as configurações até a execução. Sabendo disso, é aconselhável que se inicie o projeto pelas configurações para garantir o funcionamento esperado. De acordo com o apresentado em 4.3 o arquivo presente na Figura 7 representa o seguinte significado: os testes implementados serão executados no Google Chrome, implicitamente será esperado até cinco segundos para encontrar um elemento no HTML, os testes devem começar no endereço *http://localhost:3000/* e por fim o monitor do cliente possui as respectivas resoluções.

Figura 7 – Arquivo *Configuration*

```
CONFIGURATIONS

  INSERT BROWSER TO RUN TESTS: chrome
  INSERT MAX TIME TO WAIT ELEMENT (SECONDS): 5
  MAX AMOUNT TEST TO GENERATE: 300
  URL TO ROOT PAGE: http://localhost:3000/

  BROWSER CUSTOMER WIDTH: 1080
  BROWSER CUSTOMER HEIGHT: 720
```

Fonte: Criada pelo Autor

Após os testes configurados, o próximo passo é identificar os elementos envolvidos nos testes e criar um arquivo *Components*, este arquivo é único na aplicação e contém todos os elementos de todas as páginas. A Figura 8 contém os componentes envolvidos em uma soma com resultado cinco, para encontrar o seletor é necessário inspecionar os elementos no HTML.

Com os componentes definidos é necessário criar um arquivo do tipo *Page* que representa uma página da aplicação sob teste. Neste caso, existe somente uma página sendo essa denominada *CalculatorHomePage*. Essa página contém as ações relacionadas a página como clicar no botão cinco, clicar no botão mais, entre outras ações possíveis. A Figura 9 apresenta um exemplo de uma página.

Sempre que um arquivo do tipo *Page* for criado é necessário adicionar ele ao arquivo *PageRegisterConfig* para que todas as importações funcionem como o esperado durante a execução dos testes. Caso uma *Page* seja criada e não adicionada neste arquivo, ela não poderá ser utilizada para criar uma sequência de teste. A Figura 10 contém o exemplo deste arquivo contendo a única página presente na aplicação ReactJSCalculator.

Por fim, é necessário criar um arquivo do tipo *Flows*, neste arquivo é possível criar sequências de teste conforme a necessidade do testador. Os arquivos contém um vínculo com as ações cadastradas nos arquivos de *Pages*, assim para criar um fluxo o arquivo requisita um nome para a(s) sequência(s), além das ações que devem ser realizadas. É

Figura 8 – Arquivo *Components*

```
SYSTEM COMPONENTS

  INSERT NAME COMPONENT: BotaoAC
  INSERT SELECTOR: id
  INSERT SELECTOR VALUE: ac

  INSERT NAME COMPONENT: BotaoDois
  INSERT SELECTOR: id
  INSERT SELECTOR VALUE: 2

  INSERT NAME COMPONENT: BotaoTres
  INSERT SELECTOR: id
  INSERT SELECTOR VALUE: 3

  INSERT NAME COMPONENT: BotaoMais
  INSERT SELECTOR: id
  INSERT SELECTOR VALUE: +

  INSERT NAME COMPONENT: BotaoIgual
  INSERT SELECTOR: id
  INSERT SELECTOR VALUE: =

  INSERT NAME COMPONENT: DisplayComponent
  INSERT SELECTOR: class
  INSERT SELECTOR VALUE: display
```

Fonte: Criada pelo Autor

importante salientar que o fluxo deve ser criado a partir da primeira página da aplicação. A Figura 11 apresenta um exemplo do cálculo utilizando o arquivo de fluxos.

4.6.2 Como Executar os *Scripts* de Teste Usando a *Teasy Structure*

Depois de possuir suas sequências e páginas especificadas utilizando a *Teasy Language* para executar os testes é necessário gerar os *scripts* executáveis, este passo pode ser verificado na figura 12. Após realizar a seleção mencionada o MPS será responsável por gerar um conjunto de arquivos com a extensão *.robot*.

O acesso a estes arquivos é realizado através do diretório do projeto *Teasy* presente em seu computador. Atualmente, os arquivos ainda não são gerados diretamente na *Teasy Structure* sendo esta uma etapa manual. A primeira etapa para adicionar os arquivos à estrutura é entrar no diretório *Teasy* do seu computador e entrar nas respectivas pastas: *languages* > *Teasy* > *sandbox* > *source_gen* > *Teasy* > *sandbox*. Caso seus arquivos não estejam presentes é possível realizar um *Rebuild Model 'Teasy.sandbox'* e eliminar os problemas relacionados ao *cache* do computador, como realizar esta ação pode ser visualizado na Figura 12.

Figura 9 – Arquivo *Page*

```
PAGE NAME: CalculatorHomePage

  ACTION: Clicar no botao dois $<
    CLICK ELEMENT:  BotaoDois

  >$
  ACTION: Clicar no botao tres $<
    CLICK ELEMENT:  BotaoTres

  >$
  ACTION: Clicar no botao mais $<
    CLICK ELEMENT:  BotaoMais

  >$
  ACTION: Clicar no botao igual $<
    CLICK ELEMENT:  BotaoIgual

  >$
  ACTION: Verificar soma 5 $<
    WAIT FOR CONDITION
    CONDITION SCRIPT:  return document.getElementsByClassName('display')[0].textContent === "5"
    TIMEOUT:  1

  >$
```

Fonte: Criada pelo Autor

Figura 10 – Arquivo *PageRegisterConfig*

```
REGISTER PAGES:

    PAGE TO REGISTER: CalculatorHomePage
```

Fonte: Criada pelo Autor

Após conter os arquivos gerados é necessário clonar o projeto *TeasyStructure*[5] e na extensão do arquivo é possível identificar qual diretório o mesmo pertence. Como por exemplo o arquivo "*hooks.**pages.commons.robot***" ficará dentro da pasta *pages* e da pasta *commons*, esta lógica é replicada para todos os arquivos o resultado pode ser visualizado através da Figura 13.

Com os arquivos em seus respectivos diretórios é possível executar os *scripts* de

[5] Teasy Structure: https://github.com/yuryalencar/teasystructure

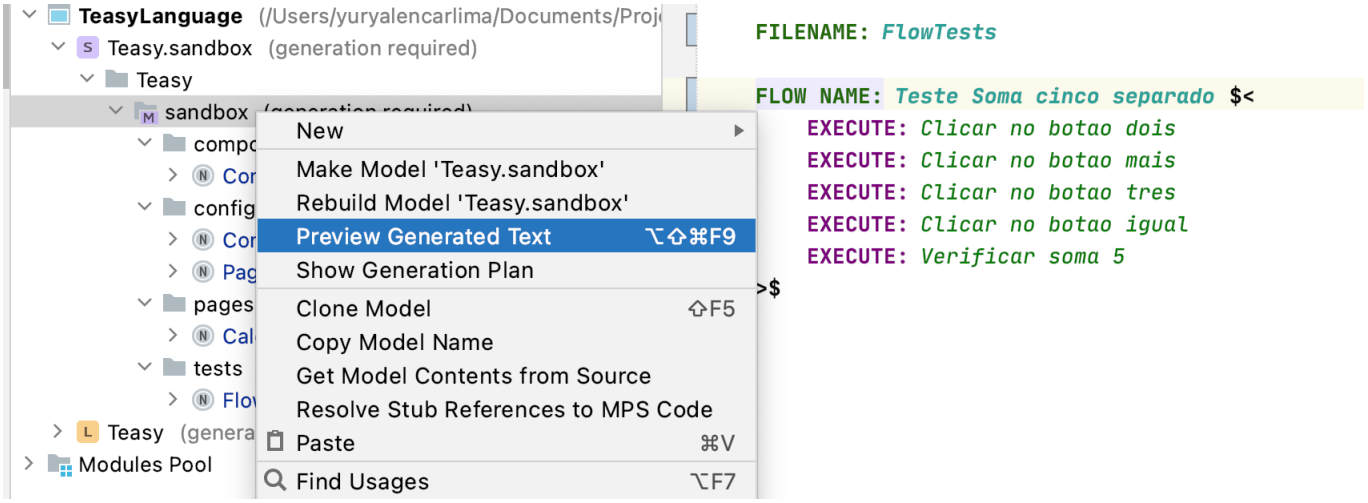
Figura 11 – Arquivo *Flows*

```
FILENAME: FlowTests

FLOW NAME: Teste Soma cinco separado $<
    EXECUTE: Clicar no botao dois
    EXECUTE: Clicar no botao mais
    EXECUTE: Clicar no botao tres
    EXECUTE: Clicar no botao igual
    EXECUTE: Verificar soma 5
>$
```

Fonte: Criada pelo Autor

Figura 12 – Gerando os arquivos



Fonte: Criada pelo Autor

testes gerados usando o comando a seguir na raiz do projeto: "*robot -d ./logs tests*". Com este comando serão executados todos os fluxos especificados e para cada fluxo será aberto um novo navegador e fechado ao final da execução, isolando cada sequência de testes. Ao final da execução será gerado um diretório denominado *logs* contendo um relatório em HTML navegável com os resultados dos testes incluindo capturas de tela.

4.6.3 Como Gerar Sequências Usando o *Teasy Generator*

Caso o testador possua interesse em gerar *scripts* de teste com sequências geradas automaticamente o *framework* também disponibiliza o *Teasy Generator*. Durante a geração de *scripts* usando a *Teasy Language* um arquivo denominado *fsm.json* é criado, este arquivo contém as *Pages* definidas e quais ações elas possuem, este arquivo será crucial

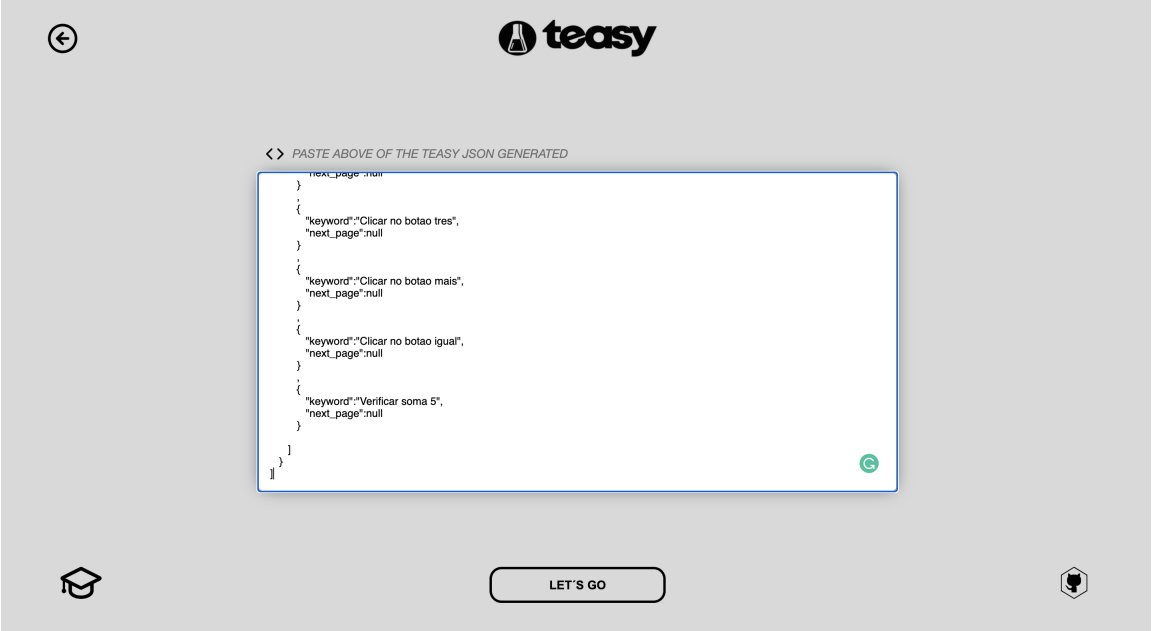
Figura 13 – Resultado após adicionar os arquivos à *TeasyStructure*

```
|~components/
| `-components.robot
|~config/
| |-config.robot
| `-page_register.config.robot
|~pages/
| |~commons/
| | `-hooks.pages.commons.robot
| `-CalculatorHomePage.pages.robot
|~tests/
| `-FlowTests.tests.robot
|-LICENSE
`-README.md
```

Fonte: Criada pelo Autor

para a geração de sequências. A fim de inserir as informações do arquivo *fsm.json*, ao abrir o *Teasy Generator* basta clicar em *start* e inserir o conteúdo do arquivo conforme o apresentado na Figura 14.

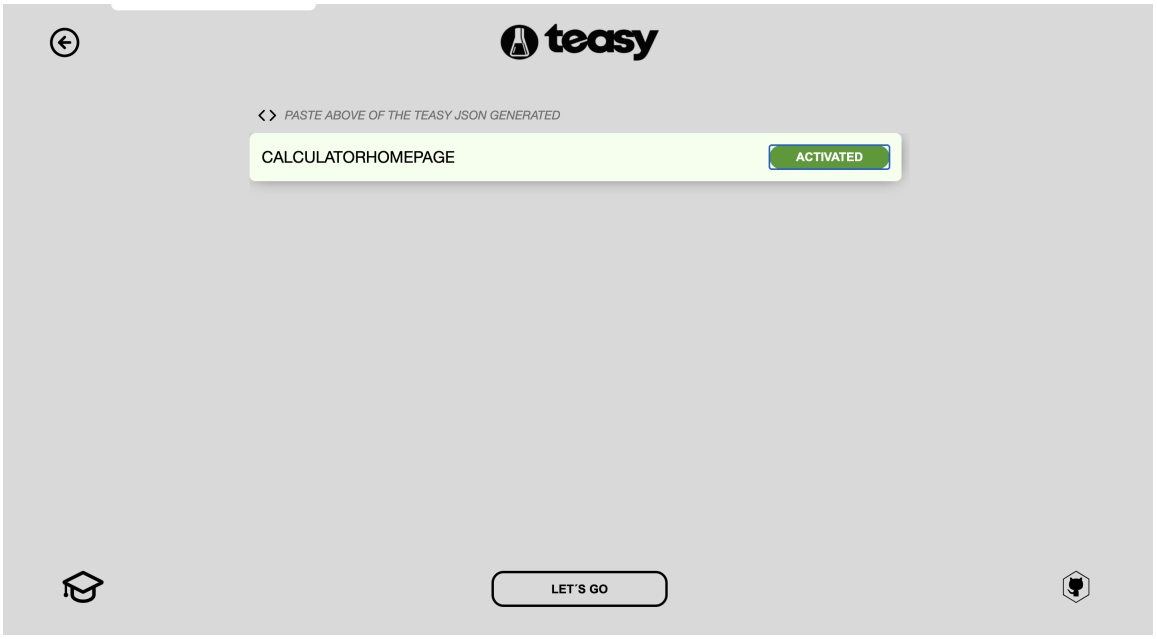
Figura 14 – Adicionando as *Pages* no *Teasy Generator*



Fonte: Criada pelo Autor

Ao clicar em *Let's go* o *Teasy Generator* pede para que o usuário informe a primeira página da aplicação, conforme a Figura 15, como o exemplo utilizado só possui uma página aparece somente um opção. Esta escolha da página inicial é importante, pois define quais ações e qual página será utilizada como base para iniciar as sequências geradas, ou seja, todas as sequências terão início na página escolhida nesta etapa.

Figura 15 – Escolhendo a primeira pagina no *Teasy Generator*



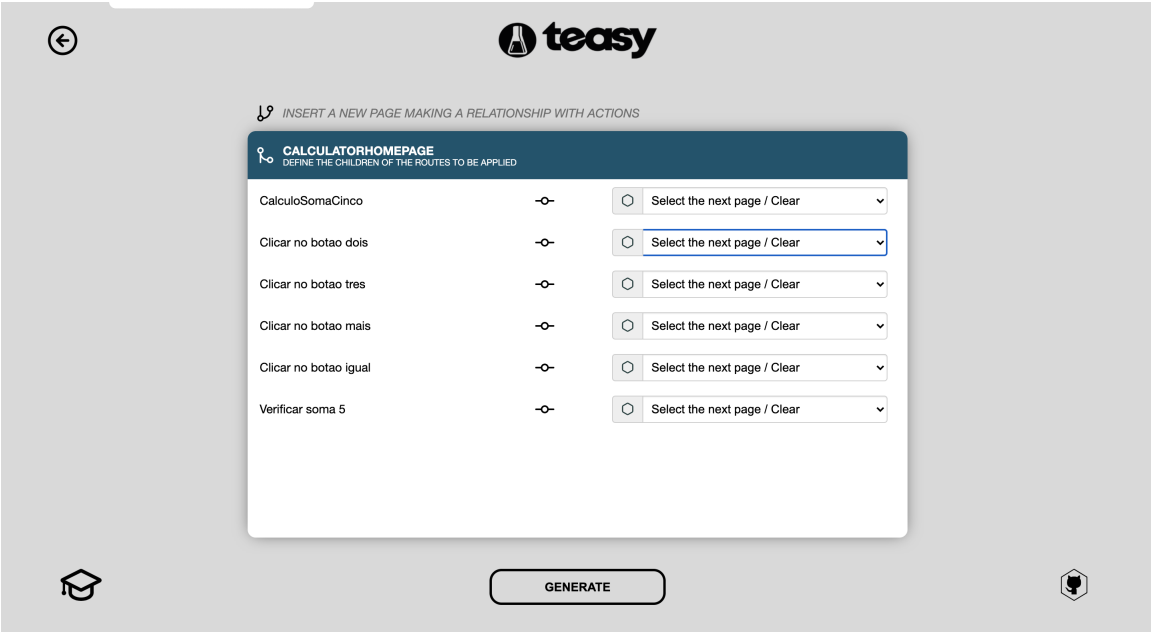
Fonte: Criada pelo Autor

Após ter escolhido a primeira página o *Teasy Generator* irá apresentar todas as ações pertencentes a primeira página, e um campo a direita para que possa selecionar para qual página a respectiva ação leva. Como a nossa aplicação é somente de uma página ela sempre será a escolhida, não é necessário escolher todas as opções o *Teasy Generator* utilizará as possibilidades disponíveis. Um ponto importante é que sempre que escolher uma respectiva página a mesma será inserida abaixo da última para possibilitar novas escolhas. Caso o usuário selecione alguma página já escolhida anteriormente a página é duplicada e inserida ao final da fila, entretanto em seu cabeçalho é apresentado o seu estado atual, ou seja, todas as ações realizadas até o respectivo momento. Por fim, ao finalizar a escolha das sequências consideradas mais relevantes, é possível clicar em *Generate* que o *Teasy Generator* irá converter essas informações para uma estrutura de dados em árvore e gerar um *script* de teste executável usando um algoritmo de busca em profundidade. O *script* gerado pode ser inserido na pasta *tests* presente no projeto criado usando a *TeasyStructure* e executado como os demais arquivos de teste.

4.7 Lições do Capítulo

Tendo em vista o *framework* apresentado neste capítulo é possível concluir que o *Teasy Framework* foi idealizado para prover o máximo de reúso e flexibilidade para o testador, além de tender a reduzir o esforço necessário para criação de *scripts* de testes executáveis. Além disso, é possível utilizar de forma isolada as soluções, o que viabiliza o

Figura 16 – Escolhendo os fluxos no *Teasy Generator*



Fonte: Criada pelo Autor

seu uso em diversos contexto, sendo esta uma limitação de outras soluções encontradas nos trabalhos relacionados. Os pontos positivos salientados nesse capítulo são apresentados a seguir:

- **Redução do esforço**: Usando a *Teasy Language* isto tende a ser possível, pois a mesma foi implementada para projetar as informações automaticamente para o usuário de acordo com o arquivo e contexto, reduzindo a quantidade de teclas necessárias para implementar uma sequência específica.

