UNIVERSIDADE FEDERAL DO PAMPA

Yury Alencar Lima

Teasy Mobile Language: Uma linguagem específica de domínio para testes funcionais em aplicativos móveis

Alegrete
2023

Yury Alencar Lima

Teasy Language Mobile: Uma linguagem específica de domínio para testes funcionais em aplicativos móveis

Dissertação apresentada ao Programa de Pós-graduação em Engenharia de Software da Universidade Federal do Pampa, como requisito parcial para obtenção do Título de Mestre em Engenharia de Software.

Dissertação defendida e aprovada em: 07/12/2023.

Banca examinadora:

_____

Prof. Dr. Elder de Macedo Rodrigues

Orientador

Unipampa

_____

Prof. Dr. Fábio Paulo Basso

Unipampa

_____

Prof. Dr. Maicon Bernardino da Silveira

Unipampa

_____

Prof. Dr. Wesley Assunção

North Carolina State University

Assinado eletronicamente por **ELDER DE MACEDO RODRIGUES**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em 07/12/2023, às 10:24, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **Wesley Klewerton Guez Assunção**, **Usuário Externo**, em 07/12/2023, às 10:26, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **FABIO PAULO BASSO**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em 07/12/2023, às 10:26, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **MAICON BERNARDINO DA SILVEIRA**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em 07/12/2023, às 10:26, 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 **1319851** e o código CRC **456B9E12**.

Yury Alencar Lima

Teasy Mobile Language: Uma linguagem específica de domínio para testes funcionais em aplicativos móveis

Dissertação de Mestrado apresentada como requisito parcial para obtenção do título de Mestre em Engenharia de Software pela Universidade Federal do Pampa.

Supervisor: Prof. PhD. Elder de Macedo Rodrigues

Alegrete
2023

Ficha catalográfica elaborada automaticamente com os dados fornecidos
pelo(a) autor(a) através do Módulo de Biblioteca do
Sistema GURI (Gestão Unificada de Recursos Institucionais) .

L732t Lima, Yury Alencar
Teasy Mobile Language: Uma linguagem específica de domínio para testes funcionais em aplicativos móveis / Yury Alencar Lima.
101 p.

Dissertação(Mestrado)-- Universidade Federal do Pampa, MESTRADO EM ENGENHARIA DE SOFTWARE, 2023.
"Orientação: Elder de Macedo Rodrigues".

1. Engenharia de Software. 2. Linguagem específica de domínio. 3. Teste em aplicativos móveis. 4. Testes automatizados. 5. Teste funcional. I. Título.

ABSTRACT

This dissertation proposes the Teasy Mobile Language, an extension of the Teasy Framework, focusing on test automation for mobile devices. The fundamental objectives of this research are to simplify and optimize the automation process in mobile applications, filling a gap identified through systematic mapping. This systematic mapping of the literature highlighted the demand for agile and efficient solutions for mobile testing. The study takes a comprehensive approach, starting with systematic mapping to consolidate existing knowledge and identify areas of research in need of attention. Then, a case study is carried out in a real industrial environment, where the Teasy Mobile Language is empirically evaluated. The results of this case study highlight the effectiveness of the language, emphasizing benefits such as clear syntax, speed in creating test scenarios and ease of learning. Additionally, specific areas that can be improved are identified, providing valuable guidance for future developments. Final considerations emphasize the complementary role of Teasy Mobile Language in the mobile test automation ecosystem. They highlight its effective integration with the Teasy Framework, providing a cohesive experience for testers familiar with the web version of the language. Furthermore, they highlight the ongoing commitment to development, encouraged by the contribution of the community and the needs of the industry. Finally, Teasy Mobile Language emerges as a promising alternative for test automation on mobile devices, offering agility, simplicity, and effective integration in an increasingly mobile-centric scenario. Its continuous development, marked by an open source and collaborative character, promises continuous improvements to meet the growing demands of the mobile software development industry.

Keywords: Functional testing, Domain-specific language, Testing automation, Software quality, Mobile applications, Software engineering.

RESUMO

Esta dissertação propõe a Teasy Mobile Language, uma extensão do Teasy Framework, com foco na automação de testes para dispositivos móveis. Os objetivos fundamentais desta pesquisa são simplificar e otimizar o processo de automação em aplicativos móveis, preenchendo uma lacuna identificada por meio de um mapeamento sistemático. Esse levantamento evidenciou a demanda por soluções ágeis e eficientes para testes móveis. O estudo adota uma abordagem abrangente, começando com um mapeamento sistemático para consolidar o conhecimento existente e identificar áreas de pesquisa carentes de atenção. Em seguida, realiza-se um estudo de caso em um ambiente industrial real, onde a Teasy Mobile Language é empiricamente avaliada. Os resultados deste estudo de caso destacam a eficácia da linguagem, enfatizando benefícios como sintaxe clara, rapidez na criação de cenários de teste e facilidade de aprendizado. Além disso, são identificadas áreas específicas que podem ser aprimoradas, proporcionando direcionamentos valiosos para futuros desenvolvimentos. As considerações finais enfatizam o papel complementar da Teasy Mobile Language no ecossistema de automação de testes móveis. Destacam sua integração efetiva com o Teasy Framework, proporcionando uma experiência coesa para testadores familiarizados com a versão web da linguagem. Além disso, ressaltam o compromisso contínuo com o desenvolvimento, incentivado pela contribuição da comunidade e pelas necessidades da indústria. Por fim, a Teasy Mobile Language emerge como uma alternativa promissora para a automação de testes em dispositivos móveis, oferecendo agilidade, simplicidade e integração eficaz em um cenário cada vez mais centrado em dispositivos móveis. Seu desenvolvimento contínuo, marcado pelo caráter de código aberto e colaborativo, promete melhorias contínuas para atender às crescentes demandas da indústria de desenvolvimento de software móvel.

Palavras-chave: Testes funcionais, Linguagem específica de domínio, Automação de testes, Qualidade de Software, Aplicativos móveis, Engenharia de Software.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

CE Critério de Exclusão

CI Critério de Inclusão

CQ Critério de Qualidade

ID Identificador

QP Questão de Pesquisa

R Requisito

LISTA DE SIGLAS

BDD Desenvolvimento direcionado a comportamento

BFS Breadth-first search

BPMN Notação de modelagem de processos de negócio

CIL Linguagem Intermediária Comum

DAG Grafo acíclico dirigido

DFS Depth-first search

DSL *Domain-Specific Language*

EMF Eclipse Modeling Framework

FSM Máquina de Estados Finitos

GMF Graphical Modeling Framework

GPL Linguagem de Propósito Geral

GUI Interface Gráfica de Usuário

HTML HyperText Markup Language

IDE Integrated Development Environment

JSON *JavaScript Object Notation*

LW *Language Workbench*

MBT *Model-based Testing*

MDP Markov Decision Process

MPS *Meta Programming System*

OCL *Object Constraint Language*

PICOC Population Intervention Comparison Outcome and Context

SMS Mapeamento Sistemático de Literatura

SUT Sistema Sob Teste

TGF Trivial Graph Format

UML Linguagem de modelagem Unificada

UTP UML Testing Profile

XML Extensible Markup Language

SUMÁRIO

1 INTRODUÇÃO

A necessidade por *software* continua a aumentar em virtude da intensa competição no mercado, com as novas *startups* cada vez mais concentradas na entrega ágil de seus produtos (SOUZA et al., 2017). A urgência em garantir uma entrega rápida pode influenciar diretamente na qualidade do produto, especialmente porque os prazos curtos frequentemente impactam a fase de verificação do *software*. A análise de um produto de *software* é comumente conduzida por meio de testes de software, os quais têm a capacidade de avaliar diversos elementos do sistema, tais como segurança, desempenho, funcionalidade, entre outros (DELAMARO; JINO; MALDONADO, 2013). No contexto desses elementos, a funcionalidade desempenha um papel crucial na verificação de se o sistema está realizando as tarefas conforme especificado pelo cliente, sendo frequentemente o aspecto mais examinado em um produto de *software* (DELAMARO; JINO; MALDONADO, 2013).

A análise das funcionalidades de um sistema é atribuição do teste funcional, podendo ser conduzido manualmente ou por meio de automação. A automação é considerada a opção mais vantajosa, uma vez que, além de viabilizar testes de regressão, assegura a execução regular e consistente dos testes, facilitando a detecção rápida de erros (MOREIRA et al., 2017a). No entanto, a elaboração dos *scripts* automatizados é manual e, sobretudo, a manutenção desses *scripts* é onerosa. Isso gera um desafio nos processos de desenvolvimento ágil, uma vez que as adaptações frequentes podem impactar nos testes de regressão, os quais deveriam estar constantemente disponíveis (TÖRSEL, 2013a).

A *Model-based Testing* (MBT) pode surgir como uma alternativa para amenizar os impactos negativos em questão, uma vez que representa uma abordagem para a geração automatizada de testes com base em modelos predefinidos, otimizando a eficiência no desenvolvimento dos casos de teste. Dessa forma, ao modificar apenas o modelo, é possível regerar todos os casos de teste, aprimorando a facilidade de manutenção desses casos (UTTING; LEGEARD, 2010). Uma opção para empregar a abordagem MBT é por meio de uma *Domain-Specific Language* (DSL). As DSLs são linguagens desenvolvidas com o propósito de representar um domínio específico, ampliando a produtividade e a participação dos envolvidos, podendo ou não permitir a geração de artefatos, dependendo de sua concepção para esse propósito (FOWLER, 2010).

Considerando essa perspectiva, o propósito deste estudo é apresentar uma DSL para testes funcionais voltada para aplicativos móveis, incorporando a geração de *scripts* de testes automatizados em uma estrutura escalável, visando simplificar a manutenção do código de testes. Além disso, esta pesquisa representa uma extensão do Teasy Framework (LIMA, 2021), aprimorando a proposta da linguagem para gerar testes, abrangendo tanto aplicações web quanto móveis.

1.1 Motivação

No ambiente altamente competitivo do desenvolvimento de *software* e nas abordagens ágeis, as empresas priorizam a fase de implementação e o engajamento direto com os clientes, conforme salientado por (SOUZA et al., 2017). No entanto, em algumas circunstâncias, pode ser inviável manter uma interação constante para realizar as validações necessárias durante o processo de desenvolvimento. Portanto, a qualidade do produto em tais cenários pode ser prejudicada, uma vez que o tempo alocado para os testes geralmente é limitado.

Com a crescente demanda do mercado por aplicativos móveis, as empresas buscam manter o foco na produção sem comprometer a qualidade em uma variedade de cenários. Contudo, para alcançar esse equilíbrio, há uma necessidade premente de métodos simplificados para realizar verificações nos produtos de *software*, minimizando o esforço exigido e reduzindo o tempo necessário. Diante dessa demanda, surge a necessidade de encontrar uma maneira simplificada para que as empresas realizem verificações eficientes em seus produtos de *software*, mantendo o foco no desenvolvimento sem comprometer a qualidade. É nesse contexto que a Teasy foi idealizada, buscando proporcionar uma solução. O seu enfoque está na geração de casos e *scripts* de testes automatizados para o sistema, utilizando uma sintaxe que simplifica o processo. A Teasy, portanto, visa atender à demanda por verificações mais eficazes, exigindo menor esforço e reduzindo o tempo necessário para garantir a qualidade do *software* desenvolvido.

1.2 Objetivo

O propósito principal da Teasy Mobile Language, como uma extensão do Teasy Framework (LIMA, 2021), é "Elevar a qualidade de um produto de *software* demandando menos esforço e tempo". Além desse objetivo geral, este trabalho têm mais cinco objetivos específicos.

Objetivo Específico 1: Atualizar a análise dos benefícios, desafios e tendências entre as DSLs já existentes, conforme apresentado em estudo anterior. Isso visa viabilizar a criação de uma solução que atenda às demandas contemporâneas tanto da indústria quanto da academia.

Objetivo Específico 2: Facilitar o reúso de componentes dentro da linguagem, diminuindo o esforço e tempo requeridos para descrever o sistema a ser testado.

Objetivo Específico 3: Automatizar a geração de *scripts* de teste com base na descrição do sistema por meio de um gerador dedicado a aplicativos móveis.

Objetivo Específico 4: Produzir *scripts* de teste em uma estrutura escalável e altamente manutenível para os artefatos de teste, eliminando a necessidade de cuidados específicos para cada caso ou *script* de teste. Nesse contexto, apenas a descrição do sistema requer atenção, simplificando consideravelmente a tarefa de manutenção.

Objetivo Específico 5: Conduzir uma avaliação da Teasy Mobile Language junto a profissionais da indústria, visando obter *insights* sobre a praticidade de uso e realizar comparações com soluções já existentes.