- **Reúso**: O reúso de componentes e atalhos presentes na *Teasy Language* são refletidos na *TeasyStructure*, reduzindo a quantidade de código necessário.

- **Manutenibilidade**: O código gerado pela *Teasy Language* tende a possuir uma fácil interpretação viabilizando as manutenções direto no código gerado caso necessário.

- **Soluções podem ser usadas sozinhas**: A *Teasy Sctructure* foi projetada usando o padrão *Page Objects*, possibilitando seu uso em outros projetos. A *Teasy Language* possui um facilitador na execução utilizando a *Teasy Structure*, mas pode ser atualizada para funcionar em outras estruturas se necessário. Por fim, o *Teasy Generator* pode ser utilizado separadamente, caso o testador possa prover um arquivo JSON no formato correto é possível gerar sequências de teste executáveis.

Mais detalhes sobre os benefícios e até limitações do *framework* serão apresentados nos capítulos 5 e 6.

5 USANDO O FRAMEWORK TEASY PARA DETECTAR DEFEITOS: UM *QUASI*-EXPERIMENTO

Este capítulo tem o intuito de apresentar o *quasi*-experimento realizado, comparando o *Teasy Framework* com o *Robot Framework*. Assim, resultando em uma análise comparativa da solução proposta nesse estudo. Dessa forma provendo alguns indícios relacionados à aceitação, usabilidade e representatividade do *framework* por parte dos possíveis usuários finais.

5.1 Definição do Quasi-experimento

Afim de auxiliar no desenvolvimento da avaliação da *Teasy Language* foi utilizado como apoio o Usa-DSL *Process* (POLTRONIERI et al., 2018) que guia o desenvolvimento de avaliações de usabilidade para DSLs. A motivação para realização deste *quasi*-experimento foi avaliar o esforço ao usar o *framework* Teasy, ou seja, a linguagem e a ferramenta de geração de sequências e *scripts* de teste automatizados. Como mencionado, o processo de teste de *software* demanda tempo, sendo uma das fases mais caras do processo de desenvolvimento de *software*. A principal vantagem de usar uma DSL é que ela representa o domínio dos testadores e tende a reduzir o tempo necessário para implementar os casos de teste. Além de fornecer uma geração automática de *scripts* de teste, reduzindo assim o esforço durante a atividade de teste de *software*. No entanto, os testadores enfrentam alguns problemas importantes quando precisam decidir se é adequado ou não usar uma linguagem específica de domínio: o esforço para configurar e aprender a utilizar a linguagem, é menor do que desenvolver os casos e *scripts* de teste diretamente com uma ferramenta ou *framework* tradicional? Ou seja, qual é a evidência de que a DSL melhora a produtividade geral dos testadores? A DSL também é capaz de detectar problemas funcionais nas aplicações *web* ?

A fim de responder estas perguntas, foram utilizadas métricas tanto qualitativas como as de usabilidade, quanto quantitativas como o esforço (tempo gasto) ao executar um conjunto de tarefas pré-definidas [1]. Estas métricas foram coletadas tanto usando o *framework* Teasy quanto um outro *framework* para automação de testes funcionais para aplicações *web*. Como *framework* de testes base, foi escolhido o *Robot Framework* (BISHT, 2013), que é amplamente utilizado por várias empresas para testar aplicativos *web*. No contexto do *quasi*-experimento, cada tarefa de teste é composta por um ou mais casos de teste, que são compostos por uma sequência de etapas. Assim, cada etapa foi usada para simular cada interação do usuário com o SUT. Desse modo, quando um sujeito do *quasi*-experimento executa uma tarefa usando a linguagem Teasy corretamente, é possível obter um *script* equivalente e cenário produzido ao executar a mesma tarefa corretamente usando o *Robot Framework*. Portanto, ambos os *scripts* devem exercer a mesma funcionalidade no SUT, por exemplo, realizar a autenticação em uma aplicação. Um ponto

[1] Tarefas realizadas: https://url.gratis/giwi3, último acesso em 24/02/2021

importante é que com a execução correta é possível reproduzir os mesmos resultados, para uma comparação significativa, foram observadas as taxas de erros cometidos pelos sujeitos durante a execução das tarefas.

Resumindo, o objetivo deste estudo experimental é coletar evidências sobre a facilidade de uso e o esforço para aplicar testes funcionais *web* ao usar o *framework* Teasy e ao usar um *framework* tradicional. O objeto de estudo é o esforço de usar duas semânticas diferentes, para criar *scripts* e cenários de teste. O objetivo do experimento é avaliar o esforço para gerar *scripts* de testes funcionais e cenários ao usar a linguagem Teasy. A principal motivação é entender qual é o esforço exigido por uma equipe de teste para criar *scripts* usando as tecnologias Teasy e *Robot Framework*. Por se tratar de um *framework* composto por uma DSL e um gerador de sequências, a Teasy tende a apresentar várias vantagens, como melhor cobertura de teste (em relação ao número de casos de teste gerados que representam mais interações do usuário), mas neste experimento nos concentramos em compreender o esforço para criar scripts de teste e cenários , usando os *frameworks* Teasy e Robot, a partir dos casos de teste.

5.2 Descrição das Tecnologias e Ferramentas Envolvidas

5.2.1 Robot Framework

O *Robot Framework* é uma tecnologia *Open source* com o objetivo de realizar a criação de testes de aceitação de sistema. O seu uso é baseado em uma sintaxe tabular e de fácil aprendizado (BISHT, 2013). O *framework* também utiliza de linguagem natural tanto para a definição das *Keywords* que são como métodos de teste, quanto para os *Steps* que são os passos dentro de cada *Keyword*. Dentre os pontos positivos para seu uso, destacam-se a compatibilidade com várias bibliotecas externas e a possibilidade de criação de bibliotecas para domínios específicos (BISHT, 2013). Essa personalização pode ser realizada por meio de *scripts* criados em linguagem Python (LUTZ, 2001) ou em Java (GOSLING et al., 2000). Nesse quasi-experimento foi utilizado o *Robot Framework* devido a ser um *framework* popular de fácil aprendizado que possibilita o desenvolvimento de testes para aplicações *web*, sendo compatível com a linguagem Teasy. A tecnologia foi utilizada juntamente com o Python decorrente da grande quantidade de bibliotecas de suporte para o respectivo domínio.

O *Selenium WebDriver* foi utilizado em conjunto com o *Robot framework* por ser uma tecnologia responsável por realizar as manipulações dos navegadores, por meio de interações com a *interface* gráfica da aplicação *web*, representando as ações do usuário (AVASARALA, 2014). A tecnologia captura os elementos gráficos por meio dos atributos presentes dentro do HTML), que por sua vez representam a *interface* com o usuário, viabilizando a realização de manipulações e ações com esses elementos. Assim, decorrente as funcionalidades presentes, a adoção desta tecnologia dentro do *quasi*-experimento foi

necessária, além de possuir uma biblioteca nativa para o *Robot Framework*. O uso do *Selenium WebDriver* para automação de testes *web* também é uma recomendação da World Wide Web Consortium (W3C) sendo esta a principal organização de padronização de aplicações *web* (W3C, 2018).

5.2.2 Teasy Framework

Como apresentado no Capítulo 4 deste estudo o *Teasy Framework* é uma solução para testes automatizados em aplicações *web*. Provendo funcionalidades como geração de sequências de teste, relatórios, configuração de todo o ambiente de testes, além da especificação de fluxos de teste personalizados.

5.2.3 *System Under Testing*

Durante a avaliação, foram utilizadas duas aplicações *web* como SUT: ReactJS-Calculator[2] e o *Shopping Cart Software*[3]. O ReactJSCalculator é uma aplicação *web* de baixa complexidade que realiza operações matemáticas simples. Não é uma aplicação utilizada na indústria mas possui um comportamento interessante quando utilizada no padrão *Page Objects*, por existir diversos estados diferentes. Esta aplicação foi desenvolvida em 2019, onde ainda utiliza um padrão de desenvolvimento de *Single Page Application* (SPA) considerado antigo devido as constantes atualizações do *framework* de desenvolvimento utilizado.

Shopping Cart Software é um *e-commerce*, que possibilita realizar um cadastro de novos usuários e edição dos seus respectivos dados. Esta aplicação possui todos os componentes de aplicações existentes na indústria, mas durante o *quasi*-experimento foi utilizada uma versão de demonstração. Entretanto, o uso de suas funcionalidades não foram restringidos, através da aplicação é possível criar uma compra de um novo produto, editar esta compra, verificar o carrinho, pesquisar por novos produtos, dentre outras funcionalidades. Por fim, é importante salientar que o *Shopping Cart Software* foi usado como SUT durante a execução do experimento e ReactJSCalculator foi usado como SUT durante a fase de treinamento dos sujeitos.

5.3 Instrumentos de Coleta do *Quasi*-experimento

Esta seção apresenta todos os instrumentos para coleta de dados utilizados durante o *quasi*-experimento. Estes artefatos foram utilizados para diversas decisões e análises no resultado final. A seguir são apresentados todos os instrumentos utilizados e em quais momentos foram utilizados, uma versão desses documentos foi adicionada nos anexos deste estudo.

[2] ReactJSCalculator: https://github.com/yuryalencar/ReactJSCalculator
[3] Shopping Cart Software: http://demo.cs-cart.com/, último acesso em 14/02/2021

- **Formulários de configuração:** Estes formulários foram preenchidos durante a configuração dos ambientes, e possuíam o objetivo de coletar o tempo gasto com a configuração de cada *framework*. Os participantes do grupo que usou o *Teasy Framework*, possuiram mais formulários devido à solução necessitar de mais configurações do que o *Robot Framework*.

- **Formulários de *feedback*:** Estes artefatos foram criados com base em um formulário de avaliação de usabilidade (BLACKWELL; GREEN, 2000) e foram preenchidos ao final da execução do *quasi*-experimento, e foi composto pelo termo de consentimento, perfil do sujeito, perguntas com o foco em analisar o *framework* e por fim um espaço para o participante apresentar um *feedback* livre sobre a solução.

- **Códigos fonte finais e tempo gasto:** Ao final da execução cada participante forneceu o código-fonte criado, juntamente com o tempo gasto em cada atividade, para viabilizar uma análise do esforço necessário para criar testes automatizados de acordo com o *framework* utilizado. Os códigos fonte estão públicos e podem ser acessados tanto para os sujeitos que usaram o *Teasy Framework*[4] quanto para os que utilizaram o *Robot Framework*[5]. Um ponto importante é que para respeitar a privacidade dos participantes não é possível saber quem foi o responsável pelo desenvolvimento dos códigos públicos.

5.4 Design Experimental

5.4.1 Contexto

Atualmente a necessidade de entrega cada vez mais por produtos de qualidade torna a etapa de verificação e validação necessária. Assim, a automação dos testes realizados pode reduzir o custo e o esforço necessário replicar determinada verificação. Entretanto, esta automação não é uma atividade trivial, existem diversas tecnologias, padrões de projeto e ferramentas de suporte que muitas vezes não possuem uma sintaxe simples e representativa ao testador. Devido a representatividade, facilidade, agilidade e redução de tempo e esforço serem premissas do *Teasy Framework* esta avaliação de *quasi*-experimento foi idealizada, comparando esta solução com outro *framework* de testes tradicional, considerado também de fácil aprendizado (BISHT, 2013).

5.4.2 Objetivos

A avaliação realizada tem como objetivo geral avaliar a performance do *framework* Teasy quando comparado a um *framework* tradicional utilizado pela indústria. A partir do objetivo geral derivou-se os seguintes objetivos específicos:

[4] Código fonte (*Teasy Framework*): https://github.com/yuryalencar/TeasyExperiment
[5] Código fonte (*Robot Framework*): https://github.com/yuryalencar/RobotExperiment

- Identificar o tempo necessário para configuração, mesmo que na versão de desenvolvimento.
- Identificar gargalos e pontos de melhorias de usabilidade e configuração.
- Identificar benefícios, limitações e sugestões de melhoria para o *framework*.

5.4.3 Questões

As questões de pesquisa que esta avaliação tende a responder são:

- **QP01:** É possível implementar testes funcionais para aplicações *web* utilizando o *framework* ?
- **QP02:** A instalação do *framework* é fácil mesmo para usuários iniciantes ?
- **QP03:** O *framework* reduz o esforço necessário para a criação de *scripts* de teste ?
- **QP04:** O *framework* possibilita uma maior cobertura do SUT ?
- **QP05:** O *framework* possui uma boa usabilidade e representatividade com o domínio ?

5.4.4 Métricas

A fim de responder as questões de pesquisa apresentadas em 5.4.3 foram criadas as Métricas (Ms) abaixo.

- **M01:** Analisar o código final dos participantes e verificar a execução dos casos de teste, respondendo a questão **QP01**.
- **M02:** Tempo de instalação em minutos, sendo considerado de 0 a 15 minutos uma instalação rápida, de 16 a 30 uma instalação razoável e acima de 31 minutos uma instalação lenta, respondendo a questão **QP02**.
- **M03:** Tempo necessário para criação de casos de teste em minutos em comparação com os tempos necessários para a criação dos casos de teste usando o *Robot Framework*, diferenças positivas a favor do *Teasy framework* acima de 19% serão consideradas significativas, entre 10 e 18% consideradas parciais e diferenças iguais ou inferiores a 9% não serão consideradas, respondendo a questão **QP03**.
- **M04:** Quantidade de casos de testes e diversidade de cenários, respondendo a questão **QP04**.
- **M05:** Análise das respostas relacionadas a representatividade e usabilidade, respondendo a questão **QP05**.

5.4.5 Ameaças à Validade

Com o objetivo de possuir um estudo mais claro e consistente foram analisados quais poderiam ser as possíveis ameaças à validade do estudo. Nesta subseção além das ameaças são salientadas as medidas tomadas para que as mesmas sejam mitigadas. A seguir são apresentadas as ameaças à validade encontradas e classificadas.

- **Ameaça à Conclusão**:
  - **Pequeno número de sujeitos:** Era ciente que o número de sujeitos poderia impactar na "precisão" dos resultados do quasi-experimento, mas o esforço na configuração do quasi-experimento e os resultados alcançados mostraram um *feedback* importante relacionado ao uso do *framework*.
  - **Confiabilidade nas métricas:** Através dessa perspectiva, é sugerido que as medidas objetivas são mais confiáveis do que as medidas subjetivas, ou seja, não dependem do somente de um julgamento humano. Nesse estudo métricas subjetivas foram evitadas.
  - **Implementação correta utilizando o *framework*:** Por mais que o quasi-experimento tenha utilizado o mesmo aplicativo SUT para que os testes sejam implementados, existe o risco de que a implementação não seja semelhante entre os diferentes sujeitos. Este risco não pode ser totalmente evitado, pois as linguagens possuem diversas formas e caminhos possíveis que podem acarretar maior esforço ou não. A fim de mitigar este problema foi disponibilizado um documento com a especificação detalhada dos casos de testes[6] que deveriam ser implementados.
  - **Heterogeneidade do conhecimento entre os sujeitos:** Os participantes podem possuir diferenças de conhecimentos, experiências e grau acadêmico. Sabendo disso, foram realizados bloqueios de acordo com nível de experiência com testes de *software* (iniciante ou avançado) entre os participantes.
- **Ameaça Interna**:
  - **Maturação:** Cada sessão do quasi-experimento foi aplicada pela manhã, pois os sujeitos estão mais motivados e menos cansados pela carga de trabalho do dia.
  - **Seleção:** Uma pesquisa foi aplicada para avaliar o conhecimento e a experiência dos assuntos e, em seguida, usada para selecionar e agrupar (bloquear) os sujeitos com base nos seus conhecimentos.

[6] Especificação de testes utilizada: https://url.gratis/giwi3, último acesso em 24/02/2021

  - **Aprendizado:** A data das execuções do quasi-experimento foram definidas anteriormente entre os grupos, que não foram avisados quais seriam as atividades e nem os participantes. Desse modo foi evitado a aprendizagem com o passar das execuções. A fim de mitigar mais esta ameaça todo o treinamento entre os participantes foi gravado e adicionado em *playlists* uma com o foco no *Robot Framework*[7] e outra no *Teasy Framework*[8].

- **Ameaça Externa**:

  - **Conhecimentos:** Uma ameaça à validade externa do quasi-experimento era selecionar um grupo de conhecimentos que podem não ser representativos para a comunidade de teste de funcional de *software*. Assim, para mitigar essa ameaça, foi selecionado alunos com algumas habilidades, como conhecimentos básicos para implementar testes funcionais e profissionais da área de desenvolvimento e testes.

  - **Tarefas:** Outra ameaça à validade do quasi-experimento é que as tarefas definidas para implementar os casos de teste podem não refletir as atividades realizadas por um testador. A fim de mitigar essa ameaça, foi entrevistado testadores funcionais e engenheiros de qualidade, de diferentes empresas, para definir quais atividades deveriam ser realizadas para representar o cenário real. Os consultores de teste não participaram e não tiveram nenhum contato com os participantes do quasi-experimento.

- **Ameaça a Construção**:

  - **Entendimento do propósito:** Por fim uma possível ameaça é o fato de que alguns sujeitos podem concluir erroneamente que seu desempenho pessoal é medido, por exemplo, classificar os sujeitos do quasi-experimento. A fim de mitigar essa ameaça, antes de cada sessão, foi explicado e relembrado que o *framework* que estava sendo medido.

5.5 Execução do *Quasi*-experimento

5.5.1 Preparação

Nesta subseção, é apresentado como o *quasi*-experimento foi conduzido, como a documentação foi preparada e como o ambiente do *quasi*-experimento foi configurado. Também é descrito como os sujeitos do *quasi*-experimento foram envolvidos e motivados.

Devido à pandemia não ocorreu contato pessoalmente com os participantes. Entretanto, cada participante foi convidado de maneira unitária para uma vídeo chamada,

[7] Treinamento de *Robot Framework*: https://url.gratis/g9nWt, último acesso em 24/02/2021
[8] Treinamento do *Teasy Framework*: https://url.gratis/q8oUP, último acesso em 24/02/2021

para participar de uma entrevista sobre seu perfil e melhor para a realização do *quasi*-experimento para que o mesmo não coincidisse com alguma atividade importante do participante. As informações coletadas foram utilizadas para realizar os bloqueios com base em suas experiências em testes funcionais e definir datas para a execução do *quasi*-experimento.

Uma precaução na fase de preparação, foi a criação de um treinamento em vídeo para cada *framework*, desse modo foi possível nivelar o contato de cada participante com cada uma das tecnologias. Os treinamentos eram compostos pela configuração do ambiente e sobre como utilizar ambos *frameworks*. O resultado final dos treinamentos eram códigos equivalentes entre ambas tecnologias, usando os mesmos padrões e estruturas de testes, além do mesmo SUT.

5.5.2 Execução

A execução do *quasi*-experimento ocorreu em janeiro de 2021 e foi composta por duas etapas sendo estas o treinamento e a execução supervisionada do *quasi*-experimento. A etapa de treinamento foi dividida em duas *playlists*, sendo uma para cada *framework*, estes artefatos foram usados para o treinamento dos sujeitos envolvidos, e abordaram sobre como utilizar cada uma das tecnologias, levando em consideração padrões de projeto, reúso e manutenibilidade. O SUT e exemplos de testes utilizados durante o treinamento são os mesmos apresentados na seção 4.6, desse modo ambos grupos obtiveram o mesmo nível de informação necessário em ambos os *frameworks*. A Tabela 9 representa a distribuição de sujeitos entre os grupos e os níveis de experiência com testes de *software* entre os usuários em cada grupo. Um ponto importante é que cada sujeito foi escolhido aleatoriamente para seu respectivo grupo.

Tabela 9 – Sujeitos por grupo

Framework	**Avançados**	**Intermediários**	**Iniciantes**
Teasy Framework	3	1	2
Robot Framework	3	1	2
Total de participantes	**6**	**2**	**4**

Devido a pandemia, em nenhum momento do *quasi*-experimento houve contato presencial com os participantes, desse modo para cada grupo de participantes o treinamento foi disponibilizado dois dias antes da execução supervisionada, a quantidade de visualizações e quem as visualizou foi monitorado com o intuito de garantir que os participantes tenham visualizado todo o treinamento e maneira uniforme. No dia marcado os participantes entraram em uma chamada de vídeo para a execução supervisionada do *quasi*-experimento, neste momento foi disponibilizado um novo vídeo apresentando quais eram as tarefas que deveriam realizar, desta vez utilizando a aplicação *Shopping Cart Software*. As tarefas também estavam descritas em um documento compartilhado entre

todos, quaisquer dúvidas relacionadas a como detectar os elementos presentes na página foram permitidas, pois, o intuito era a avaliação dos *frameworks*, desse modo foi possível mitigar problemas externos que poderiam impactar no tempo de cada participante. A fim de medir o esforço, cada participante cronometrou o tempo gasto em cada uma das atividades implementadas, assim permitindo uma melhor análise de esforço ao final do estudo. A seguir é explicado um resumo de cada atividade.

- **Atividade 1:** Pesquisar por um produto específico dentro da aplicação e verificar sua existência.
- **Atividade 2:** Simular um registro na aplicação, adicionar um produto no carrinho e continuar as compras.
- **Atividade 3:** Simular um registro na aplicação, adicionar um produto no carrinho e finalizar o *checkout*.
- **Atividade 4:** Criar uma sequência utilizando o *Teasy Generator* e verificar a quantidade de testes gerados. Esta atividade só foi requisitada aos participantes que utilizaram o *Teasy Framework* devido a compatibilidade com este framework.

Por fim, cada participante respondeu um formulário sobre o uso do respectivo *framework*. Este formulário era composto por perguntas relacionadas ao uso e funcionalidades, além de *feedbacks* e sugestões de melhoria. Na subseção a seguir são apresentados os resultados comparativos entre os *frameworks*.

5.5.3 Resultados

Nessa subseção são apresentados todos os resultados encontrados, conjuntamente com a resposta das questões de pesquisa através das métricas estabelecidas anteriormente neste capítulo.

Tabela 10 – Tempo médio por grupo de participantes

Tempo médio em minutos					
Framework	**Bloqueios**	**Tempo médio por atividade**			
		Atividade 1	**Atividade 2**	**Atividade 3**	**Média Geral**
Robot Framework	Avançados	35	62.33	65	**54.11**
	Intermediários	147	N/A	N/A	**147**
	Iniciantes	129.5	N/A	N/A	**129.5**
Teasy Framework	Avançados	27.66	78.33	47	**50.96**
	Intermediários	20	60	60	**46.66**
	Iniciantes	38.5	118	11	**55.83**

A Tabela 10 apresenta o tempo médio por grupo de participantes, esta foi a primeira análise realizada para responder a **QP03**, que diz respeito ao *Teasy Framework* ser capaz de reduzir o esforço, a métrica designinada foi o tempo de que cada participante

precisaria para concluir as atividades. Entretanto realizando uma análise dos tempos relatados e do código entregue pelos participantes é possível afirmar que nenhum participante de nível intermediário ou iniciante do grupo que utilizou o *Robot framework* conseguiu entregar as atividades 2 e 3, impactando diretamente na média. Desse modo o único tempo que poderia ser comparado entre os *frameworks* foi o tempo necessário para a atividade 1. A fim de comparar a média entre os tempos foi criada a Tabela 11 que apresenta a média independente do grupo, as médias das atividades 2 e 3 dos participantes que utilizaram o *Robot framework* foram muito superiores, entretanto foram comprometidas pelo fato dos participantes que não conseguiram concluir a tarefa. A média da atividade 3 do grupo de sujeitos que usaram o *Teasy framework* também foi comprometida devido a um dos participantes de nível iniciante não conseguir finalizar a tarefa. Mesmo com estas ressalvas, através da Tabela 10 é possível analisar que independente do nível do grupo o tempo médio de uso do *Teasy framework* foi similar, o que pode apresentar uma menor curva de aprendizagem no uso da tecnologia, auxiliando na resposta da **QP02**, sobre o *framework* possuir fácil uso mesmo para usuários iniciantes.

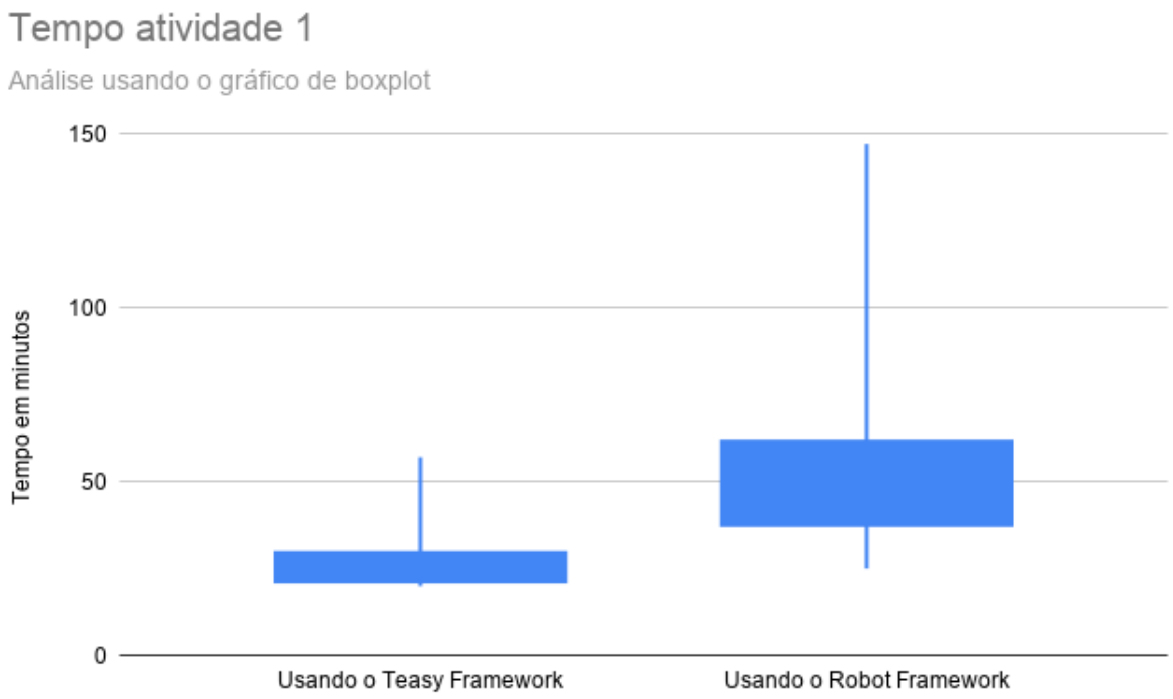
Tabela 11 – Média geral de tempo em minutos

	Tempo médio geral por atividade			
Framework	**Atividade 1**	**Atividade 2**	**Atividade 3**	**Média Geral**
Robot Framework	43.16	20.77	21.66	**28.53**
Teasy Framework	28.72	85.44	39.33	**153.49**

Com base na média do tempo necessário pelos participantes para concluir a atividade 1 e o gráfico apresentado na Figura 17, é possível responder a **QP03** como positiva, o *Teasy Framework* pode reduzir o esforço necessário para a criação de *scripts* de teste. Na figura 17 é possível verificar que mesmo o participante com o melhor desempenho com o *Robot Framework* dentro do conjunto do *Teasy Framework* se manteve na média e não superior. Esta resposta também foi confirmada pelo formulário preenchido pelos participantes ao final do *quasi*-experimento, onde através da escala *likert* informaram nota 4 de 5, afirmando que a tecnologia pode reduzir significativamente o tempo e esforço para a criação de *scripts* de teste. Além disso, os parâmetros utilizados onde somente acima de 19% de diferença seria considerado um resultado significativo foi encontrado, onde a diferença entre os tempos médios foi de 40.16% a favor do *Teasy Framework*.

Outra resposta positiva a uma questão de pesquisa chave sobre o *Teasy framework* foi em relação a **QP01**, onde é possível afirmar que a tecnologia permite a possibilidade de implementação de testes funcionais para aplicações *web*. Esta afirmação é resultado da análise do código final dos participantes onde 83.33% dos participantes conseguiram finalizar todas as atividades em comum com o *Robot Framework* e 100% dos participantes conseguiram implementar ao menos 1 atividade. Enquanto, os participantes que utilizaram o *Robot framework* apenas 50% conseguiram finalizar todas as atividades e nem todos

Figura 17 – Tempo necessário para a atividade 1



Fonte: Criada pelo Autor

os participantes conseguiram finalizar uma atividade. Um ponto importante nesta análise é que não apresenta pontos negativos no *Robot framework*, os participantes poderiam possuir outro resultado caso não necessitassem o uso de padrões de projeto ou reusabilidade de componentes, desse modo é possível comparar a entrega a nível de qualidade de código de teste e estrutura escalável. Desse modo, é possível apresentar que mesmo um testador iniciante consegue entregar um projeto escalável com reúso e com código de qualidade utilizando o *Teasy framework*.

Entretanto o *Teasy Framework* não apresentou resultados superiores em relação a **QP02**, sua instalação de acordo com o *feedback* e tempo informado pelos participantes foi extensa e por alguns casos considerada complexa. Esse resultado poderia acontecer devido aos participantes necessitarem configurar todo o *framework* em versão de desenvolvimento, mas mesmo com 83.3% dos participantes apresentando uma instalação rápida ou razoável para a configuração do *Teasy Language*, este tempo não se concretizou para as demais partes do *framework*, onde 66.7% tiveram uma instalação razoável para lenta durante a configuração tanto da *Teasy Structure* quanto do *Teasy Generator*. Alguns pontos apresentados pelos participantes podem ser levados em consideração a favor do *framework*, como parte dos sujeitos tiveram problemas com o acesso *internet* durante a configuração e

a grande maioria não teve problemas ao instalar somente as soluções propostas pelo *Teasy Framework*. O tempo que acarretou lentidão na configuração foi devido às tecnologias necessárias pré-configuração das soluções propostas. Desse modo, é possível reduzir o tempo e configuração em uma nova versão do *framework*. O tempo de configuração para o *Robot Framework* também foi considerado de razoável a lento por 66.73% dos participantes, mas por se necessitar somente de uma configuração, e também ser utilizado pelo *Teasy Framework* a solução apresentou uma quantidade de tempo inferior, portanto melhor do que comparado com o *framework* proposto neste estudo.

A **QP04**, sobre o *Teasy Framework* viabilizar uma maior cobertura também se mostrou verdadeira, entretanto somente em casos onde o *Teasy Generator* foi utilizado, isso acontece devido à solução utilizar os caminhos informados para gerar várias sequências com base no algoritmo de busca em profundidade e nas ações definidas nos arquivos de *Pages*. Um ponto importante é que os testes gerados pelo *Teasy Generator* precisam ser analisados posteriormente com o auxílio do relatório, pois, devido as ramificações é possível que em algumas sequências existam verificações antes do momento correto, a não ser que o testador insira a verificação em conjunto com a ação, ou seja, não cria uma ação nova no arquivo de *Pages* somente para essa finalidade. A quantidade de casos de testes utilizando o *Teasy Generator* foi muito superior em comparação com os usuários que somente criaram suas próprias sequências, mas tem muitas variantes que podem impactar na quantidade de testes gerados como quantas ações foram implementadas e como as sequências foram especificadas no gerador *web*. A resposta a esta pergunta também foi confirmada pelos participantes, onde 66.7% afirmaram que o *Teasy Generator* pode sim detectar problemas não mapeados anteriormente pelos testadores, aumentando assim a cobertura de testes. Essa superioridade na cobertura era esperada tendo em vista que o *Robot Framework* não gera sequências de teste, somente permite a realização de implementações.

A **QP05** é a questão relacionada à usabilidade e representatividade com o domínio de testes funcionais. Sabendo disso, a usabilidade se mostrou eficaz, onde 66.7% dos participantes responderam não possuir nenhum problema com o ambiente de desenvolvimento da *Teasy Language* e que a sintaxe da linguagem é clara, e que cada arquivo tem o seu papel bem definido, dificultando o usuário cometer erros facilmente. Um ponto importante também analisado, foi a manutenibilidade, 83.3% dos sujeitos responderam que a atualização automática das dependências e o código final de execução, possuem uma fácil leitura e compreensão, e o reúso viabiliza uma fácil manutenção dos testes criados. Um ponto interessante, dessa vez relacionado à *Teasy Structure*, foi que para 66.7% dos sujeitos que fizeram o teste com o *Teasy Framework*, a estrutura não poderia ser utilizada independente da tecnologia, enquanto para 83.4% dos participantes que utilizaram o *Robot Framework*, a mesma estrutura poderia ser utilizada de maneira isolada. Desse modo, podemos analisar que a estrutura tem potencial para ser utilizada independente

da tecnologia empregada na implementação dos testes funcionais. Enquanto isso, o *Teasy Generator* também teve uma avaliação satistfatória, 66.7% dos participantes mencionaram que é simples gerar novas sequências com sua ajuda, e o mesmo percentual citou que todas ou a maioria das sequências geradas eram representativas para o SUT. Por fim, o relatório gerado pelo *Teasy Framework* ao final dos testes, não pode ser comparado com o relatório do *Robot Framework* por serem idênticos, mas em ambos grupos, todos os participantes acreditam que o artefato pode ser usado para a análise de resultados das execuções, e até mesmo como passo a passo para replicação dos erros encontrados contendo evidências concretas.

A partir do formulário também foi possível salientar a representatividade com o domínio, onde 66.7% usaria o *Teasy Framework* novamente devido a praticidade, qualidade do código gerado, reúso e manutenibilidade, 16.7% responderam que não utilizariam por conhecimento de outras tecnologias e possuirem uma maior familiaridade, o restante dos participantes não justificou o por que não gostariam de usar a solução futuramente. Mesmo com o resultado satisfatório na **QP05**, também foi realizado um *Survey* com especialistas de usabilidade, neste contexto para avaliar somente a *Teasy Language*, o planejamento, execução e resultados podem ser visualizados no capítulo 6. Por fim, o treinamento fornecido também foi avaliado como positivo pelos participantes, e que foi muito útil tanto na configuração quanto no uso do *framework*, mas em contrapartida foi cansativo devido ao tempo fornecido aos sujeitos.

5.6 Lições do Capítulo

Tendo em vista os resultados apresentados pelo quasi-experimento, neste capítulo é possível concluir que o *Teasy Framework* atendeu aos objetivos especificados, fornecendo benefícios significativos, tanto na manutenção e reúso de componentes, quanto na redução de esforço e tempo necessário para a criação e geração de cenários de testes executáveis. Além disso, o relatório gerado ao final das execuções, pode auxiliar a análise dos resultados e na replicação de *bugs* encontrados. Na Tabela 12 é possível visualizar todos os pontos positivos e negativos do *framework* com base nesse capítulo e no *feedback* reportado através dos formulários de avaliação ao final do quasi-experimento.

Tabela 12 – Benefícios e limitações do *Teasy Framework*

Benefícios	**Limitações**
Reúso de componentes	Instalação complicada
Redução de esforço	Falta de documentação textual
Sintaxe clara e bem definida	Configuração da *Teasy Structure* é manual
Relatório completo	
Não necessita conhecimento prévio em programação	
Integração com um ambiente de desenvolvimento	
Geração de sequências de teste representativas	

6 TEASY LANGUAGE: UM *SURVEY* PARA AVALIAÇÃO HEURÍSTICA

Com o intuito de avaliar a *Teasy Language* em relação a sua usabilidade, foi utilizado o Usa-DSL *Process* (POLTRONIERI et al., 2018) como apoio. Para este estudo foi realizado uma Avaliação Heurística através do método empírico *Survey*. Mesmo que a *Teasy Language* seja uma DSL projecional, a mesma faz o uso somente de textos, desse modo foi considerada que uma avaliação de inspeção ajudaria a compreender e encontrar determinados tipos de problemas na interface apresentada para os testadores. Desta forma, para que a avaliação fosse executada utilizou-se como base o *Checklist* Heurístico para avaliar DSLs textuais (POLTRONIERI et al., 2021), esse *checklist* analisa a DSL de acordo com as dez heurísticas propostas por (NIELSEN; MOLICH, 1990). Posto isso, a motivação deste estudo foi de analisar a facilidade no uso da *Teasy Language*, tendo em vista que o objetivo da linguagem projetada é reduzir o esforço e tempo necessário para a criação de testes automatizados, a sua usabilidade pode impactar diretamente no seu uso pelos testadores. Assim, esse estudo tem o foco de responder as seguintes dúvidas: A *Teasy Language* é fácil de se usar? seus elementos, *highlights* e *feedbacks* são representativos? Qual a severidade dos problemas de usabilidade existentes ? Caso existam problemas de usabilidade, é possível utilizar a DSL, ou é necessária uma atualização urgente?

6.1 Definição do Estudo

A estratégia adotada para responder essas questões, foi a realização de um *Survey*, no qual contemplando todas as heurísticas de usabilidade propostas no Usa-DSL *Checklist* Heurístico para DSL Textual e tem por objetivo considerar as peculiaridades e diferentes representatividades, presente no domínio de DSLs. Em resumo, o objetivo deste estudo é coletar evidências sobre a usabilidade da *Teasy Langauge* para aplicar testes funcionais *web*. O objetivo da avaliação é detectar possíveis problemas de usabilidade na linguagem Teasy. A principal motivação é mitigar os problemas encontrados em uma versão futura, aumentando a sua possibilidade de adoção pela indústria.

6.1.1 Objetivos

A avaliação realizada tem como objetivo geral avaliar a usabilidade da linguagem Teasy com base em um *Checklist* Heurístico para DSL textual. A partir do objetivo geral derivou-se os seguintes objetivos específicos:

- Detectar problemas de usabilidade na *Teasy Language*.
- Categorizar os erros de acordo com o seu respectivo grau de severidade.
- Inferir sobre o entendimento dos participantes ao utilizarem a *Teasy Language*.

- Selecionar os problemas de usabilidade a fim de serem corrigidos em uma próxima versão.

6.1.2 Ameaças à Validade

Com o objetivo de possuir um estudo mais claro e consistente foram analisados quais poderiam ser as possíveis ameaças à validade do estudo. Nesta subseção além das ameaças são salientadas as medidas tomadas para que as mesmas sejam mitigadas. A seguir são apresentadas as ameaças à validade encontradas e classificadas.

- **Ameaça a conclusão**:
  - **Pequeno número de sujeitos:** Era ciente que o número de sujeitos poderia impactar na quantidade de problemas de usabilidade encontrados, mas o esforço na preparação e tarefas do *survey*, apresentou resultados importantes relacionado ao uso do da linguagem.
  - **Implementação correta utilizando o *framework*:** Por mais que o *survey* tenha utilizado o mesmo aplicativo SUT para que os testes de usabilidade sejam implementados, existe o risco de que a implementação não seja semelhante entre os diferentes sujeitos. Este risco não pode ser totalmente evitado, pois as linguagens possuem diversas formas e caminhos possíveis que podem acarretar maior esforço ou não. A fim de mitigar este problema foi disponibilizado um documento com a especificação detalhada dos casos de testes[1] que deveriam ser implementados.
  - **Heterogeneidade do conhecimento entre os sujeitos:** Dentro deste grupo de especialistas, apenas um participante nunca havia utilizado uma DSL anteriormente, o que torna a avaliação deste usuário somente com base no domínio, sem comparações com outras DSLs existentes. Neste caso, não foi possível mitigar este problema.
- **Ameaça interna**:
  - **Maturação:** Como o *Survey* foi enviado para os participantes é possível que os mesmos escolham horários que estão cansados para realizá-lo, diminuindo a sua concentração. A fim de mitigar este problema, o *Survey* ficou disponível para este especialista durante uma semana para que o mesmo escolhesse o melhor horário para realizá-lo.
- **Ameaça externa**:

[1] Especificação de testes utilizada: https://url.gratis/giwi3, último acesso em 24/02/2021

  - **Conhecimentos:** Uma ameaça à validade externa do experimento era selecionar um grupo de conhecimentos que podem não ser representativos para a comunidade de teste de funcional de *software*. Assim, para mitigar essa ameaça, foram selecionados especialistas com algumas habilidades, como conhecimentos básicos para implementar testes funcionais e profissionais da área de desenvolvimento e testes.
  - **Tarefas:** Outra ameaça à validade do *Survey* é que as tarefas definidas para implementar os casos de teste podem não refletir as atividades realizadas por um testador. A fim de mitigar essa ameaça, foi entrevistado testadores funcionais e engenheiros de qualidade, de diferentes empresas, para definir quais atividades deveriam ser realizadas para representar o cenário real.