1.3 Metodologia

Com a intenção de criar uma DSL para testes funcionais em aplicativos móveis, buscando minimizar o esforço necessário sem comprometer a qualidade dos testes gerados, tornou-se imperativo reexaminar o Mapeamento Sistemático de Literatura (SMS) conduzido no estudo anterior. Essa revisão se faz necessária para contemplar outras linguagens e funcionalidades, levando em consideração os desafios atuais do cenário de desenvolvimento de *software*. Esse SMS, teve como principal propósito identificar todas as linguagens existentes para testes funcionais, juntamente com seus benefícios, limitações e tendências. Esta análise se revela fundamental, pois fornece *insights* valiosos para o desenvolvimento da DSL em questão, visando atender aos objetivos de eficiência e qualidade no contexto específico de testes para aplicativos móveis.

Os resultados obtidos a partir desta pesquisa não apenas proporcionaram um conjunto abrangente de linguagens, incluindo seus benefícios, limitações e tendências, mas também delinearam um conjunto de ferramentas viáveis para a implementação efetiva de uma DSL. Essa compilação de informações representa um alicerce valioso para orientar o desenvolvimento de uma DSL que seja não apenas eficaz, mas também alinhada com as necessidades atuais e futuras no cenário de testes funcionais para aplicativos móveis.

Com base nas linguagens e ferramentas identificadas, juntamente com a versão destinada a aplicações web, delineou-se e implementou-se a Teasy Mobile Language. Essa implementação foi realizada utilizando o *Meta Programming System* (MPS) (JETBRAINS, 2019) como *Language Workbench* (LW) para o desenvolvimento. Essa decisão de projeto visa aproveitar a flexibilidade e poder de customização oferecidos pelo MPS para garantir uma construção eficaz e adaptável da Teasy Mobile Language. Na condução deste estudo, optou-se por manter a estrutura de testes escalável previamente estabelecida no estudo anterior. Tal decisão foi motivada pela comprovada eficácia junto aos usuários finais e pelo atendimento dos objetivos de reuso e escalabilidade. Dessa maneira, a Teasy Mobile Language já é capaz de gerar scripts de teste dentro dessa estrutura de forma automática, eliminando a necessidade de intervenção adicional por parte do testador.

Com a finalidade de avaliar a eficácia da Teasy Mobile Language, concebeu-se a realização de um estudo de caso na indústria, empregando uma aplicação real como cenário de avaliação. Assim, foi viabilizada a avaliação da representatividade da linguagem para o domínio, bem como sua eficácia na criação de testes funcionais para aplicações móveis, em conformidade com os objetivos específicos. Além disso, a análise abordou as limitações atuais da solução, proporcionando uma visão abrangente do seu desempenho no momento.

1.4 Principais Contribuições

A principal contribuição deste trabalho reside no desenvolvimento da Teasy Mobile Language, uma linguagem de testes funcionais de *software* projetada para otimizar a implementação e manutenção de *scripts* de teste automatizados em aplicações móveis, reduzindo significativamente o esforço e o tempo dedicados a essas atividades. Além disso, este estudo oferece uma contribuição valiosa ao mapear e analisar as DSLs existentes para testes funcionais, destacando seus benefícios, limitações e tendências predominantes. Essas contribuições coletivas buscam aprimorar a eficiência e a qualidade dos processos de teste, alinhando-se às demandas crescentes no desenvolvimento de *software*, especialmente em ambientes *mobile*.

Outra contribuição significativa deste trabalho é a avaliação prática da linguagem, realizada por meio de um estudo de caso na indústria, envolvendo o teste de um aplicativo real com mais de 15 mil usuários. Essa abordagem possibilitou uma análise concreta do impacto que a solução pode ter em projetos reais, fornecendo *insights* valiosos sobre sua eficácia e desafios encontrados na implementação prática. Esta avaliação prática, ao identificar benefícios observados e limitações identificadas, serve como base sólida para orientar futuros trabalhos na correção e aprimoramento contínuo da Teasy Mobile Language.

1.5 Organização do Trabalho

A continuidade deste estudo foi organizada da seguinte maneira:

- O Capítulo 2 apresenta a base teórica essencial para a realização do estudo, abordando conceitos fundamentais sobre testes de *software* e linguagens de domínio específico, além das lições apresentadas neste capítulo.
- O Capítulo 3 aborda os trabalhos relacionados ao estudo, detalhes da reexecução do SMS, e as conclusões derivadas das lições apresentadas neste capítulo.
- O Capítulo 4 apresenta os detalhes pertinentes à Teasy Mobile Language, destacando, também, as lições extraídas ao longo deste capítulo.
- O Capítulo 5 apresenta a validação conduzida na linguagem, juntamente com um conjunto de lições extraídas de cada capítulo.
- O Capítulo 6 apresenta as considerações finais e os trabalhos futuros deste estudo.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, exploramos a terminologia e delineamos os principais conceitos que permeiam todo este trabalho, especialmente em relação à DSL, conforme discutido na Seção 2.1. Na mesma seção, concentramo-nos nos principais conceitos que envolvem DSL. A Seção 2.2, por sua vez, aborda os fundamentos dos testes de software.

2.1 Linguagem de Domínio Específico

Diferentemente das Linguagens de Propósito Gerais (GPLs) como JavaScript (NEGRINO; SMITH, 2001) e Java (GOSLING; HOLMES; ARNOLD, 2005), as linguagens de domínio específico (DSLs) são formuladas com o propósito específico de fornecer suporte e abordar desafios inerentes a um domínio particular (MERNIK; HEERING; SLOANE, 2005). As distinções entre uma GPL e uma DSL não são sempre claras, uma vez que uma linguagem pode conter apenas alguns recursos específicos para um domínio particular, permitindo sua aplicação mais abrangente (FOWLER, 2010). Uma DSL, ao ter como principal propósito a resolução de problemas em um domínio específico, tipicamente não consegue abordar adversidades que ocorram fora desse cenário delimitado (FOWLER, 2010). Ela é concebida como uma componente integrante da Engenharia de Domínio, focando na otimização e solução eficiente de desafios específicos dentro do contexto para o qual foi projetada (FOWLER, 2010). Dessa forma, o nível de abstração de uma DSL tende a ser geralmente mais elevado em comparação com uma GPL. Esse aumento na abstração visa simplificar o uso por parte dos especialistas no domínio, aproximando-os efetivamente da resolução dos problemas específicos para os quais a DSL foi concebida (ŽIVANOV; RAKIĆ; HAJDUKOVIĆ, 2008).

Embora uma DSL tenha o potencial de elevar a produtividade e o envolvimento dos especialistas no domínio, sua implementação não se configura como uma tarefa trivial. Além disso, o perfil dos especialistas que utilizarão a linguagem pode exercer influência significativa no processo de desenvolvimento e na natureza da DSL. Diante dessa complexidade, existem diversas abordagens para a implementação de uma DSL, as quais impactam diretamente na definição de seus tipos (FOWLER, 2010). A seguir, serão apresentados os principais tipos de DSLs.

2.1.1 Linguagens Internas

As Linguagens Internas, ou Linguagens Embarcadas, são formuladas e implementadas dentro de uma linguagem já existente, geralmente uma GPL (FOWLER, 2006). Essas linguagens estabelecem uma relação direta com a GPL na qual foram originadas, conferindo uma vantagem substancial aos especialistas do domínio que as empregam, especialmente quando possuem conhecimento prévio da respectiva GPL. Por exemplo,

uma linguagem como "XUnit" para testes unitários, desenvolvida em Java, proporcionará benefícios significativos aos programadores familiarizados com Java, tornando a adoção e a compreensão da linguagem "XUnit" mais acessíveis para esse público.

2.1.2 Linguagens Externas

As Linguagens Externas são DSLs que operam de forma autônoma, sem depender de outras linguagens ou GPLs; elas incorporam seus próprios analisadores e geradores, caso necessário (FOWLER, 2010). A principal vantagem desse tipo de DSL reside na capacidade do desenvolvedor criar uma sintaxe específica e livre para o domínio em questão, permitindo até mesmo a elevação do nível de abstração para se adequar a um contexto específico.

2.1.3 *Languages Workbenches*

As *Languages Workbenches* (LWs) são ferramentas projetadas para facilitar a implementação de DSLs, oferecendo um ambiente de desenvolvimento especializado (FOWLER, 2005). Esse ambiente disponibiliza uma estrutura concebida para a definição de um modelo semântico, a edição da DSL e a execução de uma semântica comportamental, seja por meio de interpretação ou, se necessário, pela geração de código. Dessa forma, obtém-se uma maior facilidade e uma redução nos custos associados ao desenvolvimento de uma DSL (FOWLER, 2005). A seleção entre uma DSL textual, gráfica, projecional ou mesmo tabular é determinada pelo contexto específico do domínio em questão.

Com base no domínio em consideração, é crucial tomar a decisão de optar por um LW, considerando que existem opções tanto pagas quanto de código aberto. Entre as opções de LWs gratuitas, destacam-se o XText (BETTINI, 2016), utilizado para implementar linguagens textuais, o Sirius (SIRIUS, 2019), que facilita a criação de linguagens gráficas, e, por fim, o MPS (JETBRAINS, 2019), que permite a elaboração de linguagens projecionais. Essa comparação pode ser mais claramente compreendida ao consultar o estudo de (IUNG et al., 2020), que se concentra em um mapeamento específico de LWs, fornecendo diversos *insights* relacionados a esse contexto.

2.1.4 *Meta Programming System*

O MPS é uma LW de código aberto destinada à concepção de DSLs, consolidando-se como uma das opções mais estabelecidas na atualidade, empregando a técnica de *Projectional Editing* (FOWLER, 2010). Esse recurso capacita os desenvolvedores a criar seus próprios editores para a elaboração da linguagem especificada, oferecendo uma projeção que pode ser tanto textual quanto gráfica, conforme necessário (JETBRAINS, 2019). O MPS, desenvolvido pela JetBrains, incorpora uma linguagem de apoio ao desenvolvimento chamada MPS *BaseLanguage*, fundamentada na linguagem de programação Java.

Ao longo do tempo, foram adicionadas novas linguagens, como Extensible Markup Language (XML), C e JavaScript, que também podem ser empregadas na definição de uma linguagem. Com a finalidade de criar uma DSL no MPS, é essencial definir os conceitos da linguagem e estabelecer suas relações. Após essa etapa, é possível configurar um editor para esses conceitos e, se necessário, especificar um gerador de artefatos correspondente (JETBRAINS, 2019). O MPS também adota uma abordagem baseada em árvore de sintaxe abstrata, o que significa que cada atributo deve ser selecionado a partir das opções oferecidas ao usuário. Dessa forma, o elemento escolhido é então projetado (JETBRAINS, 2019). Adicionalmente, a LW oferece um conjunto de personalizações para o editor, possibilitando simular o comportamento de *Integrated Development Environments (IDEs)* convencionais (JETBRAINS, 2019). Essas adaptações contribuem para uma experiência de desenvolvimento mais familiar e eficiente, aproximando-se das funcionalidades típicas das IDEs tradicionais.

2.2 Teste de *software*

Nesta seção, serão abordados os fundamentos associados aos testes de software, cruciais para a concepção e implementação da Teasy Mobile Language. Destacaremos dois tipos principais de testes: os testes funcionais e os testes baseados em modelos.

2.3 Teste funcionais

Os testes funcionais visam identificar *bugs* dentro de um produto de software, conforme destacado por (DELAMARO; JINO; MALDONADO, 2013). Essa prática não apenas eleva a confiabilidade do produto a ser entregue, mas também desempenha um papel crucial na verificação do *software* em relação aos requisitos do cliente, conforme apontado por (RIOS et al., 2007). Nos testes funcionais, o produto de software é tratado como uma caixa-preta, indicando que não se verifica o código-fonte, mas sim o seu comportamento, conforme elucidado por (DELAMARO; JINO; MALDONADO, 2013). Este comportamento é derivado dos requisitos e é verificado por meio dos casos de teste, que constituem um conjunto de condições para a execução dos testes. Esses casos de teste têm como elementos essenciais as entradas, as ações realizadas e os resultados esperados, como destacado por (MYERS; SANDLER; BADGETT, 2011). Dessa forma, viabilizando a identificação de *bugs* funcionais antes da entrega do produto, o que contribui para a redução dos custos associados a correções após a entrega do *software*. Esses custos pós-entrega são geralmente os mais elevados no ciclo de vida do produto, conforme evidenciado por (DELAMARO; JINO; MALDONADO, 2013).

2.4 Testes Baseados em Modelos

O Teste Baseado em Modelo, conhecido como MBT, representa uma abordagem para o desenvolvimento de testes, visando elevar o nível de abstração nesse processo, conforme discutido por (UTTING; LEGEARD, 2010). Essa estratégia busca proporcionar uma compreensão mais aprofundada do produto de software a ser testado, promovendo, assim, uma melhor especificação.

No âmago dessa abordagem, um modelo do produto de software é delineado, definindo seu comportamento e dados correlatos. A partir dessas informações, podem ser automaticamente gerados casos de teste e scripts de testes automatizados (UTTING; LEGEARD, 2010). O MBT surge como uma alternativa valiosa, pois mitigando ambiguidades presentes na documentação textual, pode antecipar a detecção de *bugs* antes mesmo do desenvolvimento efetivo do produto (UTTING; LEGEARD, 2010). Isso se dá pela necessidade apenas do modelo relacionado ao software.

Além disso, o MBT apresenta potencial para redução de custos, uma vez que possibilita a identificação precoce de problemas, e destaca-se por permitir que profissionais não programadores participem ativamente na elaboração de testes automatizados. Isso é viabilizado pela capacidade de geração automática de scripts de teste, como apontado por (UTTING; LEGEARD, 2010).

2.5 Lições do capítulo

O exame aprofundado sobre testes de software e DSLs neste capítulo revelou insights valiosos para o entendimento e aprimoramento das práticas de desenvolvimento e qualidade. Dentre as principais lições extraídas, destacam-se:

- **Importância estratégica dos testes funcionais:** A ênfase nos testes funcionais de software se revela crucial, não apenas para mitigar custos associados à correção de defeitos pós-implantação, mas também para fortalecer a confiabilidade do produto entregue. A abordagem centrada no comportamento do software, como discutido, contribui diretamente para a verificação eficaz dos requisitos do cliente.

- **Modelos como Facilitadores de Testes Automatizados:** A utilização de modelos para definir casos de teste oferece uma alternativa eficaz para simplificar e acelerar a criação de testes automatizados. Centralizar as informações em um modelo único proporciona uma visão coesa do comportamento esperado do software, facilitando a manutenção e a compreensão dos testes.

- **DSLs para possuir uma maior abstração e produtividade:** A adoção de DSLs, especialmente quando representadas através do MPS, surge como uma estratégia para alcançar maior abstração. Ao oferecer uma sintaxe voltada para

testadores, essas linguagens específicas de domínio aprimoram a compreensão e a produtividade, possibilitando uma comunicação mais eficaz entre os envolvidos no processo de desenvolvimento.

- **Versatilidade do MPS na definição de DSLs:** O MPS destaca-se como uma alternativa versátil para a definição de DSLs. Sua capacidade de projeção, que permite o uso de representações textuais e gráficas conforme necessário, e a geração automática de artefatos reforçam sua adequação para o desenvolvimento da Teasy Mobile Language.

Em síntese, as lições extraídas deste capítulo sustentam a justificativa para a concepção e implementação da Teasy Mobile Language, evidenciando a relevância das práticas discutidas para aprimorar a eficácia e eficiência no processo de desenvolvimento de software.

3 MAPEAMENTO SISTEMÁTICO DA LITERATURA

Este capítulo representa um passo significativo na compreensão do panorama atual das DSLs direcionadas a testes funcionais. Este capítulo apresenta uma reexecução do SMS inicialmente proposto em (LIMA, 2021), adaptado e atualizado para refletir o contexto atual. Com o intuito de realizar uma análise abrangente e identificar as nuances essenciais desse domínio, conduzimos um SMS. O foco primordial deste estudo foi desvelar as DSLs existentes para testes funcionais, além de esmiuçar seus principais benefícios, limitações e as tendências observadas no cenário atual. Com base nisso, o capítulo foi estruturado da seguinte forma: A seção 3.1 descreve o protocolo adotado pelo SMS. A seção 3.2 descreve a condução deste SMS. A seção 3.3 apresenta a análise da aplicação dos Critérios de Qualidade (CQs) deste SMS. A seção 3.4 discute sobre as linguagens encontradas na literatura para responder e discutir as nossas Questões de Pesquisa (QPs). A seção 3.5 discute sobre os trabalhos relacionados ao SMS realizado. A seção 3.4.7 discute sobre as ameaças à validade presentes no estudo. Por fim a seção 3.7 apresenta as lições aprendidas neste capítulo.

3.1 Protocolo

A condução de um SMS requer um protocolo meticulosamente elaborado para assegurar resultados eficazes. No intuito de atender a essa necessidade, foi optado por adotar um protocolo proposto por (PETERSEN et al., 2008), o qual foi adaptado e ajustado para atender aos requisitos específicos do contexto DSLs para testes funcionais, possibilitando a extração abrangente dos dados desejados.

O primeiro passo desse protocolo personalizado consistiu na definição clara dos objetivos do estudo, alinhados às Questões de Pesquisa (QPs) fundamentais do SMS. Em consonância com esses parâmetros, o principal propósito do estudo é:

> ***Objetivo:*** *Identificar DSLs voltadas para testes funcionais de software, buscando compreender suas funcionalidades, benefícios, limitações e tendências. Viabilizando o fornecimento de insights sobre o estado atual da área e identificar possíveis lacunas que possam direcionar futuras pesquisas.*

Com o objetivo de atingir o objetivo citado anteriormente foram definidas as respectivas Questões de Pesquisa (QPs):

QP.1. **Em qual domínio específico as DSLs de teste de software estão direcionadas (por exemplo, aplicações web, aplicações móveis, etc.)?** O objetivo desta questão de pesquisa é categorizar as DSLs identificadas com base no

domínio que elas abordam. Vale ressaltar que, embora não seja obrigatório que as DSLs gerem artefatos, é essencial que elas ofereçam suporte a um domínio específico.

QP.2. **Quais são as funcionalidades disponibilizadas por essas DSLs?** Uma vez que as DSLs são concebidas para aprimorar a eficiência em uma etapa específica do processo, a análise dos funcinalidades que as DSLs oferecem torna-se crucial. Esta questão visa auxiliar o testador na avaliação da adequação da DSL escolhida para suas tarefas, examinando as capacidades e possibilidades proporcionadas pela linguagem específica de domínio.

QP.3. **Quais tecnologias ou notações são prevalentes no desenvolvimento das DSLs?** A diversidade de tecnologias disponíveis para a implementação de DSLs, como Xtext, Sirius, MPS, entre outras, tendem a se destacar no cenário. Contudo, o propósito desta questão é realizar um mapeamento das tecnologias frequentemente empregadas no desenvolvimento de DSLs direcionadas a testes funcionais.

QP.4. **Quais técnicas, abordagens ou metodologias as DSLs empregam para assegurar uma cobertura abrangente nos testes gerados?** A intenção desta questão é analisar de que maneira as DSLs contribuem para aprimorar ou simplificar a abrangência dos testes realizados.

QP.5. **De que maneira os dados do Sistema Sob Teste (SUT) são representados nas DSLs?** Compreender como os dados de entrada são representados nas DSLs é crucial para a execução de cenários de teste. A variabilidade nos dados de entrada, dependendo da aplicação de software, implica em diferentes casos de teste. Portanto, esta questão visa investigar as abordagens utilizadas pelas DSLs com ênfase em testes de software para representar os dados de entrada do SUT.

QP.6. **Quais benefícios, desvantagens e tendências estão associados ao desenvolvimento e adoção de DSLs para testes funcionais?** Explorar os aspectos relacionados aos benefícios, desvantagens e tendências dessas DSLs é crucial para compreender o cenário atual. Essa questão visa identificar as práticas existentes, áreas de aprimoramento e sugestões do estado da arte para aprimorar o desenvolvimento e a adoção de DSLs voltadas para testes funcionais.

Na busca por fontes, a definição precisa dos termos relevantes, bem como de seus sinônimos, relacionados aos objetivos e as QPs que desempenham um papel crucial no SMS. Com a finalidade de estabelecer essa *string* de busca, utilizamos o método *Population Intervention Comparison Outcome and Context (PICOC)* (WOHLIN et al., 2012), conforme detalhado a seguir:

- ***Population***: Todos os estudos publicados que propõem ou utilizam uma DSL para teste funcional de *software*.

- ***Intervention***: As DSLs para testes funcionais de *software* e suas funcionalidades, limitações e benefícios.
- ***Comparison***: 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.
- ***Outcome***: Um conjunto de DSLs para testes funcionais de *software*, juntamente com suas desvantagens, benefícios e tendências.
- ***Context***: Este estudo considera todos os contextos, sendo eles acadêmicos ou industriais.

Com base nas informações do PICOC, a equipe extraiu termos e sinônimos para formar a *string* base de busca, empregando os operadores *OR* e *AND*, como delineado na Tabela 1. A Tabela 2 detalha as seis bibliotecas digitais utilizadas para a pesquisa de fontes primárias, incluindo a *string* de pesquisa específica para cada biblioteca. Estas *strings* específicas foram delineadas após a execução, análise e aprimoramento da *string* de pesquisa base até alcançar resultados satisfatórios.

Por fim, para a triagem dos estudos obtidos em cada biblioteca digital, conforme o protocolo do SMS, foram estabelecidos um Critério de Inclusão (CI) e dois Critérios de Exclusão (CEs) para a seleção apenas de artigos relevantes, conforme descrito na Tabela 3.

Tabela 1 – Termos, Sinônimos e a *string* de busca

Termos	**Sinônimos**
Domain-Specific Language	DSL, DSML, domain specific language, domain-specific-language, domain-specific modeling language, domain-specific modeling language, domain-specific-modeling language, domain-specific-modeling-language
Functional testing	black box test, black-box test, black box testing, black-box testing, functional test, regression test, regression testing, acceptance test, acceptance testing
String de busca	((“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”))

Fonte: Autor.

Na próxima fase do protocolo SMS, a avaliação da qualidade dos estudos selecionados é conduzida. Dessa forma, foi estabelecido um conjunto de CQs que serviram

Tabela 2 – Bibliotecas digitais e as *Strings* de busca

Biblioteca Digital	***String* de busca**
ACM	((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*”))
Eng. Village	(“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”)
IEEE Xplore	((“Full Text & Metadata”:dsl) OR (“Full Text & Metadata”:dsml) OR (“Full Text & Metadata”:“domain specific language”) OR (“Full Text & Metadata”:“domain specific model* language”)) AND ((“Full Text & Metadata”:“functional test*”) OR (“Full Text & Metadata”:“black box test*”) OR (“Full Text & Metadata”:“regression test*”) OR (“Full Text & Metadata”:“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”))
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*”))

Fonte: Autor

como critérios para qualificação e classificação dos estudos. Vale ressaltar que essa pontuação não determinou a exclusão de nenhum estudo, todos os estudos foram considerados, desde que estivessem em conformidade com a CI. Cada pergunta recebeu uma pontuação com base nas respostas atribuídas, e após a avaliação de todas as perguntas, foi calculada uma pontuação total para cada estudo. As respostas possíveis para cada pergunta foram definidas como “total” quando o estudo continha informações suficientes para responder à

Tabela 3 – Critérios de Inclusão e Exclusão

Critério de Inclusão	**Critério de Exclusão**
CI1. O estudo deve propor ou apresentar explicitamente uma DSL para testes funcionais de *software*.	CE1. Artigos escritos em outros idiomas além do inglês.
	CE2. O estudo não apresenta a DSL para testes funcionais de *software* ou o uso de uma.

Fonte: Autor

QA, resultando em uma pontuação de 1 ponto; "parcial" quando o estudo proporcionava apenas uma resposta parcial à QA, conferindo uma pontuação de 0,5; e "não responde" quando o estudo não apresentava informações relevantes para responder à pergunta, resultando em uma pontuação de 0. As QAs e suas respectivas regras de pontuação são detalhadas a seguir.

Critérios de Qualidade (CQs):

CQ1. O estudo apresenta as principais funcionalidades da DSL?

Total: O estudo detalha todas as principais funcionalidades da DSL;

Parcial: O estudo somente cita as funcionalidades principais da DSL;

Não responde: O estudo não apresenta nenhum dado referente as funcionalidades da DSL.

CQ2. O estudo apresenta em quais tipos de *software* os *scripts* gerados pela DSL podem ser aplicados (ex.: *Web, Mobile, Desktop, etc.*)?

Total: O estudo apresenta todos os tipos de *software* possíveis no qual a DSL pode ser aplicada;

Parcial: O estudo apresenta apenas alguns tipos nos quais a DSL pode ser aplicada;

Não responde: O estudo não cita os tipos de *software* no qual a DSL pode ser aplicada.

CQ3. O estudo apresenta como os dados do SUT são representados na linguagem?

Total: O estudo apresenta e dá exemplos de como informar os dados do SUT para a DSL;

Parcial: O estudo apenas cita a possibilidade sem dar exemplos de uso;

Não responde: O estudo não informa como os dados do SUT são informados à DSL.

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

Total: O estudo apresenta os benefícios, as desvantagens e melhorias para a DSL apresentada;

Parcial: O estudo apresenta somente parte da informação;

Não responde: O estudo não apresenta nenhuma informação.