- **Ameaça a construção**:
  - **Entendimento do propósito:** Por fim uma possível ameaça é o fato de que alguns sujeitos podem concluir erroneamente que seu desempenho pessoal é medido, por exemplo, classificar os sujeitos participantes do *Survey*. A fim de mitigar essa ameaça, todos os participantes foram avisados por e-mail do propósito do estudo.

6.2 Instrumentos de Coleta de Dados

Esta seção apresenta todos os instrumentos para coleta de dados utilizados durante o *Survey*. Estes artefatos foram utilizados para diversas decisões e análises no resultado final. A seguir são apresentados todos os instrumentos utilizados:

- Treinamento no formato de vídeo sobre uso do *Teasy Framework*[2].
- Especificação de testes com as atividades a serem realizadas[3].
- Questionário completo[4] composto por:
  - Termo de Consentimento.
  - Questionário para levantar o perfil do participante.
  - Usa-DSL *Checklist* Heurístico para DSLs textuais.

6.3 Execução do Estudo

Nesta seção, é apresentada a condução do *Survey*, como a documentação foi preparada e como o ambiente utilizado foi configurado, também é descrito o perfil dos sujeitos,

[2] Treinamento Teasy Framework: https://youtube.com/playlist?list=PLrua1dSU7PC0fHVkqZqZWFefDZC57t7wy
[3] Especificação de testes utilizada: https://url.gratis/giwi3, último acesso em 24/02/2021
[4] Questionário completo: https://forms.gle/4fxJeCb7BkvJfaBQA, último acesso em 05/03/2021

coleta de dados e os resultados. Como no *quasi*-experimento, devido à pandemia não ocorreu contato pessoalmente com os participantes. Entretanto, cada participante foi contatado via e-mail com a mesma descrição do convite, objetivos e quais atividades realizar. Como o objetivo do *Survey* é não impactar nas atividades dos participantes, cada um teve uma semana para concluir todas as atividades e responder o questionário completo.

6.3.1 Perfil dos Sujeitos

Para este *Surveys* foram convidados quatro (4) participantes, através de e-mail. Esta amostra foi selecionada em busca de profissionais experientes em Interação Humano Computador (IHC). Além disso, os participantes também foram convidados por recomendação de outros participantes considerados especialistas. Após o aceite, os documentos para a entrevista foram enviados a eles. O Questionário de Perfil foi utilizado para identificar a experiência do participante e outras informações relevantes. Para obter informações sobre a experiência dos participantes, foram feitas as seguintes perguntas: 01) Você já utilizou uma *Domain-Specific Language* (DSL)?, 02) Você já criou alguma *Domain-Specific Language* (DSL)?, 03) Quais estudos empíricos já participou?, 04) Qual categoria(s) você pertence atualmente(academia ou indústria?, 05) Qual é o seu grau de escolaridade?, 05) Quanto tempo possui de experiência na Indústria?, 06) Quanto tempo possui de experiência com testes de software? e 07) Você já aplicou alguma avaliação heurística?. Neste estudo as respostas do questionário de perfil, bem como as demais respostas do estudo serão identificadas pelo rótulo atribuído a cada participante, ou seja, **E1**, **E2**, **E3**, e **E4**, com a finalidade de preservar a identidade dos participantes, a seguir será descrito o perfil levantado de cada um dos participantes.

- **E1**: O especialista já utilizou uma DSL anteriormente, também participou de estudos empíricos como estudos de caso, experimentos controlados e *Surveys*, atualmente é um discente de graduação e possui de 1 a 3 anos de experiência na indústria. O sujeito também possui um ano de experiência com testes funcionais de *software*, além de possuir experiência em avaliações heurísticas.

- **E2**: O especialista nunca utilizou uma DSL anteriormente, mas participou de estudos empíricos como estudos de caso, provas de conceito e *Surveys*, atualmente é um discente de graduação e possui de 1 a 3 anos de experiência na indústria. O sujeito não possui experiência com testes funcionais de *software*, mas possui experiência em avaliações heurísticas.

- **E3**: O especialista tanto já utilizou uma DSL quanto criou uma anteriormente, participou de estudos empíricos como experimentos controlados, estudos de caso, *quasi*-experimentos, provas de conceito e *Surveys*, atualmente é um discente de

graduação e possui de 1 a 3 anos de experiência na indústria. O sujeito também possui de 1 a 3 anos de experiência com testes funcionais de *software*, além de experiência em avaliações heurísticas.

- **E4**: O especialista já utilizou uma DSL anteriormente, também participou de estudos empíricos como estudos de caso, experimentos controlados, *quasi*-experimentos e *Surveys*, atualmente faz doutorado e não possui experiência na indústria. O sujeito possui até um ano de experiência com testes funcionais de *software*, além de possuir experiência em avaliações heurísticas.

Dentro deste grupo de especialistas, apenas um participante nunca havia utilizado uma DSLs anteriormente, o que torna a avaliação deste usuário somente com base no domínio, sem comparações com outras DSLs existentes. Entre os participantes também foi detectado um especialista que já implementou uma DSLs anteriormente, o que viabiliza uma análise tanto da implementação quanto da adequação com o domínio, podendo resultar em *feedbacks* técnicos para a resolução de um possível problema de usabilidade. Um ponto importante é que todos os sujeitos tem experiência como participante em estudos empíricos e todos também já aplicaram uma avaliação heurística. Entre os sujeitos apenas um participante possui a formação como Doutorado incompleto enquanto todos os demais são discentes de graduação, e dentre o grupo somente um sujeito não possui experiência na indústria. Por fim, em uma análise do conhecimento do domínio, apenas um sujeito apresentou não ter experiência em testes de *software*, dois apresentaram ter até um ano de experiência, enquanto o quarto participante tem entre um e três anos.

6.3.2 Preparação e Coleta de Dados

Um cuidado especial durante a etapa de preparação, foi o contato dos sujeitos com a *Teasy Language*, desse modo foi utilizado a parte do treinamento relacionado ao *Teasy Framework* que apresenta como usar a linguagem. Os vídeos do treinamento enviado aos participantes, eram compostos pela configuração e uso da linguagem Teasy. O resultado final dos treinamento foi um código executável utilizando a *Teasy Language*.

A execução do *Survey* ocorreu em janeiro de 2021 e foi composto por duas etapas sendo estas o treinamento na linguagem e o preenchimento do *Checklist* Heurístico. Os vídeos e exemplos utilizados para o treinamento estão presentes na *playlist* utilizada no *quasi*-experimento. E por fim a coleta de dados foi realizada através do Questionário completo, sendo este composto por treze páginas, que coletavam as seguintes informações: consentimento dos participantes, dados sobre o perfil, *feedbacks* gerais sobre a linguagem e dez páginas de avaliação de usabilidade, onde cada página possuía perguntas sobre uma heurística em específica.

6.3.3 Resultados

Nessa subseção são apresentados todos os resultados obtidos através do *Checklist* Heurístico utilizado para avaliar a *Teasy Language*. A partir das respostas dos participantes, foi realizada uma análise qualitativa em relação a cada um dos problemas encontrados e seu grau de severidade. A Tabela 13 apresenta a classificação dos erros a serem identificados pelos especialistas.

Tabela 13 – Classificação dos erros de usabilidade

Severidade	Tipo	Descrição
0	Não aplicável	Não concordo que seja um problema de usabilidade
1	Problema cosmético apenas	Não precisa ser corrigido a menos que haja tempo extra disponível no projeto
2	Pequeno o problema de usabilidade	Corrigir isso deve ter baixa prioridade
3	Grande problema de usabilidade	Importante corrigir, por isso deve ter uma alta prioridade
4	Catástrofe de usabilidade	Imperativo, corrigir antes que o produto seja lançado

A seguir serão apresentadas as analises realizadas através do *Heuristic Evaluation Checklist for DSL Textual* e das classificações presentes na Tabela 13 para cada heurística:

- **H1: Visibilidade do status do sistema**
  - **Pontos positivos**: Nenhum especialista encontrou problemas relacionados a ambiguidade dos comandos existentes na linguagem, ou seja, cada comando realiza especificamente uma ação. Todos os usuários afirmaram que a linguagem foi acoplada da maneira correta, quando necessária uma atualização de alguma informação em uma classe, esta informação é refletida em todos os locais que a utilizam. Por fim, nenhum dos sujeito identificou problemas nas mensagens de alerta da *Teasy Language*, sendo assim pode-se verificar que a linguagem fornece as mensagens na tela por tempo suficiente para serem lidas.
  - **Pontos negativos**: O especialista **E4** detectou que a linguagem não informa imediatamente quando o arquivo é salvo, este mesmo sujeito classificou esse erro como 1, um problema cosmético de usabilidade.
- **H2: Compatibilidade entre a DSL e o mundo real**
  - **Pontos positivos**: Todos os especialistas concordaram que a *Teasy Language* possui elementos que atendem o domínio de testes de *software* funcional e que todos estes elementos possuem uma boa representação.
  - **Pontos negativos**: O especialista **E4** mencionou que de acordo com sua experiência uma palavra-chave em específico, pode não estar coerente com o domínio. Entretanto esse problema foi classificado como 1, ou seja, erro cosmético que não afeta o uso da *Teasy Language*.
- **H3: Controle e liberdade para o usuário**

  - **Pontos positivos**: Todos os especialistas afirmaram que a linguagem possui redundância nos comandos de desfazer e refazer, permitindo uma liberdade ao usuário para restaurar o estado do código em qualquer momento durante o desenvolvimento.
  - **Pontos negativos**: Não houveram pontos negativos.

- **H4: Consistência e Padronização**
  - **Pontos positivos**: Todos os elementos textuais presentes na linguagem estão conectados uns aos outros de acordo com seus nomes e utilizando o mesmo estilo. Sobre os atalhos presentes na linguagem, um dos especialistas apresentou que em poucos momentos é permitido o texto livre, e que não considera isso um erro de usabilidade, esse ponto era esperado por a linguagem ser projecional, a maior parte do código é gerada para o testador.
  - **Pontos negativos**: Entre a conexão dos componentes o especialista **E4** encontrou inconsistências no *highlight*, esse erro foi classificado como 2 sendo assim, um pequeno erro de usabilidade e possui prioridade de correção baixa. Outro erro detectado pelo mesmo sujeito é que as vezes o mecanismo de destaque de erros demora aparecer, também classificado como um pequeno erro de usabilidade.

- **H5: Prevenção de erros**
  - **Pontos positivos**: Todos os especialistas confirmaram que a linguagem prevê decisões erradas pelos usuários e os avisa. Com ralação a requesição de alguma confirmação, quando um usuário realiza ações consideradas perigosas, os especialistas não conseguiram verificar devido aos cenários utilizados que não necessitaram confirmações.
  - **Pontos negativos**: O especialista **E4** apresentou problemas ao tentar apagar um erro de digitação utilizando a tecla "*DELETE*", onde ao invés de apagar somente um *caractere*, apagou um componente, o especialista classificou esse erro como 1, ou seja, apenas um problema cosmético.

- **H6: Reconhecimento em vez de memorização**
  - **Pontos positivos**: Todos os participantes confirmaram que a *Teasy Language* possui palavras-chave fáceis de lembrar e significativas para o domínio de teste de *software*. Além disso os especialistas também não encontraram problemas de usabilidade nos *highlights*, ou seja, os destaques aumentaram a legibilidade do código criado através da linguagem.
  - **Pontos negativos**: Não houveram pontos negativos.

- **H7: Eficiência e flexibilidade de uso**
  - **Pontos positivos**: Todos os especialistas confirmaram que a linguagem oferece todos os componentes desejados para completar as tarefas desejadas.
  - **Pontos negativos**: Devido a linguagem Teasy ser projecional, a mesma não possui atalhos para usuários experientes, todos os usuários recebem a mesma projeção, mesmo assim o especialista **E3** classificou essa ausência como problema de usabilidade, entretanto a classificou como um pequeno erro e que não afeta a eficiência no uso. O sujeito **E4** detectou que o ambiente de desenvolvimento possui algumas limitações como: não ser possível copiar um elemento através do *mouse*, sendo necessário o uso do teclado e também não é possível apagar um componente usando a tecla de *backspace*, isso quando o foco estava no início de algum *placeholder*, ambos os erros foram categorizados como 3, ou seja, erros graves e que devem ser priorizados em uma futura versão.
- **H8: Estética e design minimalista**
  - **Pontos positivos**: Todas os especialistas concordam que as informações necessárias para o entendimento de um componente ou informação são apresentadas em textos curtos, não possuindo mais do que três linhas.
  - **Pontos negativos**: Em alguns componentes, o especialista **E4** detectou alguns *highlights* de informações adicionais distintas com cores fora do padrão, considerado como um erro pequeno, de baixa prioridade.
- **H9: Ajudar os usuários a reconhecerem, diagnosticarem e recuperarem-se de erros**
  - **Pontos positivos**: Todos os especialistas responderam que em caso de erro, a DSL mostra uma mensagem de erro apropriada, onde após a leitura o usuário sabe como se recuperar do erro ou onde procurar uma solução. Em casos de problemas, as mensagens apresentam os possíveis problemas de forma clara e objetiva. Além disso, a linguagem Teasy possui um mecanismo que destaca erros de sintaxe no texto ou palavras reservadas, a *Teasy Language* também possui mecanismos para destacar erros de aninhamento ou organização de palavras reservadas que podem tornar o modelo inconsistente.
  - **Pontos negativos**: O especialista **E3** não conseguiu detectar na linguagem Teasy, mecanismos que marcam erros de conexão entre elementos que podem tornar o modelo inconsistente, este erro foi classificado como cosmético apenas.
- **H10: Ajuda e documentação**
  - **Pontos positivos**: Todos os especialistas confirmaram que a linguagem Teasy tem ajuda contextual para cada elemento.

– **Pontos negativos**: Os especialistas **E4** e **E3** detectaram os mesmos problemas relacionados ao tutorial , ambos participantes apresentaram que não existe tutorial na forma textual, e classificaram este problema como 1, ou seja, de baixa prioridade. Os especialistas **E4** e **E3** encontraram problemas também nos suportes fornecidos pelos comandos, ambos alegaram que os textos de ajuda são muito pequenos e possuem alguns erros de gramática, classificando-o como 2, um pequeno problema de usabilidade.

Em resumo os especialistas detectaram alguns problemas de usabilidade na *Teasy Language*, entretanto eles não impactaram diretamente no funcionamento e entendimento da linguagem. A Tabela 14 apresenta o resumo de todos os erros categorizados sendo eles 6 cosméticos, 3 pequenos e apenas 2 erros considerados importantes, além de nenhum problema de usabilidade catastrófico. Desse modo, se tem indícios que mesmo na primeira versão da *Teasy Language* os participantes entenderam o objetivo da linguagem, conseguindo fazer o seu uso sem muitos problemas de usabilidade. Por fim, todos os erros referentes ao editor e a linguagem serão resolvidos em uma futura versão, desse modo o *Survey* contemplou todos os objetivos especificados para o estudo.

Tabela 14 – Erros de usabilidade por categoria

Severidade	**Tipo**	**Quantidade**
1	Problema cosmético apenas	6
2	Pequeno problema de usabilidade	3
3	Grande problema de usabilidade	2
4	Catástrofe de usabilidade	0
Total e erros de usabilidade		**11**

6.4 Lições do Capítulo

Tendo em vista os resultados apresentados pelo *Survey*, neste capítulo é possível concluir que a *Teasy Language*, mesmo em sua primeira versão, tem indícios de conter uma boa usabilidade, possuindo várias funcionalidades que permitem ao testador se localizar no código, entender e resolver possíveis problemas. Além disso, também foram levantados problemas para serem resolvidos em sua próxima versão, aumentando ainda mais a qualidade da linguagem. Por fim, alguns dos benefícios apresentados no *quasi*-experimento foram identificados novamente durante o *Survey*, como uma linguagem completa e simples de se utilizar, além de rapidez durante a implementação de casos de teste, entretanto os pontos negativos em relação a sua configuração também surgiram como *feedbacks* gerais.

7 CONSIDERAÇÕES FINAIS

A quantidade de estudos acadêmicos e *reports* da indústria publicados durante os últimos anos, relacionados ao uso das *Domain-Specific Languages* (DSLs) para testes funcionais de *software*, apresentam uma crescente na tentativa de inserção destas tecnologias na etapa de testes. Desta forma a detecção das desvantagens relativas às DSLs existentes através do SMS e do *Snowballing* expôs que as soluções atuais ainda carecem de melhorias. Como uma alternativa para esta problemática o *Teasy Framework* foi definido e proposto com o foco em uma solução composta por: uma DSL de uso simples, uma estrutura escalável e um gerador de sequências de teste, além da possibilidade de geração de *scripts* executáveis e relatórios para análise. O *Teasy Framework* foi idealizado com o intuito de mitigar as limitações encontradas nas demais linguagens existentes, além de manter a maioria dos benefícios. Sabendo disso, após a sua implementação foram realizadas duas avaliações, uma no formato de quasi-experimento e outra através de um *Survey* com os especialistas.

O quasi-experimento apresentou indícios de redução do esforço necessário para a criação de *scripts* de teste, código com uma sintaxe clara e objetiva, reúso de componentes, *framework* que não necessita conhecimento prévio em programação, integração direta com um ambiente de desenvolvimento, geração de sequências de teste representativas para o domínio, além da possibilidade de geração de um relatório completo da execução. Atingindo todos os objetivos e requisitos especificados, podendo ser eficaz em diversos cenários e ambientes. Entretanto essa versão ainda apresenta uma dificil configuração e algumas etapas manuais, pontos esses que foram levantados e serão priorizados para uma segunda versão do *framework*.

O *Survey* com os especialistas apresentou outros pontos fortes da linguagem Teasy. A usabilidade, representatividade do domínio e adequação com diversas Heurísticas puderam ser analisadas através de um *Checklist* Heurístico. Em todas as dez Heurísticas a DSL apresentou no mínimo um ponto positivo, enquanto ao total foram detectados apenas 11 problemas de usabilidade, sendo esses em sua maioria categorizados como cosméticos ou pequenos, ou seja o uso da *Teasy Language* não foi impactado ou prejudicado.

Mesmo com os problemas encontrados neste momento o *Teasy Framework* já se apresenta pronto para uso em uma versão totalmente funcional atendendo a todos os requisitos definidos nesse estudo. Os problemas de usabilidade apresentados no *Survey* poderão facilmente serem atualizados sem prejudicar outras partes da linguagem. O uso de uma linguagem projecional permite que seja alterado somente a parte visual e o editor sem afetar na lógica desenvolvida, respondendo assim facilmente a melhorias e mudanças de acordo com os problemas encontrados.