Outro componente crucial de um SMS é o formulário utilizado para extrair dados. A informação obtida ao aplicar essa extração nos estudos desempenha um papel fundamental na consecução dos objetivos estabelecidos. O formulário de extração de dados empregado continha os seguintes campos:

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

Com a conclusão do delineamento do nosso protocolo, a busca pelos estudos foi iniciada. Os resultados serão detalhados nas seções seguintes.

3.2 Condução do SMS

Na primeira aplicação deste protocolo de SMS (LIMA, 2021), em conjunto com a execução de um processo de *Snowballing* (WOHLIN et al., 2012), resultou na identificação de 1.077 estudos, dos quais 77 foram considerados relevantes e analisados. Na segunda execução do SMS, um total de 3.544 estudos foi identificado, resultando na inclusão de 30 novos estudos relevantes. A tabela 4 apresenta os números de estudos mapeados por biblioteca digital na etapa. Portanto, o conjunto final de estudos analisados para responder às nossas perguntas foi composto por 107 estudos.

Tabela 4 – Números da seleção dos estudos do SMS

Biblioteca Digital	**Período 1**	**Período 2**
ACM Digital Library	16	149
IEEE Digital Library	442	1084
Science Direct	225	532
Scopus	33	79
Springer Link	335	1643
EI Compendex	26	57
Total de estudos	1.077	3.544
Estudos selecionados	**77**	**30**

Fonte: Autor.

3.3 Análise dos Critérios de Qualidade (CQs)

A aplicação dos CQs em um SMS tem como objetivo avaliar a contribuição de cada estudo aprovado no processo de seleção. Na Tabela 5, encontram-se as informações referentes às pontuações atribuídas a cada estudo. Cada entrada na tabela é identificada por um número de identificação (Identificador (ID)), associado à referência do estudo. As colunas CQ1, CQ2, CQ3 e CQ4 exibem as pontuações atribuídas a cada estudo com base nos CQs específicos. A coluna "Des" classifica o estudo como Pouco Relevante, Relevante ou Muito Relevante, alinhando-se com os objetivos desta pesquisa. A coluna "Sc" representa a pontuação total de cada estudo, calculada como a soma das pontuações atribuídas a cada critério.

Estudos			CQ				Qualidade		Estudos			CQ				Qualidade	
ID	**Citação**	**Ano**	**1**	**2**	**3**	**4**	**Sc**	**Des**	**ID**	**Citação**	**Ano**	**1**	**2**	**3**	**4**	**Sc**	**Des**
S1	(FELDERER; JESCHKO, 2018)	2018	Y	Y	Y	P	3.5	V	S55	(MORGADO; PAIVA, 2015)	2015	Y	Y	Y	Y	4.0	V
S2	(HARGASSNER et al., 2008)	2008	Y	Y	Y	P	3.5	V	S56	(HU; ZHU; YANG, 2018)	2018	Y	Y	Y	Y	4.0	V
S3	(TÖRSEL, 2013b)	2013	Y	Y	Y	Y	4.0	V	S57	(KOS et al., 2012)	2012	Y	Y	Y	N	3.0	G
S4	(IBER et al., 2015b)	2015	Y	Y	Y	P	3.5	V	S58	(MOREIRA; PAIVA, 2014b)	2014	Y	Y	Y	Y	4.0	V
S5	(DWARAKANATH et al., 2017)	2017	Y	Y	Y	Y	4.0	V	S59	(PAIVA; VILELA, 2017)	2017	Y	Y	Y	Y	4.O	V
S6	(GAFUROV; HURUM; MARKMAN, 2018)	2018	Y	Y	Y	Y	4.0	V	S60	(Dias; Paiva, 2017)	2017	Y	Y	Y	Y	4.0	V
S7	(DUCLOS et al., 2013)	2013	Y	Y	Y	P	3.5	V	S61	(HAMMOUD; ZARAKET; MASRI, 2017)	2017	Y	Y	Y	P	3.5	V
S8	(HÄSER; FELDERER; BREU, 2016a)	2016	Y	Y	P	P	3.0	G	S62	(SILVA; PAIVA; SILVA, 2018)	2018	Y	Y	Y	Y	4.0	V
S9	(SANZ et al., 2015)	2015	Y	Y	Y	P	3.5	V	S63	(PAIVA; SILVA; SILVA, 2019)	2018	Y	Y	Y	P	3.5	V
S10	(CONTAN; MICLEA; DEHELEAN, 2017)	2017	Y	Y	Y	P	3.5	V	S64	(MOREIRA; PAIVA, 2014c)	2014	Y	Y	Y	P	3.5	V
S11	(ZHOU; YIN, 2014)	2014	Y	Y	Y	Y	4.0	V	S65	(OLAJUBU et al., 2017)	2017	Y	Y	Y	P	3.5	V
S12	(MAKEDONSKI et al., 2016)	2016	Y	Y	Y	Y	4.0	V	S66	(BUSSENOT; LEBLANC; PERCEBOIS, 2016)	2016	Y	Y	Y	Y	4.0	V
S13	(SIPPL et al., 2016)	2016	P	Y	P	Y	3.0	G	S67	(KESSERWAN et al., 2019)	2019	Y	Y	Y	P	3.5	V
S14	(JORGE et al., 2018)	2018	Y	Y	Y	Y	4.0	V	S68	(HÄRLIN, 2016)	2016	Y	Y	Y	P	3.5	V
S15	(KING et al., 2014)	2014	P	Y	Y	P	3.0	G	S69	(SILVA, 2018)	2018	Y	Y	Y	Y	4.0	V
S16	(CHATLEY; AYRES; WHITE, 2010)	2010	Y	Y	Y	Y	4.0	V	S70	(CARVALHO, 2016)	2016	Y	Y	N	N	2.0	F
S17	(ARTHO et al., 2013)	2013	Y	Y	Y	Y	4.0	V	S71	(SACRAMENTO, 2014)	2014	Y	Y	Y	Y	4.0	V
S18	(NABUCO; PAIVA, 2014)	2014	Y	Y	Y	P	3.5	V	S72	(RODRIGUES, 2018)	2018	Y	Y	N	P	2.5	G
S19	(ALEGROTH; BACHE; BACHE, 2015)	2015	Y	Y	P	P	3.0	G	S73	(VILELA, 2013)	2013	Y	Y	Y	P	3.5	V
S20	(MILLER; KUMAR; SINGHAL, 2009)	2009	P	Y	P	N	2.0	F	S74	(SILVA, 2017)	2017	Y	Y	Y	Y	4.0	V
S21	(KOS; MERNIK; KOSAR, 2016)	2016	Y	Y	Y	Y	4.0	V	S75	(MACIEL, 2019)	2019	Y	Y	Y	Y	4.0	V
S22	(MAKEDONSKI et al., 2019)	2019	Y	Y	Y	Y	4.0	V	S76	(SACRAMENTO; PAIVA, 2014)	2014	Y	Y	Y	Y	4.0	V
S23	(HÄSER; FELDERER; BREU, 2014)	2014	Y	N	Y	P	2.5	G	S77	(BUSSENOT; LEBLANC; PERCEBOIS, 2018)	2018	Y	Y	Y	Y	4.0	V
S24	(HERBOLD et al., 2015)	2015	Y	Y	Y	P	3.5	V	S78	(SADOWSKI; YI, 2009)	2009	Y	Y	Y	Y	4.0	V
S25	(LUNA; ROSSI; GARRIGÓS, 2011)	2011	Y	Y	Y	Y	4.0	V	S79	(HESENIUS; GRIEBE; GRUHN, 2014)	2014	Y	Y	Y	Y	4.0	V
S26	(LUNA; GARRIGOS; ROSSI, 2010)	2010	Y	Y	Y	P	3.5	V	S80	(GRIEBE; GRUHN, 2014)	2014	Y	Y	Y	Y	4.0	V
S27	(HOLMES et al., 2018)	2018	Y	Y	Y	P	3.5	V	S81	(CAMILLI et al., 2018)	2018	Y	N	Y	P	2.5	G
S28	(IBER et al., 2015a)	2015	Y	Y	Y	Y	4.0	V	S82	(RAMLER; KLAMMER, 2019)	2019	P	Y	N	N	1.5	F
S29	(PRASETYA et al., 2011)	2011	Y	Y	Y	P	3.5	V	S83	(TUGLULAR; ŞENSüLüN, 2019)	2019	Y	Y	Y	P	3.5	V
S30	(ELODIE et al., 2018)	2018	Y	Y	Y	P	3.5	V	S84	(SIPPL et al., 2019)	2019	Y	Y	Y	P	3.5	V
S31	(TALBY, 2009)	2009	Y	N	Y	P	2.5	G	S85	(CAVALCANTE; SALES, 2019)	2019	Y	Y	Y	P	3.5	V
S32	(HÄSER; FELDERER; BREU, 2016b)	2016	Y	P	Y	P	3.0	G	S86	(MA et al., 2019)	2019	Y	Y	Y	P	3.5	V
S33	(TORSEL, 2011)	2011	Y	Y	Y	Y	4.0	V	S87	(SILVA; WINCKLER; TRæTTEBERG, 2019a)	2019	Y	Y	Y	P	3.5	V
S34	(MICALLEF; COLOMBO, 2015)	2015	Y	Y	Y	Y	4.0	V	S88	(QUYET et al., 2019)	2019	Y	Y	Y	P	3.5	V
S35	(COSTA; PAIVA; NABUCO, 2014)	2014	Y	Y	Y	Y	4.0	V	S89	(CAMILLI et al., 2019)	2019	Y	Y	Y	P	3.5	V
S36	(MOREIRA; PAIVA; MEMON, 2013)	2013	Y	Y	Y	Y	4.0	V	S90	(SILVA; WINCKLER; TRæTTEBERG, 2019b)	2019	Y	Y	Y	P	3.5	V

Legenda - **Y**: *Sim (1)*, **N**: *Não (0)*, **P**: *Parcial (0.5)*, **Sc**: *Pontuação*, **Des**: *Descrição*, **F**: *Pouco relevante*, **G**: *Relevante*, **V**: *Muito relevante.*

Estudos			CQ				Qualidade		Estudos			CQ				Qualidade	
ID	**Citação**	**Ano**	**1**	**2**	**3**	**4**	**Sc**	**Des**	**ID**	**Citação**	**Ano**	**1**	**2**	**3**	**4**	**Sc**	**Des**
S37	(VILELA; PAIVA, 2014)	2014	Y	P	Y	N	2.5	G	S91	(SILVA; WINCKLER; TRæTTEBERG, 2020)	2020	Y	Y	Y	P	3.5	V
S38	(MONTEIRO; PAIVA, 2013)	2013	Y	P	Y	P	3.0	G	S92	(NIKIFOROVA et al., 2020)	2020	Y	P	Y	N	2.5	G
S39	(MOREIRA; PAIVA, 2014a)	2014	Y	P	Y	P	3.0	G	S93	(ZANIN; ZORZO; NUNES, 2020)	2020	Y	Y	Y	Y	4.0	V
S40	(HOIS; SOBERNIG; STREMBECK, 2014)	2014	Y	Y	Y	Y	4.0	V	S94	(COTRONEO et al., 2020)	2020	Y	Y	Y	P	3.5	V
S41	(SANTIAGO et al., 2013)	2013	Y	Y	Y	P	3.5	V	S95	(RENNOCH et al., 2020)	2020	Y	Y	Y	P	3.5	V
S42	(KANSTRÉN; PUOLITAIVAL, 2012)	2012	Y	Y	Y	Y	4.0	V	S96	(WOLDE; BOLTANA, 2020)	2020	Y	Y	Y	P	3.5	V
S43	(KOS et al., 2011)	2011	Y	Y	P	P	3.0	G	S97	(MEIXNER et al., 2020)	2020	Y	Y	Y	P	3.5	V
S44	(KOS et al., 2011)	2011	P	Y	P	P	2.5	G	S98	(SNOOK et al., 2020)	2020	Y	Y	Y	P	3.5	V
S45	(ULRICH et al., 2014)	2014	Y	P	Y	P	3.0	G	S99	(KIRINUKI et al., 2021)	2021	Y	Y	Y	P	3.5	V
S46	(ARTHO et al., 2017)	2017	Y	Y	N	P	2.5	G	S100	(SCHNEID et al., 2021)	2021	Y	Y	Y	P	3.5	V
S47	(ARTHO et al., 2015b)	2015	Y	Y	Y	Y	4.0	V	S101	(KHORRAM et al., 2022a)	2022	Y	Y	Y	Y	4.0	V
S48	(ARTHO et al., 2015a)	2015	Y	N	Y	N	2.0	F	S102	(KHORRAM et al., 2022b)	2022	Y	Y	Y	Y	4.0	V
S49	(CUNHA; SONG, 2014)	2014	Y	Y	Y	Y	4.0	V	S103	(CASADEI et al., 2022)	2022	P	Y	N	P	2.0	F
S50	(LUNA et al., 2010)	2010	Y	Y	Y	P	3.5	V	S104	(OLIVEIRA et al., 2022)	2022	Y	Y	Y	Y	4.0	V
S51	(OTADUY; DÍAZ, 2017)	2017	Y	Y	Y	P	3.5	V	S105	(OTHMAN; ZEIN, 2022)	2022	Y	Y	Y	P	3.5	V
S52	(HÄSER; FELDERER; BREU, 2018)	2018	P	Y	P	N	2.0	F	S106	(OLBERG; STREY, 2022)	2022	Y	Y	Y	P	3.5	V
S53	(WINKLER; MEIXNER; BIFFL, 2018)	2018	Y	N	Y	P	2.5	G	S107	(KIRINUKI et al., 2022)	2022	Y	Y	Y	Y	4.0	V
S54	(MOREIRA et al., 2017b)	2017	Y	Y	Y	Y	4.0	V									

Legenda - **Y**: *Sim (1)*, **N**: *Não (0)*, **P**: *Parcial (0.5)*, **Sc**: *Pontuação*, **Des**: *Descrição*, **F**: *Pouco relevante*, **G**: *Relevante*, **V**: *Muito relevante.*

Tabela 5 – Análise da qualidade dos estudos

3.4 Resultados e respostas das QPs

Nesta seção, são apresentados e discutidos os resultados obtidos após a leitura e coleta de dados dos estudos selecionados. Esses resultados servem como base para responder às seis Questões de Pesquisa (QPs) formuladas.