Ciente dos problemas detectados no *framework* nesta versão atual, nos trabalhos futuros serão realizadas as automações de partes do processo hoje manuais, também será criado um instalador a fim de reduzir a barreira de entrada para o uso da tecnologia.

Além de atender a todos os problemas de usabilidade encontrados, tendo em vista que a quantidade não é significativa. Desse modo, melhorando o *Teasy Framework* viabilizando ainda mais a sua adoção na indústria e academia. Por fim, também será elaborada uma proposta conceitual utilizando metamodelo da *Teasy Language*, juntamente com um método para a utilização do *Teasy Framework*. Assim, explicando o funcionamento interno da linguagem, além de prover diretrizes para os testadores que optarem pelo uso do *framework*.

REFERÊNCIAS

ALEGROTH, E.; BACHE, G.; BACHE, E. On the industrial applicability of texttest: An empirical case study. In: IEEE. **2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)**. [S.l.], 2015. p. 1–10. Citado na página 45.

ARTHO, C. et al. Model-based api testing of apache zookeeper. In: IEEE. **2017 IEEE International Conference on Software Testing, Verification and Validation (ICST)**. [S.l.], 2017. p. 288–298. Citado na página 55.

ARTHO, C. et al. Domain-specific languages with scala. In: SPRINGER. **International Conference on Formal Engineering Methods**. [S.l.], 2015. p. 1–16. Citado na página 55.

ARTHO, C. et al. Model-based testing of stateful apis with modbat. In: IEEE. **2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)**. [S.l.], 2015. p. 858–863. Citado na página 55.

ARTHO, C. V. et al. Modbat: A model-based api tester for event-driven systems. In: SPRINGER. **Haifa Verification Conference**. [S.l.], 2013. p. 112–128. Citado na página 45.

AVASARALA, S. **Selenium WebDriver practical guide**. [S.l.]: Packet, 2014. Citado na página 82.

BETTINI, L. **Implementing domain-specific languages with Xtext and Xtend**. [S.l.]: Packt Publishing Ltd, 2016. Citado 2 vezes nas páginas 33 e 49.

BISHT, S. **Robot framework test automation**. [S.l.]: Packt Publishing Ltd, 2013. Citado 3 vezes nas páginas 81, 82 e 84.

BITTENCOURT, M. Gestão de teste e defeitos com testlink e mantis. **1º Edição**, v. 1, 2013. Citado na página 48.

BLACKWELL, A. F.; GREEN, T. R. A cognitive dimensions questionnaire optimised for users. In: CITESEER. **PPIG**. [S.l.], 2000. v. 13. Citado na página 84.

BRODER, A. et al. Graph structure in the web. **Computer networks**, Elsevier, v. 33, n. 1-6, p. 309–320, 2000. Citado na página 72.

BUSSENOT, R.; LEBLANC, H.; PERCEBOIS, C. A domain specific test language for systems integration. In: ACM. **Proceedings of the Scientific Workshop Proceedings of XP2016**. [S.l.], 2016. p. 16. Citado na página 56.

BUSSENOT, R.; LEBLANC, H.; PERCEBOIS, C. Orchestration of domain specific test languages with a behavior driven development approach. In: IEEE. **2018 13th Annual Conference on System of Systems Engineering (SoSE)**. [S.l.], 2018. p. 431–437. Citado na página 56.

CARTAXO, E. G. et al. Lts-bt: a tool to generate and select functional test cases for embedded systems. In: ACM. **Proceedings of the 2008 ACM symposium on Applied computing**. [S.l.], 2008. p. 1540–1544. Citado na página 50.

CARVALHO, R. E. V. de. A comparative study of gui testing approaches. 2016. Citado na página 56.

CHATLEY, R.; AYRES, J.; WHITE, T. Lift: Driving development using a business-readable dsl for web testing. In: IEEE. **2010 Third International Conference on Software Testing, Verification, and Validation Workshops**. [S.l.], 2010. p. 460–468. Citado na página 45.

CIMATTI, A. et al. Nusmv: A new symbolic model verifier. In: SPRINGER. **International conference on computer aided verification**. [S.l.], 1999. p. 495–499. Citado na página 49.

CONTAN, A.; MICLEA, L.; DEHELEAN, C. Automated testing framework development based on social interaction and communication principles. In: IEEE. **2017 14th International Conference on Engineering of Modern Electric Systems (EMES)**. [S.l.], 2017. p. 136–139. Citado na página 45.

COSTA, P.; PAIVA, A. C.; NABUCO, M. Pattern based gui testing for mobile applications. In: IEEE. **2014 9th International Conference on the Quality of Information and Communications Technology**. [S.l.], 2014. p. 66–74. Citado na página 55.

CUNHA, C. D.; SONG, M. A. An environment for automatic tests generation from use case specifications. In: THE STEERING COMMITTEE OF THE WORLD CONGRESS IN COMPUTER SCIENCE, COMPUTER .... **Proceedings of the International Conference on Software Engineering Research and Practice (SERP)**. [S.l.], 2014. p. 1. Citado na página 55.

DELAMARO, M.; JINO, M.; MALDONADO, J. **Introdução ao teste de software**. [S.l.]: Elsevier Brasil, 2013. Citado 2 vezes nas páginas 27 e 31.

DEV, S. **Page object models**. 2021. Disponível em: <https://www.selenium.dev/documentation/en/guidelines_and_recommendations/page_object_models/g>. Citado na página 68.

Dias, F.; Paiva, A. C. R. Pattern-based usability testing. In: **2017 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)**. [S.l.: s.n.], 2017. p. 366–371. Citado na página 55.

DUCLOS, E. et al. Acre: An automated aspect creator for testing c++ applications. In: IEEE. **2013 17th European Conference on Software Maintenance and Reengineering**. [S.l.], 2013. p. 121–130. Citado na página 45.

DWARAKANATH, A. et al. Accelerating test automation through a domain specific language. In: IEEE. **2017 IEEE International Conference on Software Testing, Verification and Validation (ICST)**. [S.l.], 2017. p. 460–467. Citado na página 45.

ELODIE, B. et al. Lightweight model-based testing for enterprise it. In: IEEE. **2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)**. [S.l.], 2018. p. 224–230. Citado na página 45.

FELDERER, M.; JESCHKO, F. A process for evidence-based engineering of domain-specific languages. In: ACM. **Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018**. [S.l.], 2018. p. 169–174. Citado na página 45.

FIREFOX, M. Mozilla firefox. **Capturado em: http://ptbr. www. mozilla. com/pt-BR/firefox**, 2008. Citado na página 69.

FIREFOX, M. Google chrome. **Opera ( )** , 2018. Citado na página 69.

FOWLER, M. Language workbench. 2005. Disponível em: <https://martinfowler.com/bliki/LanguageWorkbench.html>. Citado na página 33.

FOWLER, M. Internal dsl style. 2006. Disponível em: <https://martinfowler.com/bliki/InternalDslStyle.html>. Citado na página 32.

FOWLER, M. **Domain-specific languages**. [S.l.]: Pearson Education, 2010. Citado 3 vezes nas páginas 27, 32 e 33.

GAFUROV, D.; HURUM, A. E.; MARKMAN, M. Achieving test automation with testers without coding skills: An industrial report. In: ACM. **Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering**. [S.l.], 2018. p. 749–756. Citado na página 45.

GOSLING, J.; HOLMES, D. C.; ARNOLD, K. **The Java programming language**. [S.l.]: Addison-Wesley, 2005. Citado na página 32.

GOSLING, J. et al. **The Java language specification**. [S.l.]: Addison-Wesley Professional, 2000. Citado na página 82.

HAMMOUD, D.; ZARAKET, F. A.; MASRI, W. Guicop: Approach and toolset for specification-based gui testing. **Software Testing, Verification and Reliability**, Wiley Online Library, v. 27, n. 8, p. e1642, 2017. Citado na página 55.

HARGASSNER, W. et al. A script-based testbed for mobile software frameworks. In: IEEE. **2008 1st International Conference on Software Testing, Verification, and Validation**. [S.l.], 2008. p. 448–457. Citado na página 45.

HÄRLIN, M. **Testing and Gherkin in agile projects**. 2016. Citado na página 56.

HÄSER, F.; FELDERER, M.; BREU, R. Test process improvement with documentation driven integration testing. In: IEEE. **2014 9th International Conference on the Quality of Information and Communications Technology**. [S.l.], 2014. p. 156–161. Citado na página 45.

HÄSER, F.; FELDERER, M.; BREU, R. An integrated tool environment for experimentation in domain specific language engineering. In: ACM. **Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering**. [S.l.], 2016. p. 20. Citado na página 45.

HÄSER, F.; FELDERER, M.; BREU, R. Is business domain language support beneficial for creating test case specifications: A controlled experiment. **Information and software technology**, Elsevier, v. 79, p. 52–62, 2016. Citado na página 55.

HÄSER, F.; FELDERER, M.; BREU, R. Evaluation of an integrated tool environment for experimentation in dsl engineering. In: SPRINGER. **International Conference on Software Quality**. [S.l.], 2018. p. 147–168. Citado na página 55.

HERBOLD, S. et al. The midas cloud platform for testing soa applications. In: IEEE. **2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)**. [S.l.], 2015. p. 1–8. Citado na página 45.

HOIS, B.; SOBERNIG, S.; STREMBECK, M. Natural-language scenario descriptions for testing core language models of domain-specific languages. In: IEEE. **2014 2nd International Conference on Model-Driven Engineering and Software Development (MODELSWARD)**. [S.l.], 2014. p. 356–367. Citado na página 55.

HOLMES, J. et al. Tstl: the template scripting testing language. **International Journal on Software Tools for Technology Transfer**, Springer, v. 20, n. 1, p. 57–78, 2018. Citado na página 45.

HOLZMANN, G. J. **The SPIN model checker: Primer and reference manual**. [S.l.]: Addison-Wesley Reading, 2004. v. 1003. Citado na página 50.

HU, G.; ZHU, L.; YANG, J. Appflow: using machine learning to synthesize robust, reusable ui tests. In: ACM. **Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering**. [S.l.], 2018. p. 269–282. Citado na página 55.

IBER, J. et al. Ubtl uml testing profile based testing language. In: IEEE. **2015 3rd International Conference on Model-Driven Engineering and Software Development (MODELSWARD)**. [S.l.], 2015. p. 1–12. Citado na página 45.

IBER, J. et al. A textual domain-specific language based on the uml testing profile. In: SPRINGER. **International Conference on Model-Driven Engineering and Software Development**. [S.l.], 2015. p. 155–171. Citado na página 45.

JETBRAINS. How does mps work?. 2019. Disponível em: <https://www.jetbrains.com/mps/concepts/index.html\#projection-editor>. Citado na página 33.

JETBRAINS, M. Meta programming system. 2014. Disponível em: <http://www.jetbrains.com/mps>. Citado na página 29.

JORGE, D. N. et al. Integrating requirements specification and model-based testing in agile development. In: IEEE. **2018 IEEE 26th International Requirements Engineering Conference (RE)**. [S.l.], 2018. p. 336–346. Citado na página 45.

KANSTRÉN, T.; PUOLITAIVAL, O.-P. Using built-in domain-specific modeling support to guide model-based test generation. **Model-Driven Engineering of Information Systems: Principles, Techniques, and Practice**, p. 295–319, 2012. Citado na página 55.

KESSERWAN, N. et al. From use case maps to executable test procedures: a scenario-based approach. **Software & Systems Modeling**, Springer, v. 18, n. 2, p. 1543–1570, 2019. Citado na página 56.

KING, T. M. et al. Legend: an agile dsl toolset for web acceptance testing. In: ACM. **Proceedings of the 2014 International Symposium on Software Testing and Analysis**. [S.l.], 2014. p. 409–412. Citado na página 45.

KITCHENHAM, B.; CHARTERS, S. Guidelines for performing systematic literature reviews in software engineering. Citeseer, 2007. Citado na página 36.

KOS, T. et al. From dcom interfaces to domain-specific modeling language: A case study on the sequencer. **Computer Science and Information Systems**, Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko, v. 8, n. 2, p. 361–378, 2011. Citado na página 55.

KOS, T. et al. Model refactoring within a sequencer. In: WORLD SCIENTIFIC AND ENGINEERING ACADEMY AND SOCIETY (WSEAS). **Proceedings of the 5th WSEAS congress on Applied Computing conference, and Proceedings of the 1st international conference on Biologically Inspired Computation**. [S.l.], 2012. p. 90–94. Citado na página 55.

KOS, T. et al. Sett: testing-tool for measurement system dewesoft. In: **Proceedings of the 2nd International conference on design and product development (ICDPD'11), WSEAS Press**. [S.l.: s.n.], 2011. p. 111–115. Citado na página 55.

KOS, T.; MERNIK, M.; KOSAR, T. Test automation of a measurement system using a domain-specific modelling language. **Journal of Systems and Software**, Elsevier, v. 111, p. 74–88, 2016. Citado na página 45.

LEE-KWANG, H.; FAVREL, J.; BAPTISTE, P. Generalized petri net reduction method. **IEEE transactions on systems, man, and cybernetics**, IEEE, v. 17, n. 2, p. 297–303, 1987. Citado na página 50.

LIMA, Y. A. et al. Mps x xtext: Uma comparação de languages workbenches para o desenvolvimento de dsls. In: **Anais da III Escola Regional de Engenharia de Software**. Porto Alegre, RS, Brasil: SBC, 2019. p. 97–104. ISSN 0000-0000. Disponível em: <https://sol.sbc.org.br/index.php/eres/article/view/8501>. Citado na página 68.

LUNA, E. R. et al. Capture and evolution of web requirements using webspec. In: SPRINGER. **International Conference on Web Engineering**. [S.l.], 2010. p. 173–188. Citado na página 45.

LUNA, E. R.; GARRIGOS, I.; ROSSI, G. Capturing and validating personalization requirements in web applications. In: IEEE. **2010 First International Workshop on the Web and Requirements Engineering**. [S.l.], 2010. p. 13–20. Citado na página 55.

LUNA, E. R.; ROSSI, G.; GARRIGÓS, I. Webspec: a visual language for specifying interaction and navigation requirements in web applications. **Requirements Engineering**, Springer, v. 16, n. 4, p. 297, 2011. Citado na página 45.

LUTZ, M. **Programming python**. [S.l.]: "O'Reilly Media, Inc.", 2001. Citado na página 82.

MACIEL, D. A. M. Model based testing-from requirements to tests. 2019. Citado na página 56.

MAKEDONSKI, P. et al. Evolving the etsi test description language. In: SPRINGER. **International Conference on System Analysis and Modeling**. [S.l.], 2016. p. 116–131. Citado na página 45.

MAKEDONSKI, P. et al. Test descriptions with etsi tdl. **Software Quality Journal**, Springer, p. 1–33, 2019. Citado na página 45.

MCDOWELL, C. et al. The impact of pair programming on student performance, perception and persistence. In: IEEE. **25th International Conference on Software Engineering, 2003. Proceedings.** [S.l.], 2003. p. 602–607. Citado na página 65.

MERNIK, M.; HEERING, J.; SLOANE, A. M. When and how to develop domain-specifi c languages. **ACM computing surveys (CSUR)**, ACM, v. 37, n. 4, p. 316–344, 2005. Citado na página 32.

MICALLEF, M.; COLOMBO, C. Lessons learnt from using dsls for automated software testing. In: IEEE. **2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)**. [S.l.], 2015. p. 1–6. Citado na página 55.

MILLER, A.; KUMAR, B.; SINGHAL, A. Photon: A domain-specific language for testing converged applications. In: IEEE. **2009 33rd Annual IEEE International Computer Software and Applications Conference**. [S.l.], 2009. v. 2, p. 269–274. Citado na página 45.

MONTEIRO, T.; PAIVA, A. C. Pattern based gui testing modeling environment. In: IEEE. **2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops**. [S.l.], 2013. p. 140–143. Citado na página 55.

MOREIRA, R. M.; PAIVA, A. C. A gui modeling dsl for pattern-based gui testing paradigm. In: IEEE. **2014 9th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE)**. [S.l.], 2014. p. 1–10. Citado na página 55.

MOREIRA, R. M.; PAIVA, A. C. Pbgt tool: an integrated modeling and testing environment for pattern-based gui testing. In: ACM. **Proceedings of the 29th ACM/IEEE international conference on Automated software engineering**. [S.l.], 2014. p. 863–866. Citado na página 55.

MOREIRA, R. M.; PAIVA, A. C. Towards a pattern language for model-based gui testing. In: ACM. **Proceedings of the 19th European Conference on Pattern Languages of Programs**. [S.l.], 2014. p. 26. Citado na página 55.

MOREIRA, R. M.; PAIVA, A. C.; MEMON, A. A pattern-based approach for gui modeling and testing. In: IEEE. **2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE)**. [S.l.], 2013. p. 288–297. Citado na página 55.

MOREIRA, R. M. et al. Pattern-based gui testing: Bridging the gap between design and quality assurance. **Software Testing, Verification and Reliability**, Wiley Online Library, v. 27, n. 3, p. e1629, 2017. Citado 2 vezes nas páginas 27 e 55.

MORGADO, I. C.; PAIVA, A. C. Test patterns for android mobile applications. In: ACM. **Proceedings of the 20th European Conference on Pattern Languages of Programs**. [S.l.], 2015. p. 32. Citado na página 55.

MYERS, G. J.; SANDLER, C.; BADGETT, T. **The art of software testing**. [S.l.]: John Wiley & Sons, 2011. Citado na página 31.

NABUCO, M.; PAIVA, A. C. Model-based test case generation for web applications. In: SPRINGER. **International Conference on Computational Science and Its Applications**. [S.l.], 2014. p. 248–262. Citado na página 45.

NEGRINO, T.; SMITH, D. **JavaScript para world wide web**. [S.l.]: Pearson Education, 2001. Citado na página 32.

NIELSEN, J.; MOLICH, R. Heuristic evaluation of user interfaces. In: **Proceedings of the SIGCHI conference on Human factors in computing systems**. [S.l.: s.n.], 1990. p. 249–256. Citado 2 vezes nas páginas 29 e 95.

OLAJUBU, O. et al. Automated test case generation from high-level logic requirements using model transformation techniques. In: IEEE. **2017 9th Computer Science and Electronic Engineering (CEEC)**. [S.l.], 2017. p. 178–182. Citado na página 55.

OTADUY, I.; DÍAZ, O. User acceptance testing for agile-developed web-based applications: Empowering customers through wikis and mind maps. **Journal of Systems and Software**, Elsevier, v. 133, p. 212–229, 2017. Citado na página 55.

PAIVA, A. C.; VILELA, L. Multidimensional test coverage analysis: Paradigm-cov tool. **Cluster Computing**, Springer, v. 20, n. 1, p. 633–649, 2017. Citado na página 55.

PETERSEN, K. et al. Systematic mapping studies in software engineering. In: **Ease**. [S.l.: s.n.], 2008. v. 8, p. 68–77. Citado na página 35.

POLTRONIERI, I. et al. Heuristic evaluation checklist for domain-specific languages. In: INSTICC. **Proceedings of the 16th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - Volume 1: HUCAPP,**. [S.l.]: SciTePress, 2021. p. 37–48. ISBN 978-989-758-488-6. Citado na página 95.

POLTRONIERI, I. et al. Usa-dsl: Usability evaluation framework for domain-specific languages. In: **Proceedings of the 33rd Annual ACM Symposium on Applied Computing**. New York, NY, USA: Association for Computing Machinery, 2018. (SAC '18), p. 2013–2021. ISBN 9781450351911. Disponível em: <https://doi.org/10.1145/3167132.3167348>. Citado 2 vezes nas páginas 81 e 95.

PRASETYA, I. et al. Using haskell to script combinatoric testing of web services. In: IEEE. **6th Iberian Conference on Information Systems and Technologies (CISTI 2011)**. [S.l.], 2011. p. 1–6. Citado na página 45.