3.4.1 QP.1 Em qual domínio específico as DSLs de teste de software estão direcionadas (por exemplo, aplicações *web*, aplicações móveis, etc.)?

Com o intuito de mapear os domínios das DSLs a serem analisados, a Figura 1 representa os domínios das DSLs por estudo e ano. O gráfico da Figura 1 não indica o número exato de DSLs encontradas; em vez disso, ele destaca os domínios das DSLs por estudo, permitindo analisar a evolução ao longo dos anos em um conjunto de DSLs que abrangem diversos domínios. Com base no gráfico, observa-se uma progressão no número de estudos que propõem ou utilizam DSLs para testes funcionais, evidenciando a relevância do uso dessas linguagens para simplificar o processo de teste de *software* em diversos domínios. Além disso, identificamos 36 domínios diferentes nos quais as DSLs foram aplicadas para auxiliar no processo de teste, sendo que quatro estudos não especificam um domínio correspondente para a aplicação da DSL.

É possível observar que 26,16% (28) dos estudos focalizam o uso de DSLs em aplicações *Web*. O domínio entre plataformas, que oferece suporte a testes em aplicações *Web*, móveis e de desktop, corresponde a 16,82% (18) dos estudos. Adicionalmente, 14,95% (16) dos estudos empregam DSLs capazes de representar todos os domínios de *software*. O domínio de aplicativos Java é abordado em 5,60% (6) dos estudos. No entanto, os aplicativos móveis representam apenas 3,73% (4) dos estudos, sendo que um desses estudos é datado de 2008, indicando que os aplicativos móveis testados não são compatíveis com as tecnologias móveis atuais. Alguns estudos não especificaram um domínio, totalizando 3,73% (4). Por fim, foram categorizados outros 28,97% (31) dos estudos, nos quais foi possível identificar os seguintes domínios: Aplicativos Java, Delphi, Python, C++, C, Scala, Sistemas de direção, Aplicativos *multithread*, Aplicativos orientados a processos, Aplicativos em nuvem, Sistemas IoT, Microsserviços, Programas agregados, Aplicativos de trem, Sistema ciberfísico, Sistemas críticos, Sistemas incorporados, ADAS (Sistemas avançados de assistência ao motorista), Sistemas orientados a eventos, Aplicativos convergentes, Aplicativos orientados a serviços, Serviços da *Web*, ERPs, *E-commerces*, Jogos Android, Procedimentos de medição, Sistemas de rede, Sistemas de informação, Sistemas aviônicos, APIs REST.

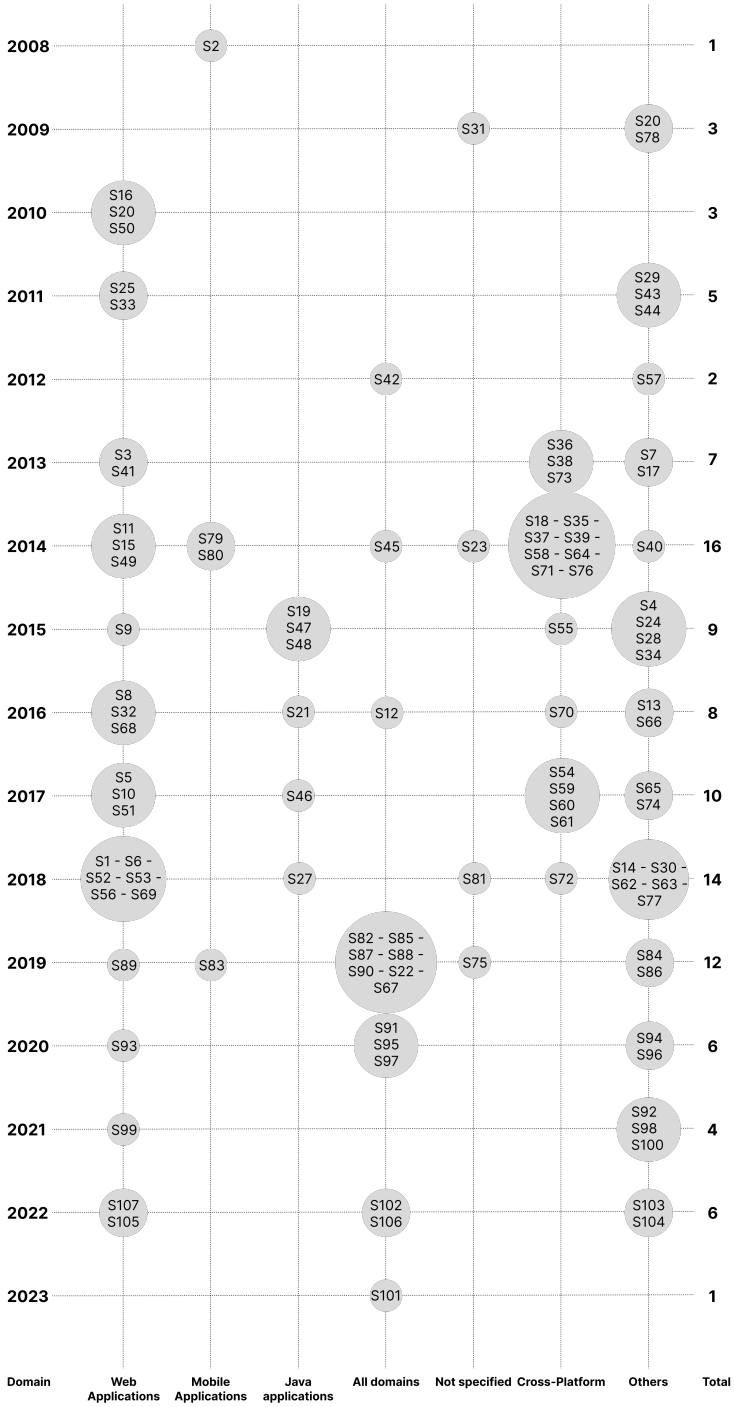


Figura 1 – Gráfico de estudos por domínio entre os anos

Respondendo a QP.1: Em qual domínio específico as DSLs de teste de software estão direcionadas (por exemplo, aplicações web, aplicações móveis, etc.)? *Partindo da análise da Figura 1 revela várias iniciativas no uso de DSLs para testes funcionais de software em diversos domínios. No entanto, observa-se que o foco predominante das atuais DSLs permanece nos domínios de aplicações web e cross-plataforma.*

3.4.2 QP.2 Quais são as funcionalidades disponibilizadas por essas DSLs?

Antes de abordar a resposta a esta pergunta, é crucial destacar que os recursos das DSLs mencionados nesta análise são exclusivos dos estudos e não referentes aos respectivos sites ou ferramentas das DSLs. Essa escolha foi deliberada para focalizar apenas nos recursos avaliados por meio de um método empírico específico. Além disso, categorizamos as DSLs de acordo com os domínios aos quais podem ser aplicadas, reconhecendo que isso não implica que todos os estudos utilizaram a DSL dentro do domínio atribuído, uma vez que algumas DSLs são versáteis e podem ser aplicadas a vários domínios. Para apresentar essas funcionalidades, foi criada a Tabela 6, que inclui o nome da DSL, o domínio associado com suas funcionalidades, e os estudos que referenciaram a linguagem.

Tabela 6 Lista dos domínios e funcionalidades das DSLs por estudo

Nome	Domínio e funcionalidades	Estudos
Gherkin	Domínio: Todos os domínios. As funcionalidades da DSL Gherkin são: descrever o comportamento do software usando a linguagem natural, parametrização de dados, integração com um conjunto de estruturas de automação de testes e pode ser facilmente extensível. A maior parte dos estudos usa o Gherkin para testar aplicativos da *Web* e móveis.	S1, S10, S53, S56, S68, S69, S79, S80, S82, S85, S86, S87, S90, S91, S96, S97, S98, S106
PARADIGM	Domínio: Multiplataforma (*Web*, móvel, *desktop*). A DSL define o comportamento do aplicativo de software usando padrões de interação da Interface Gráfica de Usuário (GUI), execução de casos de teste, geração de relatórios e integração com a ferramenta Sikuli para detectar elementos nas páginas. A maior parte dos estudos usa o PARADIGM para testar aplicativos da Web.	S18, S35, S36, S37, S38, S39, S54, S55, S58, S59, S60, S64, S70, S71, S72, S73, S76
TDL	Domínio: Todos os domínios. A TDL é uma linguagem genérica usada para descrever o comportamento do sistema usando semântica livre escrita em linguagem natural.	S12, S22, S45, S67, S95, S101

Continua na próxima página

Tabela 6 Lista dos domínios e funcionalidades das DSLs por estudo

Nome	Domínio e funcionalidades	Estudos
Sequencer	Domínio: Aplicações Java e Delphi. A DSL Sequencer ajuda a garantir a cobertura fornecendo asserções, definição de entrada de teste e geração de relatórios.	S21, S43, S44, S57
TSL	Domínio: Sistemas de informação. A principal funcionalidade é a transformação da especificação RSL para TSL e a transformação de TSL para Gherkin. Além disso, a linguagem TSL pode ser usada para gerar casos de teste.	S62, S63, S74
ModBat	Domínio: Sistemas de rede, aplicações Java, Scala e C. A principal funcionalidade do ModBat é definir o comportamento do sistema usando transições de uma Máquina de Estados Finitos (FSM), adicionando pré-condições e pós-condições. O FSM definido pelo ModBat é chamado de FSM estendido. Além disso, o ModBat usa o FSM estendido para gerar sequências de teste.	S46, S47, S48
WebSpec	Domínio: Aplicações *Web*. O WebSpec permite que o testador crie diagramas de comportamento e simular o comportamento do aplicativo por meio da integração com o Selenium WebDriver e o JUnit. A simulação pode ser acompanhada por uma interface gráfica.	S25, S26, S50
Legend	Domínio: Aplicações *Web*. As funcionalidades da linguagem Legend incluem a definição, a execução e a depuração de casos de teste. A linguagem também oferece parametrização de dados.	S15, S41

Continua na próxima página

Tabela 6 Lista dos domínios e funcionalidades das DSLs por estudo

Nome	Domínio e funcionalidades	Estudos
Ubtl (UTP-based Test Language)	Domínio: Sistemas críticos. A DSL fornece uma integração de ferramentas de modelagem Linguagem de modelagem Unificada (UML). O Ubtl suporta a integração com várias ferramentas de modelagem UML que podem importar/exportar modelos compatíveis com o Eclipse UML2. Outra funcionalidade é a transformação de modelos: A gramática da Ubtl é transformada automaticamente no meta-modelo, analisador, vinculador e serializador Ubtl. Além disso, também possui integração com o Simulink e o Matlab.	S4, S28
Tiddle	Domínio: Aplicações *Multithreads*. A principal funcionalidade da DSL é descrever o *trace* dos aplicativos e os converter na execução de programas Java concorrentes determinísticos.	S78
SPL-AT Gherkin	Domínio: Linhas de produtos de *software*. Essa linguagem é uma extensão do Gherkin, portanto, é possível definir o comportamento das aplicações usando linguagem natural. Além disso, a linguagem oferece suporte a linhas de produtos de *software* desenvolvidas usando várias linguagens de programação diferentes, para isso, insere a DSL insere em sua sintaxe comportamentos para ser possível definir cenários de teste que sejam mais representativos do domínio da linha de produtos de *software*.	S83