RAGHAVENDRA, S. Page objects. In: **Python Testing with Selenium**. [S.l.]: Springer, 2021. p. 143–150. Citado na página 68.

RIOS, E. et al. **Base de Conhecimento em Teste de Software. 2ª Edição–S. Paulo**. [S.l.]: Martins Fontes, 2007. Citado na página 31.

RODRIGUES, F. C. Pattern based usability testing. 2018. Citado na página 56.

SACRAMENTO, C. Reverse engineering of interaction patterns. 2014. Citado na página 56.

SACRAMENTO, C.; PAIVA, A. C. Web application model generation through reverse engineering and ui pattern inferring. In: IEEE. **2014 9th International Conference on the Quality of Information and Communications Technology**. [S.l.], 2014. p. 105–115. Citado na página 56.

SANTIAGO, D. et al. Towards domain-specific testing languages for software-as-a-service. In: CITESEER. **MDHPCL@ MoDELS**. [S.l.], 2013. p. 43–52. Citado na página 55.

SANZ, C. et al. Automated model-based testing based on an agnostic-platform modeling language. In: IEEE. **2015 3rd International Conference on Model-Driven Engineering and Software Development (MODELSWARD)**. [S.l.], 2015. p. 1–8. Citado na página 45.

SILVA, A. R. da; PAIVA, A. C.; SILVA, V. E. da. A test specification language for information systems based on data entities, use cases and state machines. In: SPRINGER. **International Conference on Model-Driven Engineering and Software Development**. [S.l.], 2018. p. 455–474. Citado na página 55.

SILVA, A. R. da; PAIVA, A. C.; SILVA, V. E. R. da. Towards a test specification language for information systems: Focus on data entity and state machine tests. In: **MODELSWARD**. [S.l.: s.n.], 2018. p. 213–224. Citado na página 55.

SILVA, T. R. **A behavior-driven approach for specifying and testing user requirements in interactive systems**. Tese (Doutorado) — Université de Toulouse, Université Toulouse III-Paul Sabatier, 2018. Citado na página 56.

SILVA, V. E. R. da. Model based testing-from requirements to tests. 2017. Citado na página 56.

SIPPL, C. et al. From simulation data to test cases for fully automated driving and adas. In: SPRINGER. **IFIP International Conference on Testing Software and Systems**. [S.l.], 2016. p. 191–206. Citado na página 45.

SIRIUS. **Sirius. https://www.eclipse.org/sirius/**. 2019. Disponível em: <https://www.eclipse.org/sirius/>. Citado na página 33.

SOUZA, G. et al. Diretrizes para uma metodologia de desenvolvimento de software aplicada a startups de tecnologia da informação. **XI Simpósio Brasileiro de Sistemas de Informação**, v. 2, n. 1, p. 32–35, 2017. Citado 2 vezes nas páginas 27 e 28.

TALBY, D. The perceived value of authoring and automating acceptance tests using a model driven development toolset. In: IEEE. **2009 ICSE Workshop on Automation of Software Test**. [S.l.], 2009. p. 154–157. Citado na página 45.

TARJAN, R. Depth-first search and linear graph algorithms. **SIAM journal on computing**, SIAM, v. 1, n. 2, p. 146–160, 1972. Citado na página 72.

THOMPSON, S. **Haskell: the craft of functional programming**. [S.l.]: Addison-Wesley, 2011. v. 2. Citado na página 49.

TORSEL, A.-M. Automated test case generation for web applications from a domain specific model. In: IEEE. **2011 IEEE 35th Annual Computer Software and Applications Conference Workshops**. [S.l.], 2011. p. 137–142. Citado na página 55.

TÖRSEL, A.-M. A testing tool for web applications using a domain-specific modelling language and the nusmv model checker. In: IEEE. **2013 IEEE Sixth International Conference on Software Testing, Verification and Validation**. [S.l.], 2013. p. 383–390. Citado 4 vezes nas páginas 27, 45, 60 e 68.

ULRICH, A. et al. The etsi test description language tdl and its application. In: IEEE. **2014 2nd International Conference on Model-Driven Engineering and Software Development (MODELSWARD)**. [S.l.], 2014. p. 601–608. Citado na página 55.

UTTING, M.; LEGEARD, B. **Practical model-based testing: a tools approach**. [S.l.]: Elsevier, 2010. Citado 2 vezes nas páginas 27 e 31.

VILELA, L.; PAIVA, A. C. Paradigm-cov: A multimensional test coverage analysis tool. In: IEEE. **2014 9th Iberian Conference on Information Systems and Technologies (CISTI)**. [S.l.], 2014. p. 1–7. Citado na página 55.

VILELA, L. B. Test coverage analysis. 2013. Citado na página 56.

W3C. **WebDriver W3C Recommendation**. 2018. <https://www.w3.org/TR/webdriver1/>. Online; acessado 29 Fevereiro de 2020. Citado na página 83.

WINKLER, D.; MEIXNER, K.; BIFFL, S. Towards flexible and automated testing in production systems engineering projects. In: IEEE. **2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)**. [S.l.], 2018. v. 1, p. 169–176. Citado na página 55.

WOHLIN, C. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In: **Proceedings of the 18th international conference on evaluation and assessment in software engineering**. [S.l.: s.n.], 2014. p. 1–10. Citado na página 52.

YEH, T.; CHANG, T.-H.; MILLER, R. C. Sikuli: using gui screenshots for search and automation. In: ACM. **Proceedings of the 22nd annual ACM symposium on User interface software and technology**. [S.l.], 2009. p. 183–192. Citado na página 48.

ZHOU, J.; YIN, K. Automated web testing based on textual-visual ui patterns: the utf approach. **ACM SIGSOFT Software Engineering Notes**, ACM, v. 39, n. 5, p. 1–6, 2014. Citado na página 45.

ŽIVANOV, Ž.; RAKIĆ, P.; HAJDUKOVIĆ, M. Using code generation approach in developing kiosk applications. **Computer Science and Information Systems**, ComSIS Consortium, v. 5, n. 1, p. 41–59, 2008. Citado na página 32.

Apêndices

Anexos

ANEXO A – FORMULÁRIOS UTILIZADOS NO QUASI-EXPERIMENTO

A.1 Avaliação do *Teasy Framework*

Teasy Framework seu feedback é importante !

Prezado(a), este questionário destina-se a coletar dados para uma pesquisa na área de Engenharia de Software, conduzida na Universidade Federal do Pampa (UNIPAMPA), com objetivo de avaliar todo o framework Teasy. A sua participação é de suma importância para que essa linguagem possa ser melhorada. Tenha em vista que os dados informados são para fins de análise e que não é você que está sendo avaliado e sim a facilidade no uso do framework Teasy. As informações obtidas serão confidenciais e asseguramos o sigilo sobre sua participação. Os dados publicados não permitirão a sua identificação. As informações obtidas serão utilizadas de modo único e excluso para os fins específicos deste estudo.

Yury Alencar Lima <yuryalencar19@gmail.com>
Elder de Macedo Rodrigues <eldermr@gmail.com>
Rafael Paes Oliveira <olivrap@gmail.com>
Ildevana Poltronieri Rodrigues <ildevana.rodrigues@acad.pucrs.br>

* Required

1. Email address *

2. Termo de Consentimento *

Check all that apply.

☐ Concordo em participar deste estudo e declaro ter lido os detalhes descritos neste formulário. Eu entendo que sou livre para aceitar ou recusar, e que posso interromper minha participação a qualquer momento sem indicar o motivo. Concordo que os dados coletados serão utilizados para os fins descritos acima. Compreendo as informações apresentadas neste TERMO DE CONSENTIMENTO. Tive a oportunidade de fazer perguntas e todas as minhas perguntas foram respondidas. Receberei uma cópia assinada e datada deste Termo de Consentimento Livre e Esclarecido.

Vamos nos conhecer

Seção para sabermos melhor o seu perfil

3. 01) Você já utilizou uma Linguagem de Domínio Específico (DSL) ? *

   Mark only one oval.

   - ◯ Sim
   - ◯ Não

4. 02) Você já criou alguma Linguagem de Domínio Específico (DSL) ? *

   Mark only one oval.

   - ◯ Sim
   - ◯ Não

5. 03) Quais estudos empíricos já participou ? *

   Check all that apply.

   - ☐ Estudo de Caso
   - ☐ Experimento Controlado
   - ☐ Quase Experimento
   - ☐ Survey (Questionários)
   - ☐ Provas de conceito
   - ☐ Nunca participei de um estudo empírico

6. 04) Qual categoria(s) você pertence atualmente ? *

   Check all that apply.

   - ☐ Academia
   - ☐ Indústria

7. 05) Qual é o seu grau de escolaridade? *

Mark only one oval.

- ◯ Ensino fundamental completo
- ◯ Ensino médio incompleto
- ◯ Ensino médio completo
- ◯ Graduação incompleta
- ◯ Graduação completa
- ◯ Mestrado incompleto
- ◯ Mestrado completo
- ◯ Doutorado incompleto
- ◯ Doutorado completo
- ◯ Other: _____

8. 05) Quanto tempo possui de experiência na Indústria ? *

Mark only one oval.

- ◯ Não possuo
- ◯ Até 1 ano
- ◯ Entre 1 ano e 3 anos
- ◯ Entre 3 anos e 6 anos
- ◯ Entre 6 anos 9 anos
- ◯ acima de 9 anos

9. 06) Quanto tempo possui de experiência com testes de software? *

Mark only one oval.

- ◯ Não possuo
- ◯ Até 1 ano
- ◯ Entre 1 ano e 3 anos
- ◯ Entre 3 anos e 6 anos
- ◯ Entre 6 anos e 9 anos
- ◯ Acima de 9 anos

Teasy Language

Seção para nos dar um feedback sobre a linguagem Teasy

10. 01) A IDE (MPS) é fácil de ser utilizada. *

Mark only one oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo Totalmente

11. Caso sua resposta seja inferior a 3, Justifique a sua resposta para colaborar com a melhoria do Framework.

12. 02) A linguagem Teasy pode reduzir o tempo necessário para implementar testes funcionais de aplicações web. *

Mark only one oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

13. Caso sua resposta seja inferior a 3, Justifique a sua resposta para colaborar com a melhoria do Framework.

14. 03) A linguagem Teasy possibilita um fácil reúso dos componentes agilizando o tempo necessário para implementação dos testes funcionais. *

Mark only one oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

15. Caso sua resposta seja inferior a 3, Justifique a sua resposta para colaborar com a melhoria do Framework.

16. 04) Você utilizaria a linguagem Teasy em seus projetos ? *

Mark only one oval.

- ◯ Sim
- ◯ Não

17. Justifique a sua escolha *

_____

_____

_____

_____

_____

18. 05) As mensagens de ajuda da Teasy são fáceis de compreender e ajudam na implementação. *

Mark only one oval.

- ◯ Sim
- ◯ Não
- ◯ Other: _____

19. Em caso negativo justifique a sua resposta

_____

_____

_____

_____

_____

20. 06) É fácil identificar ou localizar todas as informações necessárias da linguagem enquanto se está utilizando. *

Mark only one oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

21. Caso sua resposta seja inferior a 3, Justifique a sua resposta para colaborar com a melhoria do Framework.

22. 07) É fácil realizar atualizações e manutenções em um código Teasy já existente. *

Mark only one oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

23. Caso sua resposta seja inferior a 3, Justifique a sua resposta para colaborar com a melhoria do Framework.

24. 08) A linguagem Teasy permite que você insira o que deseja de maneira razoavelmente breve. (Pouco verboso) *

Mark only one oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

25. Caso sua resposta seja inferior a 3, Justifique a sua resposta para colaborar com a melhoria do Framework.

26. 09) A linguagem Teasy necessita de pouco esforço mental em seu uso. *

Mark only one oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

27. Caso sua resposta seja inferior a 3, Justifique a sua resposta para colaborar com a melhoria do Framework.

28. 10) A sintaxe da linguagem Teasy pode levar o usuário a cometer erros facilmente. *

Mark only one oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

29. Caso sua resposta seja inferior a 3, Justifique a sua resposta para colaborar com a melhoria do Framework.

30. 11) É possível relacionar facilmente o código gerado pela linguagem Teasy com a implementação realizada na DSL. *

Mark only one oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

31. Caso sua resposta seja inferior a 3, Justifique a sua resposta para colaborar com a melhoria do Framework.

32. 12) O código gerado pela Teasy possui fácil leitura. *

Mark only one oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

33. Caso sua resposta seja inferior a 3, Justifique a sua resposta para colaborar com a melhoria do Framework.

34. 13) Ao ler um código escrito na linguagem Teasy, é fácil dizer para que serve cada arquivo e funcionalidade no contexto de testes funcionais. *

Mark only one oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

35. Caso sua resposta seja inferior a 3, Justifique a sua resposta para colaborar com a melhoria do Framework.

36. 14) Você entendeu para que serve cada parte da sintaxe da DSL ou apenas não inseriu uma informação para que o teste funcionasse. *

Mark only one oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

37. Caso sua resposta seja inferior a 3, Justifique a sua resposta para colaborar com a melhoria do Framework.

38. 15) Todas as informações necessárias para o uso da linguagem estão visíveis e de fácil acesso. *

Mark only one oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

39. Caso sua resposta seja inferior a 3, Justifique a sua resposta para colaborar com a melhoria do Framework.

40. 16) As dependências se atualizam automaticamente e isso facilita o uso e manutenção do código desenvolvido utilizando a Teasy. *

Mark only one oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

41. Caso sua resposta seja inferior a 3, Justifique a sua resposta para colaborar com a melhoria do Framework.

42. 17) Os highlights (destaques) em cores diferentes estão coerentes e ajudam no entendimento no uso da linguagem. *

Mark only one oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

43. Caso sua resposta seja inferior a 3, Justifique a sua resposta para colaborar com a melhoria do Framework.

44. 18) O código gerado pela Teasy possui fácil manutenção caso seja necessário. *

Mark only one oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

45. Caso sua resposta seja inferior a 3, Justifique a sua resposta para colaborar com a melhoria do Framework.

Teasy Structure

Seção para nos dar um feedback sobre a TeasyStructure

46. 01) A TeasyStructure reduz o tempo necessário para configurar o projeto de executação do código gerado. *

Mark only one oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

47. 02) A TeasyStructure pode ser utilizada independente do uso da Teasy Language. *

Mark only one oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

48. 03) A TeasyStructure é fácil de se entender. *

Mark only one oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

49. 04) A TeasyStructure torna fácil a realização de manutenções futuras. *

Mark only one oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

50. 05) O relatório gerado pela Teasy auxilia na análise e detecção de bugs. *

Mark only one oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

51. 06) Você utilizaria o TeasyStructure para auxiliar na qualidade de suas aplicações? *

Mark only one oval.

- ◯ Sim
- ◯ Não
- ◯ Other: _____

52. Justifique a sua resposta *

Teasy Generator

Seção para nos dar um feedback sobre o TeasyStructure

53. 01) O TeasyGenerator pode encontrar problemas que você como testador não teria pensado. *

Mark only one oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

54. 02) O código gerado pelo TeasyGenerator é de fácil compreensão. *

Mark only one oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

55. 03) O código gerado pelo TeasyGenerator é de fácil manutenção. *

Mark only one oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

56. 04) O TeasyGenerator é de fácil usar. *

Mark only one oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

57. 05) Caso a resposta anterior seja negativa (abaixo de 3), pode explicitar o motivo ?

_____

_____

_____

_____

_____

58. 06) O relatório gerado pelo código do TeasyGenerator pode auxiliar na detecção de problemas. *

Mark only one oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

59. 07) As sequências do TeasyGenerator são representativas para aplicação e fazem sentido no respectivo contexto. *

Mark only one oval.

- ◯ São representativas
- ◯ A maioria faz sentido
- ◯ A menor parte faz sentido
- ◯ Todas não fazem sentido
- ◯ Other: _____

60. 08) Você utilizaria o TeasyGenerator para auxiliar na etapa de qualidade das suas aplicações? *

Mark only one oval.

- ◯ Sim
- ◯ Não
- ◯ Other: _____

61. Justifique sua resposta. *

_____

_____

_____

_____

_____

Treinamento Teasy

O que achou do Treinamento ?

62. 01) O treinamento foi capaz de apresentar de forma clara e objetiva como utilizar o Teasy Framework. *

Mark only one oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

63. 02) Na sua opinião o(s) treinamento(s) podem facilitar o uso do framework ? *

Mark only one oval.

- ◯ Sim
- ◯ Não

64. Justifique sua resposta *

65. 03) Você participaria de novos treinamentos neste modelo ? *

Mark only one oval.

- Sim
- Não

66. 04) O treinamento foi cansativo ? *

Mark only one oval.

- Sim
- Não

67. 05) Se tiver algum feedback, ou sugestão insira abaixo, assim consigo lançar treinamentos ainda melhores.

Considerações Finais

Até aqui você ja deve estar um pouco cansado. Mas espero que tenha conseguido aprender um novo conteúdo, você foi e será muito importante para a melhoria deste framework. Creio que alguns vídeos possam ter qualidades diferentes, mas estamos melhorando o treinamento com o feedback de vocês, obrigado.

68. 01) Na sua opnião quais são os pontos positivos do framework Teasy ? *

69. 02) Na sua opnião, quais são os pontos negativos do framework Teasy ? *

70. 03) Tem alguma sugestão de melhoria ? Se sim, ficarei feliz em implementar, basta descrever abaixo.

71. 04) Você teria interesse em participar de novos estudos nesta área ou nesta estrutura ? *

Mark only one oval.

- ◯ Sim
- ◯ Não

72. Se sim, qual seu e-mail?

73. Agradecemos a sua participação e dedicamos este espaço para você. Caso tenha alguma sugestão ou informação que não foi abordada nesta avaliação ou na Teasy, poderia descreve-lá para que possamos melhorar nosso estudo.

A.2 Avaliação do *Robot Framework*

Robot Framework seu feedback é importante !

Prezado(a), este questionário destina-se a coletar dados para uma pesquisa na área de Engenharia de Software, conduzida na Universidade Federal do Pampa (UNIPAMPA), com objetivo de avaliar todo o framework Robot. A sua participação é de suma importância para que essa linguagem possa ser melhorada. Tenha em vista que os dados informados são para fins de análise e que não é você que está sendo avaliado e sim a facilidade no uso do framework Robot. As informações obtidas serão confidenciais e asseguramos o sigilo sobre sua participação. Os dados publicados não permitirão a sua identificação. As informações obtidas serão utilizadas de modo único e excluso para os fins específicos deste estudo.

Yury Alencar Lima <yuryalencar19@gmail.com>
Elder de Macedo Rodrigues <eldermr@gmail.com>
Rafael Paes Oliveira <olivrap@gmail.com>
Ildevana Poltronieri Rodrigues <ildevana.rodrigues@acad.pucrs.br>

* Required

1. Email address *

2. Termo de Consentimento *

Check all that apply.

☐ Concordo em participar deste estudo e declaro ter lido os detalhes descritos neste formulário. Eu entendo que sou livre para aceitar ou recusar, e que posso interromper minha participação a qualquer momento sem indicar o motivo. Concordo que os dados coletados serão utilizados para os fins descritos acima. Compreendo as informações apresentadas neste TERMO DE CONSENTIMENTO. Tive a oportunidade de fazer perguntas e todas as minhas perguntas foram respondidas. Receberei uma cópia assinada e datada deste Termo de Consentimento Livre e Esclarecido.

Vamos nos conhecer

Seção para sabermos melhor o seu perfil

3. 01) Quais estudos empíricos já participou ? *

Check all that apply.

- [ ] Estudo de Caso
- [ ] Experimento Controlado
- [ ] Quase Experimento
- [ ] Survey (Questionários)
- [ ] Provas de conceito
- [ ] Nunca participei de um estudo empírico

4. 02) Qual categoria(s) você pertence atualmente ? *

Check all that apply.

- [ ] Academia
- [ ] Indústria

5. 03) Qual é o seu grau de escolaridade? *

Mark only one oval.

- ◯ Ensino fundamental completo
- ◯ Ensino médio incompleto
- ◯ Ensino médio completo
- ◯ Graduação incompleta
- ◯ Graduação completa
- ◯ Mestrado incompleto
- ◯ Mestrado completo
- ◯ Doutorado incompleto
- ◯ Doutorado completo
- ◯ Other: _____

6. 04) Quanto tempo possui de experiência na Indústria ? *

Mark only one oval.

- ◯ Não possuo
- ◯ Até 1 ano
- ◯ Entre 1 ano e 3 anos
- ◯ Entre 3 anos e 6 anos
- ◯ Entre 6 anos 9 anos
- ◯ acima de 9 anos

7. 05) Quanto tempo possui de experiência com testes de software? *

Mark only one oval.

- ◯ Não possuo
- ◯ Até 1 ano
- ◯ Entre 1 ano e 3 anos
- ◯ Entre 3 anos e 6 anos
- ◯ Entre 6 anos e 9 anos
- ◯ Acima de 9 anos

Robot Framework

Seção para nos dar um feedback sobre a linguagem Teasy

8. 01) A IDE que realizou o teste é fácil de ser utilizada. *

Mark only one oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo Totalmente

9. Caso sua resposta seja inferior a 3, Justifique a sua resposta

10. 02) O Robot Framework pode reduzir o tempo necessário para implementar testes funcionais de aplicações web. *

Mark only one oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

11. Caso sua resposta seja inferior a 3, Justifique a sua resposta

12. 03) O Robot Framework possibilita um fácil reúso dos componentes agilizando o tempo necessário para implementação dos testes funcionais. *

Mark only one oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

13. Caso sua resposta seja inferior a 3, Justifique a sua resposta

14. 04) Você utilizaria o Robot Framework em seus projetos ? *

Mark only one oval.

- ◯ Sim
- ◯ Não

15. Justifique a sua escolha *

16. 05) As mensagens de ajuda da Robot são fáceis de compreender e ajudam na implementação. *

Mark only one oval.

- ◯ Sim
- ◯ Não
- ◯ Other: _____

17. Em caso negativo justifique a sua resposta

18. 06) É fácil identificar ou localizar todas as informações necessárias do framework enquanto se está utilizando. *

Mark only one oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

19. Caso sua resposta seja inferior a 3, Justifique a sua resposta

20. 07) É fácil realizar atualizações e manutenções em um código Robot já existente. *

Mark only one oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

21. Caso sua resposta seja inferior a 3, Justifique a sua resposta

22. 08) O Robot Framework permite que você insira o que deseja de maneira razoavelmente breve. (Pouco verboso) *

Mark only one oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

23. Caso sua resposta seja inferior a 3, Justifique a sua resposta

24. 09) O Robot Framework necessita de pouco esforço mental em seu uso. *

Mark only one oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

25. Caso sua resposta seja inferior a 3, Justifique a sua resposta

26. 10) A sintaxe do Robot Framework pode levar o usuário a cometer erros facilmente. *

Mark only one oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

27. Caso sua resposta seja inferior a 3, Justifique a sua resposta

28. 12) O código do Robot Framework possui fácil leitura. *

Mark only one oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

29. Caso sua resposta seja inferior a 3, Justifique a sua resposta

30. 13) Ao ler um código escrito usando Robot Framework, é fácil dizer para que serve cada arquivo e funcionalidade no contexto de testes funcionais. *

Mark only one oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

31. Caso sua resposta seja inferior a 3, Justifique a sua resposta

32. 14) Você entendeu para que serve cada parte da sintaxe do Robot Framework com o Selenium ou apenas não inseriu uma informação para que o teste funcionasse. *

Mark only one oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

33. Caso sua resposta seja inferior a 3, Justifique a sua resposta

34. 15) Todas as informações necessárias para o uso da linguagem estão visíveis e de fácil acesso. *

Mark only one oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

35. Caso sua resposta seja inferior a 3, Justifique a sua resposta

36. 16) As dependências se atualizam automaticamente e isso facilita o uso e manutenção do código desenvolvido utilizando o Robot Framework. *

Mark only one oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

37. Caso sua resposta seja inferior a 3, Justifique a sua resposta

38. 17) Os highlights (destaques) em cores diferentes estão coerentes e ajudam no entendimento no uso da framework. *

Mark only one oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

39. Caso sua resposta seja inferior a 3, Justifique a sua resposta

40. 18) O código implementado utilizando o Robot Framework possui fácil manutenção caso seja necessário. *

Mark only one oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

41. Caso sua resposta seja inferior a 3, Justifique a sua resposta

42. 19) O relatório gerado pela Robot auxilia na análise e detecção de bugs. *

Mark only one oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

43. Caso sua resposta seja inferior a 3, Justifique a sua resposta

Teasy Structure

Seção para nos dar um feedback sobre a TeasyStructure

44. 01) A TeasyStructure reduz o tempo necessário para configurar o projeto. *

Mark only one oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

45. 02) A TeasyStructure é fácil de se entender. *

Mark only one oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

46. 03) A TeasyStructure torna fácil a realização de manutenções futuras. *

Mark only one oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

47. 06) Você utilizaria o TeasyStructure para auxiliar na qualidade de suas aplicações? *

Mark only one oval.

- ◯ Sim
- ◯ Não
- ◯ Other: _____

48. Justifique a sua resposta *

_____

_____

_____

_____

_____

Treinamento Robot

O que achou do Treinamento ?

49. 01) O treinamento foi capaz de apresentar de forma clara e objetiva como utilizar o Robot Framework. *

Mark only one oval.

	1	2	3	4	5	
Discordo totalmente	◯	◯	◯	◯	◯	Concordo totalmente

50. 02) Na sua opinião o(s) treinamento(s) podem facilitar o uso do framework ? *

Mark only one oval.

- ◯ Sim
- ◯ Não

51. Justifique sua resposta *

52. 03) Você participaria de novos treinamentos neste modelo ? *

Mark only one oval.

- ◯ Sim
- ◯ Não

53. 04) O treinamento foi cansativo ? *

Mark only one oval.

- Sim
- Não

54. 05) Se tiver algum feedback, ou sugestão insira abaixo, assim consigo lançar treinamentos ainda melhores.

Considerações Finais

Até aqui você ja deve estar um pouco cansado. Mas espero que tenha conseguido aprender um novo conteúdo, você foi e será muito importante para a melhoria deste framework. Creio que alguns vídeos possam ter qualidades diferentes, mas estamos melhorando o treinamento com o feedback de vocês, obrigado.

55. 01) Na sua opnião quais são os pontos positivos do framework Robot ? *

56. 02) Na sua opnião, quais são os pontos negativos do framework Robot ? *

57. 03) Tem alguma sugestão de melhoria ? Se sim, pode descrever neste espaço

58. 04) Você teria interesse em participar de novos estudos nesta área ou nesta estrutura ? *

Mark only one oval.

- Sim
- Não

59. Se sim, qual seu e-mail?

60. Agradecemos a sua participação e dedicamos este espaço para você. Caso tenha alguma sugestão ou informação que não foi abordada nesta avaliação ou no Robot, poderia descreve-lá para que possamos melhorar nosso estudo.

ANEXO B – FORMULÁRIO DE UTILIZADO NO *SURVEY* PARA AVALIAR A *TEASY LANGUAGE*

Teasy Framework seu feedback é importante !

Prezado(a), este questionário destina-se a coletar dados para uma pesquisa na área de Engenharia de Software, conduzida na Universidade Federal do Pampa (UNIPAMPA), com objetivo de avaliar a usabilidade do framework Teasy. A sua participação é de suma importância para que essa linguagem possa ser melhorada. Tenha em vista que os dados informados são para fins de análise e que não é você que está sendo avaliado e sim a facilidade no uso do framework Teasy. As informações obtidas serão confidenciais e asseguramos o sigilo sobre sua participação. Os dados publicados não permitirão a sua identificação. As informações obtidas serão utilizadas de modo único e excluso para os fins específicos deste estudo.

Yury Alencar Lima <yuryalencar19@gmail.com>
Elder de Macedo Rodrigues <eldermr@gmail.com>
Rafael Paes Oliveira <olivrap@gmail.com>
Ildevana Poltronieri Rodrigues <ildevana.rodrigues@acad.pucrs.br>

* Required

1. Email address *

2. Termo de Consentimento *

Check all that apply.

☐ Concordo em participar deste estudo e declaro ter lido os detalhes descritos neste formulário. Eu entendo que sou livre para aceitar ou recusar, e que posso interromper minha participação a qualquer momento sem indicar o motivo. Concordo que os dados coletados serão utilizados para os fins descritos acima. Compreendo as informações apresentadas neste TERMO DE CONSENTIMENTO. Tive a oportunidade de fazer perguntas e todas as minhas perguntas foram respondidas. Receberei uma cópia assinada e datada deste Termo de Consentimento Livre e Esclarecido.

Vamos nos conhecer

Seção para sabermos melhor o seu perfil

3. 01) Você já utilizou uma Linguagem de Domínio Específico (DSL) ? *

   Mark only one oval.

   - ◯ Sim
   - ◯ Não

4. 02) Você já criou alguma Linguagem de Domínio Específico (DSL) ? *

   Mark only one oval.

   - ◯ Sim
   - ◯ Não

5. 03) Quais estudos empíricos já participou ? *

   Check all that apply.

   - ☐ Estudo de Caso
   - ☐ Experimento Controlado
   - ☐ Quase Experimento
   - ☐ Survey (Questionários)
   - ☐ Provas de conceito
   - ☐ Nunca participei de um estudo empírico

6. 04) Qual categoria(s) você pertence atualmente ? *

   Check all that apply.

   - ☐ Academia
   - ☐ Indústria

7. 05) Qual é o seu grau de escolaridade? *

Mark only one oval.

- ◯ Ensino fundamental completo
- ◯ Ensino médio incompleto
- ◯ Ensino médio completo
- ◯ Graduação incompleta
- ◯ Graduação completa
- ◯ Mestrado incompleto
- ◯ Mestrado completo
- ◯ Doutorado incompleto
- ◯ Doutorado completo
- ◯ Other: _____

8. 05) Quanto tempo possui de experiência na Indústria ? *

Mark only one oval.

- ◯ Não possuo
- ◯ Até 1 ano
- ◯ Entre 1 ano e 3 anos
- ◯ Entre 3 anos e 6 anos
- ◯ Entre 6 anos 9 anos
- ◯ acima de 9 anos

9. 06) Quanto tempo possui de experiência com testes de software? *

Mark only one oval.

- ◯ Não possuo
- ◯ Até 1 ano
- ◯ Entre 1 ano e 3 anos
- ◯ Entre 3 anos e 6 anos
- ◯ Entre 6 anos e 9 anos
- ◯ Acima de 9 anos

10. 07) Você já aplicou alguma avaliação heurística? *

Mark only one oval.

- ◯ Sim
- ◯ Não

H1: Visibilidade do status do sistema	A DSL deve sempre manter os usuários informados sobre o que está acontecendo, por meio de feedback apropriado dentro de um prazo razoável.

Para cada questão aplicável você pode detectar um ou mais problemas de usabilidade. Desta forma existe uma pergunta para que liste as ocorrências desses erros e a severidade deles de acordo com uma respectiva classificação. A classificação e um exemplo de descrição estão abaixo.

A seguinte escala de classificação de 0 a 4 pode ser usada para classificar a gravidade dos problemas de usabilidade:		
Severidade	**Tipo**	**Descrição**
0	Não Aplicável	Não concordo que seja um problema de usabilidade
1	Problema cosmético apenas	Não precisa ser corrigido, a menos que haja tempo extra disponível no projeto
2	Pequeno problema de usabilidade	Corrigir isso deve ter baixa prioridade
3	Grande problema de usabilidade	Importante corrigir, por isso deve ser dada alta prioridade
4	Catástrofe de usabilidade	Imperativo corrigir isso antes que o produto possa ser lançado

Exemplo de descrição da ocorrência do erro:
ERRO 1 - 0
ERRO 2 - 4

11. Q01) A Teasy Language textual fornece feedback imediato e adequado sobre seu status para cada ação do usuário? Por exemplo, após uma tarefa de inclusão ou exclusão, o idioma exibe uma mensagem de confirmação? *

Mark only one oval.

- ◯ Aplicável
- ◯ Não aplicável

12. Q01) Se aplicável, descreva como cada erro ocorreu separado por enter. (Caso não seja aplicável insira um ponto final) *

13. Q01) Para cada erro listado acima insira o grau de severidade separados por enter. (Caso não possua nenhum erro acima insira um ponto final) *

14. Q02) Os elementos disponíveis para o usuário executam especificamente apenas um comando? Por exemplo, o botão "desfazer" executa apenas ações desfazer. *

Mark only one oval.

- ◯ Aplicável
- ◯ Não aplicável

15. Q02) Se aplicável, descreva como cada erro ocorreu separado por enter. (Caso não seja aplicável insira um ponto final) *

16. Q02) Para cada erro listado acima insira o grau de severidade separados por enter. (Caso não possua nenhum erro acima insira um ponto final) *

17. Q03) Todas as mensagens de alerta e erro que o Teasy Framework fornece permanecem na tela por tempo suficiente para serem lidas mais de uma vez ou (preferencialmente) até que o usuário opte por fechá-las? *

Mark only one oval.

- ◯ Aplicável
- ◯ Não aplicável

18. Q03) Se aplicável, descreva como cada erro ocorreu separado por enter. (Caso não seja aplicável insira um ponto final) *

19. Q03) Para cada erro listado acima insira o grau de severidade separados por enter. (Caso não possua nenhum erro acima insira um ponto final) *

20. Q04) A linguagem Teasy está estruturada de forma acoplada? Por exemplo, se o usuário altera alguma informação como nome de classe, esta alteração é refletida nos locais que utilizam esta informação. *

Mark only one oval.

- ◯ Aplicável
- ◯ Não aplicável

21. Q04) Se aplicável, descreva como cada erro ocorreu separado por enter. (Caso não seja aplicável insira um ponto final) *

22. Q04) Para cada erro listado acima insira o grau de severidade separados por enter. (Caso não possua nenhum erro acima insira um ponto final) *

H2: Combine o idioma com o domínio do mundo real	A DSL deve representar o idioma do usuário com palavras, frases e conceitos familiares ao usuário, em vez de orientada ao sistema. Deve seguir as convenções do mundo real, tornando as informações naturais e lógicas.

Para cada questão aplicável você pode detectar um ou mais problemas de usabilidade. Desta forma existe uma pergunta para que liste as ocorrências desses erros e a severidade deles de acordo com uma respectiva classificação. A classificação e um exemplo de descrição estão abaixo.

A seguinte escala de classificação de 0 a 4 pode ser usada para classificar a gravidade dos problemas de usabilidade:		
Severidade	**Tipo**	**Descrição**
0	Não Aplicável	Não concordo que seja um problema de usabilidade
1	Problema cosmético apenas	Não precisa ser corrigido, a menos que haja tempo extra disponível no projeto
2	Pequeno problema de usabilidade	Corrigir isso deve ter baixa prioridade
3	Grande problema de usabilidade	Importante corrigir, por isso deve ser dada alta prioridade
4	Catástrofe de usabilidade	Imperativo corrigir isso antes que o produto possa ser lançado

Exemplo de descrição da ocorrência do erro:
ERRO 1 - 0
ERRO 2 - 4

23. Q01) As palavras-chave reservadas que compõem a Linguagem Teasy são usadas no domínio do mundo real do usuário? Por exemplo, uma linguagem desenvolvida para um domínio relacionado a bibliotecas precisa ter palavras-chave relacionadas ao domínio da biblioteca. *

Mark only one oval.

- ◯ Aplicável
- ◯ Não aplicável

24. Q01) Se aplicável, descreva como cada erro ocorreu separado por enter. (Caso não seja aplicável insira um ponto final) *

25. Q01) Para cada erro listado acima insira o grau de severidade separados por enter. (Caso não possua nenhum erro acima insira um ponto final) *

26. Q02) Os elementos que compõem a linguagem Teasy são uma boa representação do domínio de Testes de Software? *

Mark only one oval.

- ◯ Aplicável
- ◯ Não aplicável

27. Q02) Se aplicável, descreva como cada erro ocorreu separado por enter. (Caso não seja aplicável insira um ponto final) *

28. Q02) Para cada erro listado acima insira o grau de severidade separados por enter. (Caso não possua nenhum erro acima insira um ponto final) *

29. Q03) Os elementos que compõem a linguagem Teasy atendem ao objetivo de Testes de Software? O usuário entende o objetivo da linguagem ao interagir com seus elementos. *

Mark only one oval.

- ◯ Aplicável
- ◯ Não aplicável

30. Q03) Se aplicável, descreva como cada erro ocorreu separado por enter. (Caso não seja aplicável insira um ponto final) *

31. Q03) Para cada erro listado acima insira o grau de severidade separados por enter. (Caso não possua nenhum erro acima insira um ponto final) *

H3: Controle e liberdade do usuário	Os usuários geralmente escolhem as funções DSL por engano e precisarão de uma "saída de emergência" claramente marcada para deixar o estado indesejado sem ter que se submeter a um diálogo extenso. Desfazer e refazer o suporte.

Para cada questão aplicável você pode detectar um ou mais problemas de usabilidade. Desta forma existe uma pergunta para que liste as ocorrências desses erros e a severidade deles de acordo com uma respectiva classificação. A classificação e um exemplo de descrição estão abaixo.

A seguinte escala de classificação de 0 a 4 pode ser usada para classificar a gravidade dos problemas de usabilidade:		
Severidade	**Tipo**	**Descrição**
0	Não Aplicável	Não concordo que seja um problema de usabilidade
1	Problema cosmético apenas	Não precisa ser corrigido, a menos que haja tempo extra disponível no projeto
2	Pequeno problema de usabilidade	Corrigir isso deve ter baixa prioridade
3	Grande problema de usabilidade	Importante corrigir, por isso deve ser dada alta prioridade
4	Catástrofe de usabilidade	Imperativo corrigir isso antes que o produto possa ser lançado

Exemplo de descrição da ocorrência do erro:
ERRO 1 - 0
ERRO 2 - 4

32. Q01) O DSL textual tem redundância nos comandos desfazer/refazer (por exemplo, atalhos (CTRL+Z) entre outros)? *

Mark only one oval.

- ◯ Aplicável
- ◯ Não aplicável

33. Q01) Se aplicável, descreva como cada erro ocorreu separado por enter. (Caso não seja aplicável insira um ponto final) *

34. Q01) Para cada erro listado acima insira o grau de severidade separados por enter. (Caso não possua nenhum erro acima insira um ponto final) *

H4: Consistência e padrões

Os usuários do domínio não devem se perguntar se palavras, situações ou ações diferentes significam a mesma coisa.

Para cada questão aplicável você pode detectar um ou mais problemas de usabilidade. Desta forma existe uma pergunta para que liste as ocorrências desses erros e a severidade deles de acordo com uma respectiva classificação. A classificação e um exemplo de descrição estão abaixo.

A seguinte escala de classificação de 0 a 4 pode ser usada para classificar a gravidade dos problemas de usabilidade:		
Severidade	**Tipo**	**Descrição**
0	Não Aplicável	Não concordo que seja um problema de usabilidade
1	Problema cosmético apenas	Não precisa ser corrigido, a menos que haja tempo extra disponível no projeto
2	Pequeno problema de usabilidade	Corrigir isso deve ter baixa prioridade
3	Grande problema de usabilidade	Importante corrigir, por isso deve ser dada alta prioridade
4	Catástrofe de usabilidade	Imperativo corrigir isso antes que o produto possa ser lançado

Exemplo de descrição da ocorrência do erro:
ERRO 1 - 0
ERRO 2 - 4

35. Q01) Todos os elementos textuais estão aparentemente conectados uns aos outros? Por exemplo, todas as palavras-chave reservadas são mostradas na mesma cor? *

Mark only one oval.