Continua na próxima página

Tabela 6 Lista dos domínios e funcionalidades das DSLs por estudo

Nome	Domínio e funcionalidades	Estudos
TAS DSL	Domínio: Não especificado. A DSL permite definir o Markov Decision Process (MDP) descrevendo as possíveis decisões do sistema e, em seguida, é possível traduzir o modelo MDP textual no arquivo de entrada do verificador de modelos PRISM para testar o sistema.	S81
ScaFi	Domínio: Sistemas agregados. Essa DSL contém as palavras-chave para representar programas agregados, com as quais é possível definir *scripts* de teste para testar programas agregados.	S103
MAJD DSTL e MAJD DSVL	Domínio: Aplicações *Web*. Essas DSLs contêm as respectivas funcionalidades: Descrever cenários de teste e gerar *scripts* de teste com Selenium e JAVA.	S105
Aquila	Domínio: Aplicações *Web*. Essa DSL é uma extensão do Gherkin e, por isso, é possível descrever cenários de teste usando linguagem natural e parametrização de dados como o Gherkin. No entanto, o Aquila suporta o uso da abordagem MBT para gerar *scripts* de teste.	S93
myDSL	Domínio: Todos os domínios. A DSL denominada myDSL pelo respectivo estudo, concentra-se na geração de dados de teste para um contexto específico.	S88

Continua na próxima página

Tabela 6 Lista dos domínios e funcionalidades das DSLs por estudo

Nome	Domínio e funcionalidades	Estudos
TBL	Domínio: Aplicativos móveis. O código do *script* Tbl fornece cerca de 50 instruções que são executadas por um intérprete. As instruções se enquadram em duas categorias: instruções gerais e instruções de teste. A primeira categoria inclui instruções para fluxo de controle, operações aritméticas e relacionais, atribuições e acesso a campos. A segunda categoria inclui instruções para fins de teste, como invocação de drivers de teste, envio e recebimento de eventos.	S2
ATAP DSL	Domínio: Aplicações *Web*. Essa DSL permite que um testador crie *scripts* de teste de automação usando linguagem natural. Essa DSL é semelhante ao Gherkin e permite a geração de código Java usando o Selenium WebDriver. Além disso, o ATAP oferece suporte ao código de edição em tempo de execução, por meio de pausa no *script*.	S5
Claret	Domínio: Sistemas embarcados. As funcionalidades da Claret são: Descrição de casos de uso, pré e pós-condições, fluxos alternativos e fluxos de exceção, geração de testes através de *toolkits*. Além disso, a linguagem fornece, com base nas especificações, a geração de: i) modelos de teste Trivial Graph Format (TGF) para MBT e ii) documentação para comunicação com as partes interessadas.	S14
Photonese	Domínio: Aplicações convergentes. A DSL permite que os testadores definam *scripts* de teste em formato textual.	S20

Continua na próxima página

Tabela 6 Lista dos domínios e funcionalidades das DSLs por estudo

Nome	Domínio e funcionalidades	Estudos
MIDAS	Domínio: Aplicações orientadas a serviços. O MIDAS DSL é dividido em várias partes conceituais, incluindo planejamento de teste, projeto de teste, dados de teste e programação de testes. Além disso, o MIDAS DSL permite a geração de *scripts* de teste TTCN-3 executáveis a partir de modelos baseados em UML Testing Profile (UTP).	S24
TSTL (Template Scripting Testing Language)	Domínio: Aplicações Python e Java. O TSTL suporta a construção de cláusulas SQL de comprimento arbitrário. O TSTL inclui um gerador de testes randômicos que pode ser usado para testar qualquer SUT com uma cobertura definida através do TSTL. O TSTL fornece relatórios contínuos de cobertura de ramificação incremental. Além disso, a linguagem também oferece suporte à depuração delta, uma técnica para reduzir os casos de teste com falha. Por fim, a DSL inclui pós-processamento adicional específico que reduz o tamanho e a complexidade dos casos de teste.	S27
CTy	Domínio: *Web Services*. O CTy fornece um *Web Service* de teste combinatório, cria oráculos de testes e define partições de equivalência. Além disso, também é possível utilizar métodos Haskell nativos.	S29

Continua na próxima página

Tabela 6 Lista dos domínios e funcionalidades das DSLs por estudo

Nome	Domínio e funcionalidades	Estudos
Yest	Domínio: ERPs e aplicativos de negócios. O Yest DSL oferece suporte à geração de testes automatizados. Isso significa que ele pode criar automaticamente cenários de teste com base em critérios e entradas especificadas. O Yest DSL permite a criação interativa de cenários de teste. Além disso, o Yest oferece recursos de preenchimento automático que auxiliam os testadores na criação de cada etapa do cenário de teste. Por fim, o Yest pode ajudar os testadores a rastrear a cobertura dos requisitos por meio de modelos durante a criação dos cenários.	S30
Natural-language requirements DSL	Domínio: DSLs. A DSL permite especificar requisitos em linguagem natural por meio de cenários. Além disso, é possível gerar casos de teste executáveis.	S40
OSMOTester MBT	Domínio: Todos os domínios. Funcionalidades: Gerar testes a partir do modelo de teste genérico, restringir a geração de casos de teste variados para um ou mais cenários específicos ou criar manualmente casos de teste específicos a partir do modelo e gerar dados de teste.	S42
TestMind	Domínio: Aplicações *Web*. A DSL pode ser usada de forma colaborativa por meio do Fitnesse e o TestMind DSL gera um *scripts* de teste que podem ser salvos como páginas de teste do Fitnesse, em conjunto com as especificações de teste e capturas de tela.	S51
GUI Specification Language	Domínio: Multiplataforma (*Web*, *mobile*, *desktop*). A linguagem de especificação de GUI tem como objetivo capturar comportamentos da interface gráfica posicionais, aritméticos, lógicos e relacionais.	S61

Continua na próxima página

Tabela 6 Lista dos domínios e funcionalidades das DSLs por estudo

Nome	Domínio e funcionalidades	Estudos
RSL	Domínio: Não especificado. É possível gerar testes funcionais a partir da especificação RSL.	S75
ATA21 e ATA42	Domínio: Sistemas aviônicos. Com a DSL, é possível especificar o comportamento de um aspecto específico do sistema aviônico e gerar códigos na linguagem Pivot. Além disso, a linguagem Pivot gera código de execução em Python. (Usando transformações de modelo para modelo).	S77
S102 DSL	Domínio: Todos os domínios. Funcionalidades: Gerar casos de teste com base na especificação TDL e gerar novos dados de entrada para os testes.	S102
S100 DSL	Domínio: Aplicativos orientados por processos. Funcionalidades: Descrever casos de teste usando uma sintaxe semelhante à notação Notação de modelagem de processos de negócio (BPMN) e gerar testes automatizados.	S100
S94 DSL	Domínio: Aplicações Python. A principal funcionalidade dessa DSL é especificar modelos de falha (especificando como o código-fonte deve ser alterado para introduzir *bugs* de *software*, informando o padrão de código e a sua substituição), de modo que seja possível testar aplicativos Python usando testes de mutação.	S94
S99 DSL	Domínio: Aplicações *Web*. Com essa DSL, é possível descrever *scripts* de cenários de teste sem incluir seletores de elementos *Web*. Os elementos *Web* em páginas HTML podem ser pesquisados diretamente pelo *script* da DSL. Além disso, o código da DSL pode ser lido por não desenvolvedores.	S99

Continua na próxima página

Tabela 6 Lista dos domínios e funcionalidades das DSLs por estudo

Nome	Domínio e funcionalidades	Estudos
S107 DSL	Domínio: Aplicações *Web*. Com essa DSL, é possível descrever cenários de teste.	S107
S84 DSL	Domínio: Sistemas de direção. Essa DSL suporta a importação de cenários e o testador pode definir cenários de teste e gerar casos de teste automatizados.	S84
S89 DSL	Domínio: Aplicações *Web*. A principal funcionalidade dessa DSL é definir o modelo e o comportamento do aplicativo e, por meio do modelo, é possível gerar casos de teste.	S89
S3 DSL	Domínio: Aplicações *Web*. Essa DSL modela a estrutura de uma aplicação *Web*, expressa dependências entre fluxos de trabalho no modelo e também é possível incluir dados de teste externos de referência (usando XML) e abstrações de dados de especificação de implementação, por exemplo, identificadores HyperText Markup Language (HTML).	S3
S6 DSL	Domínio: Aplicações *Web*. A DSL permite a criação de casos de teste automatizados com ou sem dados parametrizados usando linguagem natural.	S6
S7 DSL	Domínio: Aplicações C++. Funcionalidades: Encontrar um vazamento de memória difícil em um programa C++, NOMAD, Verificar um ponto matemático crucial dentro do NOMAD, encontrar possíveis erros de interferência nas especificações do NOMAD e DSL diretamente no código-fonte, sem afetar a compilação.	S7
S8 DSL	Domínio: Aplicações *Web*. As funcionalidades apresentadas neste estudo foram a criação de casos de teste para futuros testadores e a integração com extensões para facilitar a criação de testes.	S8

Continua na próxima página

Tabela 6 Lista dos domínios e funcionalidades das DSLs por estudo

Nome	Domínio e funcionalidades	Estudos
S9 DSL	Domínio: Aplicações *Web*. Funcionalidades: Criar projetos de teste, descrever os comportamentos do SUT, Descrever o contexto para o teste, criar cenários de teste, a parametrizar dados de teste e a gerar de *scripts* de teste.	S9
S11 DSL	Domínio: Aplicações *Web*. A DSL apresentada permite que os usuários possam escrever e gerar *scripts* de teste de forma declarativa.	S11
S13 DSL	Domínio: ADAS (Sistemas Avançados de Assistência ao Motorista). Essa DSL textual pode descrever casos de teste com base na sintaxe e na semântica do catálogo de situações e nos requisitos do sistema.	S13
S16 DSL	Domínio: Aplicações *Web*. A DSL permite a criação de *scripts* de teste por meio de uma especificação textual baseada em JAVA.	S16
S17 DSL	Domínio: Sistemas orientados a eventos. A DSL oferece suporte ao não determinismo tanto na especificação (para simular falhas ou eventos não determinísticos) quanto no tratamento do comportamento do SUT resultante (para tratar falhas ou exceções). Além disso, a DSL permite a inserção de pré-condições.	S17
S19 DSL	Domínio: Aplicações Java. A DSL permite definir *scripts* de teste por meio de ações. As ações definidas pela DSL e podem ser agrupadas para criar eventos mais avançados.	S19
S23 DSL	Domínio: Não especificado. Funcionalidades: Gerar relatório de atividades de teste, descrição de pré-condições e pós-condições. Além disso, também é possível descrever fluxos de teste.	S23

Continua na próxima página

Tabela 6 Lista dos domínios e funcionalidades das DSLs por estudo

Nome	Domínio e funcionalidades	Estudos
S31 DSL	Domínio: Não especificado. A DSL fornece um ambiente de modelagem genérico e integrado, que oferece uma plataforma abrangente para a criação e a edição de testes. Os usuários podem contar com recursos como o preenchimento automático para ajudar no processo de edição de testes, tornando-o mais eficiente. A DSL oferece um recurso de análise de impacto. Por fim, a linguagem também oferece a geração de relatórios.	S31
S32 DSL	Domínio: Aplicações *Web*. A DSL é uma extensão do Gherkin e implementa uma abordagem minimalista para Desenvolvimento direcionado a comportamento (BDD), como palavras-chave, 'given', 'when' e 'then' para expressar testes no formato 'Given [contexto inicial], when [evento ocorrido], then[garantir saídas]'. O testador é apoiado por novos conceitos como "open", "go to", "fill in" e outros.	S32
S33 DSL	Domínio: Aplicações *Web*. Funcionalidades: geração de oráculos de teste a partir de informações do modelo e a transformação de casos de teste abstratos em *scripts* de teste executáveis.	S33
S34 DSL	Domínio: Comércios eletrônico, jogos para Android e aplicações *Web*. Este estudo apresenta duas linguagens. Ambas são uma extensão do Gherkin e fornecem construções para definir vários cenários que, por sua vez, consistem em várias etapas, sendo cada uma delas uma pré-condição para o cenário (Given), seguida de um estímulo para o sistema (When) e, finalmente, a pós-condição (Then).	S34