- ◯ Aplicável
- ◯ Não aplicável

36. Q01) Se aplicável, descreva como cada erro ocorreu separado por enter. (Caso não seja aplicável insira um ponto final) *

37. Q01) Para cada erro listado acima insira o grau de severidade separados por enter. (Caso não possua nenhum erro acima insira um ponto final) *

38. Q02) Todos os elementos textuais estão conectados uns aos outros de acordo com seus nomes? Por exemplo, todas as palavras-chave reservadas são escritas no mesmo estilo, usando apenas camelCase, apenas sublinhados (_) ou somente com espaços para separar palavras? *

Mark only one oval.

- ◯ Aplicável
- ◯ Não aplicável

39. Q02) Se aplicável, descreva como cada erro ocorreu separado por enter. (Caso não seja aplicável insira um ponto final) *

40. Q02) Para cada erro listado acima insira o grau de severidade separados por enter. (Caso não possua nenhum erro acima insira um ponto final) *

41. Q03) No caso de usar abreviações, todas as abreviações são consistentes ao longo do uso do DSL? *

Mark only one oval.

- ◯ Aplicável
- ◯ Não aplicável

42. Q03) Se aplicável, descreva como cada erro ocorreu separado por enter. (Caso não seja aplicável insira um ponto final) *

43. Q03) Para cada erro listado acima insira o grau de severidade separados por enter. (Caso não possua nenhum erro acima insira um ponto final) *

H5: Prevenção de erros

Ainda melhor do que boas mensagens de erro é um design cuidadoso que evita a ocorrência de um problema. Elimine as condições propensas a erros ou verifique-as e apresente aos usuários uma opção de confirmação antes que eles se comprometam com a ação. (Ex: Caixa de confirmação)

Para cada questão aplicável você pode detectar um ou mais problemas de usabilidade. Desta forma existe uma pergunta para que liste as ocorrências desses erros e a severidade deles de acordo com uma respectiva classificação. A classificação e um exemplo de descrição estão abaixo.

A seguinte escala de classificação de 0 a 4 pode ser usada para classificar a gravidade dos problemas de usabilidade:		
Severidade	**Tipo**	**Descrição**
0	Não Aplicável	Não concordo que seja um problema de usabilidade
1	Problema cosmético apenas	Não precisa ser corrigido, a menos que haja tempo extra disponível no projeto
2	Pequeno problema de usabilidade	Corrigir isso deve ter baixa prioridade
3	Grande problema de usabilidade	Importante corrigir, por isso deve ser dada alta prioridade
4	Catástrofe de usabilidade	Imperativo corrigir isso antes que o produto possa ser lançado

Exemplo de descrição da ocorrência do erro:
ERRO 1 - 0
ERRO 2 - 4

44. Q01) A linguagem Teasy requere alguma confirmação quando o usuário realiza ações consideradas perigosas (deletar coisas, etc)? *

Mark only one oval.

- Aplicável
- Não aplicável

45. Q01) Se aplicável, descreva como cada erro ocorreu separado por enter. (Caso não seja aplicável insira um ponto final) *

46. Q01) Para cada erro listado acima insira o grau de severidade separados por enter. (Caso não possua nenhum erro acima insira um ponto final) *

47. Q02) A linguagem Teasy fornece um mecanismo para informar as restrições de uso em caso de, por exemplo, erros de digitação ou cometer algum tipo de erro? *

Mark only one oval.

- ◯ Aplicável
- ◯ Não aplicável

48. Q02) Se aplicável, descreva como cada erro ocorreu separado por enter. (Caso não seja aplicável insira um ponto final) *

49. Q02) Para cada erro listado acima insira o grau de severidade separados por enter. (Caso não possua nenhum erro acima insira um ponto final) *

50. Q03) A linguagem Teasy prevê decisões erradas feitas pelos usuários e os avisa? Por exemplo, quando o usuário insere uma informação clica em um botão para fechar todos os arquivos e as tarefas realizadas pelo usuário não foram salvas, a DSL avisa o usuário que o aplicativo será fechado e nada será salvo? *

Mark only one oval.

- ◯ Aplicável
- ◯ Não aplicável

51. Q03) Se aplicável, descreva como cada erro ocorreu separado por enter. (Caso não seja aplicável insira um ponto final) *

52. Q03) Para cada erro listado acima insira o grau de severidade separados por enter. (Caso não possua nenhum erro acima insira um ponto final) *

H6: Reconhecimento em vez de lembrar	Minimize a carga de memória do usuário, tornando objetos, ações e opções visíveis. O usuário não deve ter que se lembrar de informações de uma parte do sistema para outra. As instruções de uso do sistema devem ser visíveis ou facilmente recuperáveis quando apropriado.

Para cada questão aplicável você pode detectar um ou mais problemas de usabilidade. Desta forma existe uma pergunta para que liste as ocorrências desses erros e a severidade deles de acordo com uma respectiva classificação. A classificação e um exemplo de descrição estão abaixo.

A seguinte escala de classificação de 0 a 4 pode ser usada para classificar a gravidade dos problemas de usabilidade:		
Severidade	**Tipo**	**Descrição**
0	Não Aplicável	Não concordo que seja um problema de usabilidade
1	Problema cosmético apenas	Não precisa ser corrigido, a menos que haja tempo extra disponível no projeto
2	Pequeno problema de usabilidade	Corrigir isso deve ter baixa prioridade
3	Grande problema de usabilidade	Importante corrigir, por isso deve ser dada alta prioridade
4	Catástrofe de usabilidade	Imperativo corrigir isso antes que o produto possa ser lançado

Exemplo de descrição da ocorrência do erro:
ERRO 1 - 0
ERRO 2 - 4

53. Q01) As palavras-chave reservadas têm um nome fácil e significativo para que o usuário possa se lembrar delas com rapidez? Nesse caso, palavras-chave são palavras relacionadas ao domínio do Testes de Software. *

Mark only one oval.

- ◯ Aplicável
- ◯ Não aplicável

54. Q01) Se aplicável, descreva como cada erro ocorreu separado por enter. (Caso não seja aplicável insira um ponto final) *

_____

_____

_____

_____

_____

55. Q01) Para cada erro listado acima insira o grau de severidade separados por enter. (Caso não possua nenhum erro acima insira um ponto final) *

56. Q02) A linguagem Teasy é composta por palavras fáceis de serem vistas na tela (com cores de bom contraste, facilmente identificadas pelo usuário final)? Por exemplo, em outras linguagens os comandos como “if / else” são mostrados em uma cor diferente do resto do código. *

Mark only one oval.

- ◯ Aplicável
- ◯ Não aplicável

57. Q02) Se aplicável, descreva como cada erro ocorreu separado por enter. (Caso não seja aplicável insira um ponto final) *

58. Q02) Para cada erro listado acima insira o grau de severidade separados por enter. (Caso não possua nenhum erro acima insira um ponto final) *

H7: Flexibilidade e eficiência de uso

Aceleradores - não vistos pelo usuário novato - muitas vezes podem acelerar a interação do usuário experiente, de modo que a linguagem pode atender tanto a usuários inexperientes quanto experientes. Permita que os usuários personalizem ações frequentes.

Para cada questão aplicável você pode detectar um ou mais problemas de usabilidade. Desta forma existe uma pergunta para que liste as ocorrências desses erros e a severidade deles de acordo com uma respectiva classificação. A classificação e um exemplo de descrição estão abaixo.

A seguinte escala de classificação de 0 a 4 pode ser usada para classificar a gravidade dos problemas de usabilidade:		
Severidade	**Tipo**	**Descrição**
0	Não Aplicável	Não concordo que seja um problema de usabilidade
1	Problema cosmético apenas	Não precisa ser corrigido, a menos que haja tempo extra disponível no projeto
2	Pequeno problema de usabilidade	Corrigir isso deve ter baixa prioridade
3	Grande problema de usabilidade	Importante corrigir, por isso deve ser dada alta prioridade
4	Catástrofe de usabilidade	Imperativo corrigir isso antes que o produto possa ser lançado

Exemplo de descrição da ocorrência do erro:
ERRO 1 - 0
ERRO 2 - 4

59. Q01) A linguagem Teasy possui atalhos para usuários experientes fazerem o uso da DSL de forma rápida? Por exemplo, fornecer atalhos ou funções para escrever estruturas de código básicas. *

Mark only one oval.

- ◯ Aplicável
- ◯ Não aplicável

60. Q01) Se aplicável, descreva como cada erro ocorreu separado por enter. (Caso não seja aplicável insira um ponto final) *

_____

_____

_____

_____

_____

61. Q01) Para cada erro listado acima insira o grau de severidade separados por enter. (Caso não possua nenhum erro acima insira um ponto final) *

62. Q02) A linguagem Teasy fornece todos os elementos para completar a tarefa desejada? *

Mark only one oval.

- ◯ Aplicável
- ◯ Não aplicável

63. Q02) Se aplicável, descreva como cada erro ocorreu separado por enter. (Caso não seja aplicável insira um ponto final) *

64. Q02) Para cada erro listado acima insira o grau de severidade separados por enter. (Caso não possua nenhum erro acima insira um ponto final) *

65. Q03) As tarefas comuns são fáceis e rápidas de serem executadas? Por exemplo, abrir um arquivo ou excluir um elemento. *

Mark only one oval.

- ◯ Aplicável
- ◯ Não aplicável

66. Q03) Se aplicável, descreva como cada erro ocorreu separado por enter. (Caso não seja aplicável insira um ponto final) *

67. Q03) Para cada erro listado acima insira o grau de severidade separados por enter. (Caso não possua nenhum erro acima insira um ponto final) *

H8: Design estético e minimalista

Os diálogos não devem conter informações irrelevantes ou raramente necessárias. Cada unidade extra de informação em um diálogo compete com as unidades relevantes de informação e diminui sua visibilidade relativa.

Para cada questão aplicável você pode detectar um ou mais problemas de usabilidade. Desta forma existe uma pergunta para que liste as ocorrências desses erros e a severidade deles de acordo com uma respectiva classificação. A classificação e um exemplo de descrição estão abaixo.

A seguinte escala de classificação de 0 a 4 pode ser usada para classificar a gravidade dos problemas de usabilidade:		
Severidade	**Tipo**	**Descrição**
0	Não Aplicável	Não concordo que seja um problema de usabilidade
1	Problema cosmético apenas	Não precisa ser corrigido, a menos que haja tempo extra disponível no projeto
2	Pequeno problema de usabilidade	Corrigir isso deve ter baixa prioridade
3	Grande problema de usabilidade	Importante corrigir, por isso deve ser dada alta prioridade
4	Catástrofe de usabilidade	Imperativo corrigir isso antes que o produto possa ser lançado

Exemplo de descrição da ocorrência do erro:
ERRO 1 - 0
ERRO 2 - 4

68. Q01) As cores do DSL ajudam a entender o estado atual do sistema, mas não são as únicas fontes de informação para entender um estado? *

Mark only one oval.

- ◯ Aplicável
- ◯ Não aplicável

69. Q01) Se aplicável, descreva como cada erro ocorreu separado por enter. (Caso não seja aplicável insira um ponto final) *

70. Q01) Para cada erro listado acima insira o grau de severidade separados por enter. (Caso não possua nenhum erro acima insira um ponto final) *

71. Q02) No caso de mostrar uma caixa de diálogo, esta caixa de diálogo contém as informações mostradas em um texto curto? A quantidade de linhas de um texto curto pode variar de acordo com o diálogo e as informações expostas, mas é aconselhável que um diálogo não tenha mais de 3 linhas de aviso. *

Mark only one oval.

- ◯ Aplicável
- ◯ Não aplicável
- ◯ Other: _____

72. Q02) Se aplicável, descreva como cada erro ocorreu separado por enter. (Caso não seja aplicável insira um ponto final) *

73. Q02) Para cada erro listado acima insira o grau de severidade separados por enter. (Caso não possua nenhum erro acima insira um ponto final) *

H9: Ajude os usuários a reconhecer, diagnosticar e recuperar de erros

As mensagens de erro devem ser expressas em linguagem simples (sem códigos), indicar precisamente o problema e sugerir uma solução de forma construtiva.

Para cada questão aplicável você pode detectar um ou mais problemas de usabilidade. Desta forma existe uma pergunta para que liste as ocorrências desses erros e a severidade deles de acordo com uma respectiva classificação. A classificação e um exemplo de descrição estão abaixo.

A seguinte escala de classificação de 0 a 4 pode ser usada para classificar a gravidade dos problemas de usabilidade:		
Severidade	**Tipo**	**Descrição**
0	Não Aplicável	Não concordo que seja um problema de usabilidade
1	Problema cosmético apenas	Não precisa ser corrigido, a menos que haja tempo extra disponível no projeto
2	Pequeno problema de usabilidade	Corrigir isso deve ter baixa prioridade
3	Grande problema de usabilidade	Importante corrigir, por isso deve ser dada alta prioridade
4	Catástrofe de usabilidade	Imperativo corrigir isso antes que o produto possa ser lançado

Exemplo de descrição da ocorrência do erro:
ERRO 1 - 0
ERRO 2 - 4

74. Q01) Em caso de erro, a DSL mostra uma mensagem de erro apropriada onde o usuário, após a leitura, sabe como se recuperar do erro ou onde procurar uma solução? *

Mark only one oval.

- Aplicável
- Não aplicável

75. Q01) Se aplicável, descreva como cada erro ocorreu separado por enter. (Caso não seja aplicável insira um ponto final) *

76. Q01) Para cada erro listado acima insira o grau de severidade separados por enter. (Caso não possua nenhum erro acima insira um ponto final) *

77. Q02) No caso de mostrar uma caixa de diálogo (quando o usuário comete um erro, por exemplo), esta caixa de diálogo mostra de forma clara os possíveis motivos pelos quais o erro foi cometido? *

Mark only one oval.

- ◯ Aplicável
- ◯ Não aplicável
- ◯ Other: _____

78. Q02) Se aplicável, descreva como cada erro ocorreu separado por enter. (Caso não seja aplicável insira um ponto final) *

79. Q02)Para cada erro listado acima insira o grau de severidade separados por enter. (Caso não possua nenhum erro acima insira um ponto final) *

80. Q03) A linguagem Teasy tem um mecanismo que destaca erros de sintaxe no texto ou palavras reservadas? *

Mark only one oval.

- ◯ Aplicável
- ◯ Não aplicável
- ◯ Other: _____

81. Q03) Se aplicável, descreva como cada erro ocorreu separado por enter. (Caso não seja aplicável insira um ponto final) *

82. Q03) Para cada erro listado acima insira o grau de severidade separados por enter. (Caso não possua nenhum erro acima insira um ponto final) *

83. Q04) A linguagem Teasy possui mecanismos para destacar erros de aninhamento ou organização de palavras reservadas que tornam o modelo inconsistente? *

Mark only one oval.

- ◯ Aplicável
- ◯ Não aplicável
- ◯ Other: _____

84. Q04) Se aplicável, descreva como cada erro ocorreu separado por enter. (Caso não seja aplicável insira um ponto final) *

_____

_____

_____

_____

_____

85. Q04) Para cada erro listado acima insira o grau de severidade separados por enter. (Caso não possua nenhum erro acima insira um ponto final) *

_____

_____

_____

_____

_____

86. Q05) A linguagem Teasy possui mecanismos que marcam erros de conexão entre elementos que podem tornar o modelo inconsistente? *

Mark only one oval.

- ◯ Aplicável
- ◯ Não aplicável
- ◯ Other: _____

87. Q05) Se aplicável, descreva como cada erro ocorreu separado por enter. (Caso não seja aplicável insira um ponto final) *

_____

_____

_____

_____

_____

88. Q05) Para cada erro listado acima insira o grau de severidade separados por enter. (Caso não possua nenhum erro acima insira um ponto final) *

_____

_____

_____

_____

_____

H10: Ajuda e documentação

Mesmo que seja melhor se o sistema puder ser usado sem documentação, pode ser necessário fornecer ajuda e documentação. Essas informações devem ser fáceis de pesquisar, focadas na tarefa do usuário, listar etapas concretas a serem realizadas e não ser muito extensas.

Para cada questão aplicável você pode detectar um ou mais problemas de usabilidade. Desta forma existe uma pergunta para que liste as ocorrências desses erros e a severidade deles de acordo com uma respectiva classificação. A classificação e um exemplo de descrição estão abaixo.

A seguinte escala de classificação de 0 a 4 pode ser usada para classificar a gravidade dos problemas de usabilidade:		
Severidade	**Tipo**	**Descrição**
0	Não Aplicável	Não concordo que seja um problema de usabilidade
1	Problema cosmético apenas	Não precisa ser corrigido, a menos que haja tempo extra disponível no projeto
2	Pequeno problema de usabilidade	Corrigir isso deve ter baixa prioridade
3	Grande problema de usabilidade	Importante corrigir, por isso deve ser dada alta prioridade
4	Catástrofe de usabilidade	Imperativo corrigir isso antes que o produto possa ser lançado

Exemplo de descrição da ocorrência do erro:
ERRO 1 - 0
ERRO 2 - 4

89. Q01) A linguagem Teasy tem um tutorial para fornecer o objetivo do DSL, o uso da sintaxe e seu significado, bem como fornecer exemplos de uso da linguagem? *

Mark only one oval.

- ◯ Aplicável
- ◯ Não aplicável

90. Q01) Se aplicável, descreva como cada erro ocorreu separado por enter. (Caso não seja aplicável insira um ponto final) *

91. Q01) Para cada erro listado acima insira o grau de severidade separados por enter. (Caso não possua nenhum erro acima insira um ponto final) *

92. Q02) Todos os comandos incluídos na DSL textual estão descritos de forma clara no suporte de documentação? *

Mark only one oval.

- ◯ Aplicável
- ◯ Não aplicável
- ◯ Other: _____

93. Q02) Se aplicável, descreva como cada erro ocorreu separado por enter. (Caso não seja aplicável insira um ponto final) *

94. Q02) Para cada erro listado acima insira o grau de severidade separados por enter. (Caso não possua nenhum erro acima insira um ponto final) *

95. Q03) A linguagem Teasy tem ajuda contextual para cada elemento? *

Mark only one oval.

- ◯ Aplicável
- ◯ Não aplicável
- ◯ Other: _____

96. Q03) Se aplicável, descreva como cada erro ocorreu separado por enter. (Caso não seja aplicável insira um ponto final) *

_____

_____

_____

_____

_____

97. Q03) Para cada erro listado acima insira o grau de severidade separados por enter. (Caso não possua nenhum erro acima insira um ponto final) *

_____

_____

_____

_____

_____

Considerações finais

Até aqui você já deve estar um pouco cansado. Mas você foi e será muito importante para a melhoria de nosso framework. Creio que alguns vídeos possam ter qualidades diferentes, mas estamos melhorando nosso treinamento a partir do seu feedback, obrigado.

98. 01) Na sua opnião quais são os pontos positivos do framework Teasy ? *

99. 02) Na sua opnião, quais são os pontos negativos do framework Teasy atualmente ?

100. 03) Tem alguma sugestão de melhoria ? Se sim, ficarei feliz em implementar, basta descrever abaixo.

101. 04) Você teria interesse de participar de novos estudos nesta área ou nesta formato ?

Mark only one oval.

- Sim
- Não

102. Se sim, qual é o seu e-mail ?

103. Este espaço é seu, caso tenha algum feedback, sugestão ou algo não abordado até aqui. Basta descrever aqui, e desde já agradeço a sua participação neste estudo.

ÍNDICE