Continua na próxima página

Tabela 6 Lista dos domínios e funcionalidades das DSLs por estudo

Nome	Domínio e funcionalidades	Estudos
S49 DSL	Domínio: Aplicações *Web*. A linguagem permite que o testador especifique testes e gere novos testes.	S49
S52 DSL	Domínio: Aplicações *Web*. A DSL da suporte aos testadores na criação de cenários de teste.	S52
S65 DSL	Domínio: Sistemas aviônicos. A DSL fornece ao testador um método para especificar requisitos de alto nível.	S65
S66 DSL	Domínio: Sistemas aviônicos. Funcionalidades: Gerar a estrutura dos testes de unidade e testes de integração. Além disso, é possível utilizar a parte de automação dos procedimentos e a integração com restrições *Object Constraint Language* (OCL).	S66
S92 DSL	Domínio: Sistemas IoT. Este estudo apresenta duas DSLs: A DSL1 é usada para definir o objeto de dados a ser usado nos testes e a DSL2 é usada para definir requisitos/condições. Com essas DSLs, é possível gerar casos de teste.	S92
S104 DSL	Domínio: Sistema ciberfísico e solicitações HTTP usando uma API REST. Essa DSL permite que o testador defina e execute casos de teste.	S104
		Final da tabela

Após uma análise abrangente das DSLs, destaca-se que a linguagem Gherkin se destaca como a mais utilizada para testes funcionais, com ênfase especial na descrição e parametrização de testes, sendo estes os recursos mais comumente implementados nas DSLs. Os domínios da *Web* e de dispositivos móveis emergem como os principais campos nos quais se busca aprimorar a qualidade por meio do emprego de DSLs. A adoção de MBT surge como uma estratégia para ampliar a cobertura de testes, entretanto, conforme indicam os estudos, sua aplicação não é tão simples quanto a descrição de testes. Essa complexidade pode explicar a redução do uso por parte dos testadores, possivelmente contribuindo para a predominância do emprego de linguagens naturais. A garantia de cobertura de teste permanece como um desafio significativo, com tentativas de utilização

de verificadores de modelos, MBT e outras abordagens, embora DSLs que incorporam essas práticas apareçam repetidamente nos estudos. Curiosamente, nenhum estudo destaca práticas recomendadas para DSLs, sugerindo uma falta de consenso entre os testadores sobre as melhores formas de aplicar ou utilizar essas linguagens para obter o máximo desempenho.

Respondendo a QP.2 Quais são as funcionalidades disponibilizadas por essas DSLs? *As DSLs abrangem uma variedade extensa de funcionalidades, destacando-se, entre elas, a descrição de cenários de teste, a geração de casos de teste, a parametrização de dados, a geração de dados de teste e a análise de cobertura. É importante notar que nem todas as DSLs incorporam todas essas funcionalidades, mas esses elementos representam um escopo significativo das características existentes dentro desse domínio de linguagens específicas.*

3.4.3 QP.3 Quais tecnologias ou notações são prevalentes no desenvolvimento das DSLs?

A implementação de uma DSL não é uma tarefa trivial, sendo crucial compreender as tecnologias empregadas nesse processo. Desenvolver uma DSL requer a implementação de elementos como compilador, sintaxe, analisador, entre outros. Para simplificar esse desenvolvimento, os desenvolvedores podem optar por utilizar linguagens de propósito geral, como Java, Python e Ruby, para incorporar a sua linguagem específica. Outra abordagem é fazer uso de um LW, que gera automaticamente compiladores e analisadores se o desenvolvedor definir uma sintaxe. Na Tabela 7, é possível observar que os estudos selecionados neste SMS empregam diversas tecnologias para a implementação de suas respectivas DSLs. Vale ressaltar que um estudo pode adotar uma ou mais tecnologias para desenvolver sua DSL, e alguns estudos podem não detalhar as tecnologias utilizadas na implementação da linguagem específica.

Tabela 7 Lista de tecnologias mencionadas para desenvolver DSLs por estudos.

Tecnologias	**Qtd.**	**Estudos**
Xtext	23	S1, S3, S4, S5, S12, S22, S28, S33, S34, S40, S45, S49, S62, S63, S65, S67, S74, S75, S81, S88, S89, S95, S100
Eclipse Modeling Framework (EMF)	12	S9, S22, S25, S26, S36, S45,S50, S54, S55, S58, S95, S102
Xtend	08	S3, S4, S5, S28, S67, S68, S81, S89

Continua na próxima página

Tabela 7 Lista de tecnologias mencionadas para desenvolver DSLs por estudos.

Tecnologias	**Qtd.**	**Estudos**
Graphical Modeling Framework (GMF)	07	S9, S22, S25, S26, S38, S39, S50
Java	06	S4, S16, S42, S81, S88, S105
Scala	05	S17, S46, S47, S48, S103
Sirius	03	S12, S22, S95
Meta Programming System (MPS)	03	S52, S66, S84
JUnit	03	S16, S21, S34
C#	02	S06, S41
ANTLR	02	S61, 104
Aspect C++	01	S7
Haskell	02	S29, S78
Object Constraint Language (OCL)	02	S12, S22
Selenium	02	S18, S34
Sikuli API	02	S18, S34
Docx4j	02	S12, S22
UML Profiles	02	S23, S24
Extended Backus-Naur Form (EBNF)	01	S14
DLex	01	S44
RSL	01	S63
Ruby	01	S20
XML	01	S11
AspectJ	01	S81
XMI	01	S22
Epsilon Validation Language (EVL)	01	S22
Papyrus	01	S12
Robot Framework	01	S12
SelenDroid	01	S34
HTMLUnit	01	S16
DUnit	01	S21
Próprio ambiente de desenvolvimento	01	S8

Final da tabela

Com base na Tabela 7, destaca-se o amplo uso do ambiente Eclipse pelos desenvolvedores na criação de novas linguagens para testes funcionais, por meio de ferramentas como Xtext, Xtend, *Eclipse Modeling Framework (EMF)*, *Graphical Modeling Framework (GMF)* e Sirius. O principal benefício dessas soluções reside na capacidade do desenvolvedor de conceber a linguagem com uma sintaxe livre, sem ficar restrito à sintaxe de

outra linguagem. Outro ambiente relevante é o MPS, que compartilha uma finalidade semelhante, mas opera no ambiente JetBrains.

Na sequência das Language Workbenches, surgem as linguagens incorporadas, fazendo uso de tecnologias como Java, C#, Scala e Haskell. A principal vantagem ao utilizar uma linguagem de uso geral para desenvolver uma DSL reside na possibilidade dos usuários empregarem seu conhecimento prévio da linguagem para assimilar a DSL. Embora outras tecnologias e bibliotecas possam ser aplicadas para aprimorar o tempo de aprendizado da linguagem, é crucial entender as nuances dos usuários do domínio-alvo. Por fim, um conjunto diversificado de tecnologias, incluindo bibliotecas e formatos de arquivo, é empregado por DSLs para aprimorar a eficácia dos testes funcionais.

Respondendo a QP.3 Quais tecnologias ou notações são prevalentes no desenvolvimento das DSLs? *Dentro do conjunto de estudos mapeados, observa-se que nenhuma DSL foi implementada de forma isolada, sem o auxílio de outras tecnologias ou LWs. Assim, destaca-se a prevalência e a recomendação do uso de LWs para o desenvolvimento de DSLs. Em alguns casos específicos, a utilização de GPLs pode ser vantajosa, principalmente para capitalizar o conhecimento da equipe durante a fase de aprendizado. Considerando esses aspectos, é notável que o ambiente Eclipse composto por ferramentas como Xtext, GMF e EMF, figura como um dos conjuntos mais amplamente empregados na atualidade.*

3.4.4 QP.4 Quais técnicas, abordagens ou metodologias as DSLs empregam para assegurar uma cobertura abrangente nos testes gerados?

A maioria das linguagens identificadas não se concentra na cobertura do sistema, optando por gerar testes com base em métodos específicos para essa finalidade. O objetivo primordial da maioria das DSLs é simplificar a criação de scripts de teste e suas definições, visando a otimização do tempo necessário para essa tarefa ou o aprimoramento da compreensão dos cenários de teste pelos usuários do domínio. Contudo, alguns estudos buscam aprimorar a cobertura do teste, seja por meio da integração direta com a DSL ou utilizando outra ferramenta mediante integração. A lista completa das soluções está apresentada na Tabela 8. A ausência de um estudo na tabela indica que o mesmo não oferece uma solução específica para a cobertura.

Tabela 8 Solução para garantia de cobertura por DSL.

Nome	Solução
S102 DSL	Este estudo apresenta um método de amplificação de teste baseado nas descrições da DSL.

Continua na próxima página

Tabela 8 Solução para garantia de cobertura por DSL.

Nome	**Solução**
Gherkin (S80)	O estudo usa a rede Petri e os modelos UML com Gherkin para melhorar a cobertura.
Gherkin (S106)	O estudo propõe a geração de cenários Gherkin usando modelos BPMN.
Gherkin (S96)	Neste estudo, é apresentada a possibilidade de gerar caminhos usando uma FSM.
TAS DSL	A TAS DSL usa o processo de decisão Markov para garantir a cobertura do teste.
S92 DSLs	Este estudo apresenta duas DSLs e os casos de teste são gerados pelas restrições da DSL2 com base em requisitos/condições.
Gherkin (S86)	Neste estudo, é possível gerar testes Gherkin com base em um grafo.
Aquila	O modelo escolhido para o uso do MBT, neste trabalho, é o Grafo acíclico dirigido (DAG). A extensão transforma a especificação Gherkin em nós, que são usados por um algoritmo de busca em profundidade Depth-first search (DFS) para gerar sequências.
myDSL (S88)	A DSL garante a cobertura do sistema por meio da geração de dados.
S89 DSL	A DSL usa o Processo de Decisão de Markov para garantir a cobertura do teste.
S3 DSL	A DSL permite o uso do verificador de modelo NuSMV. O testador pode escolher a meta de cobertura por meio da ferramenta.
S13 DSL	A DSL melhora a cobertura ao gerar dados por padrões definidos diretamente no *script* da DSL.
Claret	A DSL tem uma integração com a ferramenta LTS-BT e, com ela, é possível gerar casos de teste extraindo caminhos de modelos usando uma pesquisa DFS.
PARADIGM	Com o uso da PARADIGM DSL, é possível usar algoritmos de profundidade para gerar sequências de teste para garantir a cobertura.
Legend (S15)	A DSL fornece a relação entre a história do usuário e o *script* da DSL para rastreabilidade, com a qual o engenheiro de testes pode ver o status da cobertura.
MIDAS	A DSL combina especificamente execução simbólica e técnicas de verificação baseadas em redes de Petri para geração de um oráculo e casos de teste.
TSTL	O DSL pode ser usado com o verificador de modelos SPIN para verificar o modelo descrito usando o TSTL.

Continua na próxima página

Tabela 8 Solução para garantia de cobertura por DSL.

Nome	**Solução**
CTy	Dada uma árvore de classificação, a DSL pode gerar seu conjunto completo, que consiste em todos os casos de teste possíveis induzidos por ela. Ela também pode gerar seu conjunto minimalista, que é o menor conjunto em que cada classe é coberta pelo menos uma vez.
Yest	A DSL gera testes automatizados com base em uma tabela de decisão que usa todas as linhas da tabela para cobrir os testes do sistema.
S33 DSL	A DSL usa uma pesquisa Breadth-first search (BFS) estendida para explorar esse gráfico a partir da visualização inicial designada e gerar sequências de teste.
Legend (S41)	Essa DSL pode ser usada com o *framework* Echo para gerar *scripts* de teste. O EchoGenerator tem dois modos de geração: *Script* - gera *scripts* de teste escritos na sintaxe do Echo, e Assembly - gera uma representação da Linguagem Intermediária Comum (CIL) do teste Echo. A integração da DSL mapeia a CIL para as etapas de teste específicas do domínio.
OSMOTester MBT	O OSMOTester inclui os seguintes algoritmos a serem executados com base em seus respectivos modelos: Aleatório: escolhe uma transição aleatória dentre as disponíveis, Equilíbrio: escolhe aleatoriamente uma transição disponível, mas favorece as menos cobertas, Ponderado: escolhe aleatoriamente uma transição disponível, mas dá maior probabilidade àquelas com maior peso.
ModBat	A DSL usa a FSM estendida para gerar casos de teste. O estudo usa abordagens aleatórias e estocásticas para gerar testes usando métodos aleatórios e baseados em pesquisa na geração de testes.
S49 DSL	Essa DSL usa o diagrama de atividades para gerar testes.
WebSpec	A DSL gera casos de teste com base nos diagramas especificados por sua sintaxe.
TSL	A DSL usa uma máquina de estado e também usa a cobertura Switch-0 (o que significa que a DSL verifica todas as transições diretas possíveis no modelo).
S65 DSL	Usando as especificações da DSL, é possível definir condições de teste, e a DSL gera casos de teste para usar todas as condições definidas.

Continua na próxima página

Tabela 8 Solução para garantia de cobertura por DSL.

Nome	Solução
S66 DSL	A DSL gera uma matriz de rastreabilidade entre os objetivos e os procedimentos de teste.
TDL (S67)	É possível usar uma geração automatizada de descrições de teste TDL a partir de requisitos expressos como modelos de cenário UCM usando a ferramenta jUCMNav. Essa transformação permite a exploração de testes baseados em modelos com a linguagem TDL.

Final da tabela

Algumas das DSLs listadas na Tabela 8 estão duplicadas, sendo incluídas junto com seus respectivos estudos para aprimorar a rastreabilidade. Essa duplicidade ocorre devido à apresentação de diferentes soluções de cobertura de teste em estudos distintos. Algumas DSLs não aprimoram a cobertura de teste por padrão, exigindo o uso de outras ferramentas ou abordagens para tal. Entretanto, ao analisar a Tabela 8, torna-se evidente que um conjunto específico de DSLs oferece suporte à garantia da cobertura do sistema, predominantemente utilizando abordagens de MBT, como algoritmos de grafo, para gerar casos de teste. Adicionalmente, todas as DSLs destacadas na Tabela 8 estão centradas na criação de mais casos de teste com base na especificação da DSL, algumas vezes empregando abordagens de MBT e, em outras ocasiões, utilizando a geração de dados para aprimorar os cenários de teste. Em apenas dois estudos, a DSL aprimora a cobertura de teste do sistema por meio da rastreabilidade entre requisitos e casos de teste definidos pelo engenheiro de teste.

Respondendo a QP.4 Quais técnicas, abordagens ou metodologias as DSLs empregam para assegurar uma cobertura abrangente nos testes gerados? *A maioria das DSLs não oferece uma solução para garantir a cobertura de testes dos SUTs. No entanto, a geração de casos de teste é o método mais comumente empregado entre as DSLs, frequentemente respaldado pelo uso de algoritmos de grafo. A rastreabilidade entre funcionalidades e testes também surgiu como suporte à cobertura, embora em um número limitado de estudos.*

3.4.5 QP.5 De que maneira os dados do SUT são representados nas DSLs?

Com o intuito de responder a essa pergunta, todos os métodos de inserção de dados do SUT nas DSLs foram analisados por meio dos estudos identificados pelo SMS, e essas DSLs foram categorizadas em três métodos distintos. O primeiro método envolve a inserção direta de dados no script ou modelo de teste, onde o engenheiro de teste incorpora os dados junto com a etapa de teste. Esse método é mais fácil de compreender

em comparação com os outros, mas, como o caso de teste não é parametrizado, é necessário um grande número de casos de teste para abranger todos os cenários do SUT. O segundo método utiliza a parametrização de dados, permitindo definir um conjunto de dados a ser utilizado pelo caso de teste. A parametrização pode ocorrer em conjunto com outro arquivo ou dentro do script DSL. Com esse método, a quantidade de casos de teste é menor em comparação com o primeiro método, mas o entendimento do caso de teste pode ser mais desafiador. Por fim, o terceiro método faz uso da geração de dados, possibilitando a criação de dados de teste com base nos requisitos definidos previamente na DSL ou em outra ferramenta integrada a ela. Esse método é indicado para gerar casos de teste rapidamente, embora seu uso não seja trivial e demande mais lógica de programação do que os outros métodos. Além disso, a aplicação de cada abordagem por papel pode ser compreendida por meio da Tabela 9 e do diagrama de Venn presente na Figura 2.

Tabela 9 Lista de tipos de entrada de dados por estudo

Tipo	**Qtd.**	**Estudos**
Diretamente no *script*/modelo	85	S69 S26 S33 S11 S107 S19 S65 S106 S3 S101 S95 S86 S14 S79 S60 S8 S83 S5 S97 S16 S36 S4 S94 S40 S51 S32 S41 S21 S7 S52 S31 S98 S77 S13 S30 S53 S50 S43 S100 S20 S78 S23 S71 S64 S49 S105 S22 S73 S68 S47 S99 S37 S66 S57 S85 S74 S59 S58 S27 S67 S44 S81 S84 S28 S76 S39 S75 S2 S48 S45 S80 S90 S55 S18 S54 S12 S87 S61 S17 S104 S91 S89 S35 S38
Geração de dados	4	S29 S92 S88 S102 S42
Parametrização de dados	2	S63 S96
Diretamente no *script*/modelo e Parametrização de dados	8	S1 S93 S9 S6 S15 S10 S56 S34
Diretamente no *script*/modelo e Geração de dados	2	S25 S62
Diretamente no *script*/modelo e Geração de dados e Parametrização de dados	1	S24
Não identificado	5	S103 S70 S72 S46 S82

Final da tabela

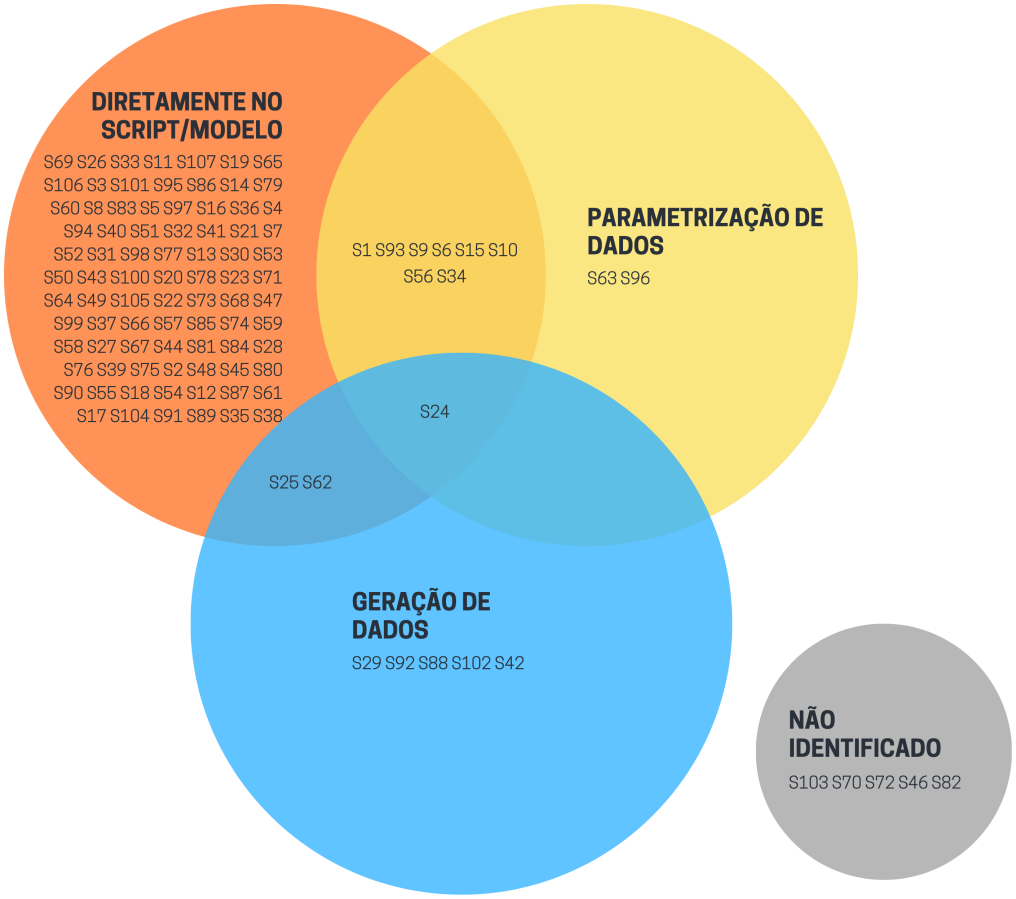


Figura 2 – Diagrama Venn representando os tipos de inserção de dados nos SUTs por estudo

> ***Respondendo a QP.5 De que maneira os dados do SUT são representados nas DSLs?*** *A modalidade mais prevalente para representar os dados do SUT é a inserção direta nos scripts ou modelos de testes das linguagens. No entanto, algumas DSLs também oferecem opções como geração e/ou parametrização de dados.*

3.4.6 QP.6 Quais benefícios, desvantagens e tendências estão associados ao desenvolvimento e adoção de DSLs para testes funcionais?

A análise dos estudos revelou que a adoção de DSLs proporciona benefícios significativos na fase de testes de *software*. Com base nesses estudos, destacam-se os seguintes benefícios associados à implementação de DSLs para testes funcionais: 1) Aumento da legibilidade dos *scripts* de teste. 2) Facilidade de envolver especialistas no domínio durante a fase de teste. 3) Redução da complexidade dos códigos de teste e diminuição do tempo necessário para a execução dos testes. 4) Prevenção de erros relacionados à ortografia ou compreensão inadequada dos requisitos. 5) Adoção de práticas como desenvolvimento orientado por testes e desenvolvimento orientado por comportamento. 6) Possibilidade de desenvolver testes em paralelo ou antes da implementação do aplicativo. 7) Facilidade

na manutenção dos *scripts* de teste. 8) Melhoria da cobertura de teste do aplicativo por meio da geração de casos de teste. 9) Curva de aprendizado reduzida para especialistas no domínio ou engenheiros de teste. 10) Capacidade de descrever testes usando linguagem natural ou gráfica. 11) Integração facilitada com outras ferramentas.

Apesar dos benefícios destacados, as DSLs para testes funcionais de *software* não estão isentas de desafios, conforme evidenciado pelos estudos analisados, que apontam as seguintes desvantagens associadas às DSLs para testes funcionais: 1) Ausência de regras de validação em algumas DSLs. 2) Limitação do tamanho dos modelos ou *scripts*, especialmente em linguagens em que a verificação do *software* é realizada por ferramentas externas. 3) Dificuldade em verificar os *scripts* de teste gerados, crucial para determinar se eventuais erros estão relacionados à implementação inadequada do *script* automatizado ou a falhas no sistema. 4) Restrição em linguagens que garantem cobertura, impedindo a definição de caminhos específicos dentro da DSL. 5) Limitada capacidade de reutilização de componentes em algumas DSLs, aumentando o esforço necessário para modelar o sistema. 6) Exigência de conhecimento em métodos formais, como FSM, em algumas linguagens.

É relevante observar que nem todas as DSLs apresentam essas desvantagens ou os benefícios mencionados anteriormente. A análise detalhada de cada DSL, considerando o contexto específico de domínio, é crucial, uma vez que algumas dessas desvantagens podem não ser verdadeiramente desvantajosas, dependendo da equipe de engenheiros de teste da empresa ou do grupo.

Por fim, observam-se tendências significativas nas DSLs voltadas para testes de *software*, destacando-se: 1) Realização de avaliações empíricas nas linguagens: Muitas DSLs carecem de avaliações empíricas, limitando sua aplicação no cenário real de desenvolvimento. 2) Utilização de outra linguagem de *script* de suporte: Algumas abordagens buscam mitigar os problemas associados às DSLs ao integrar suporte a outras linguagens de *script*, como a integração com o Javascript do navegador em DSLs voltadas para aplicativos da *Web*. 3) Inclusão da possibilidade de definir caminhos específicos: Linguagens direcionadas à garantia de cobertura por meio da geração de casos de teste estão explorando a inclusão de recursos para que os testadores possam especificar bugs recorrentes e prováveis, incorporando o conhecimento do testador. 4) Abordagens de teste baseadas em modelos: Algumas tendências indicam a adoção de abordagens de teste baseadas em modelos para oferecer suporte mais eficiente aos engenheiros de teste. 5) Geração de casos de teste e dados de entrada: A tendência é fortalecer a capacidade de geração automática de casos de teste e dados de entrada por meio das DSLs. 6) Suporte a mudanças rápidas relacionadas à SUT: Tornar mais fácil a atualização simplificada dos *scripts* DSL em resposta a alterações rápidas na SUT é uma área de foco. 7) Suporte a vários domínios com a mesma linguagem: Exploração de linguagens ou estruturas semelhantes capazes de oferecer suporte a diversos domínios, proporcionando flexibilidade na aplicação das

DSLs. 8) Elaboração de diretrizes para o uso: Há uma crescente ênfase na elaboração de diretrizes para o uso dessas linguagens, facilitando a adoção e implementação por parte de terceiros.

Respondendo a QP.6 Quais benefícios, desvantagens e tendências estão associados ao desenvolvimento e adoção de DSLs para testes funcionais? *Diversos benefícios, limitações e tendências foram previamente discutidas. Entre os benefícios, destaca-se a facilidade de aprendizado e a redução do esforço da equipe de testes na criação de cenários/scripts de testes automatizados. Em relação às limitações, observa-se a falta de flexibilidade de algumas linguagens para oferecer mais de uma maneira de especificar os casos de teste. Por exemplo, é desafiador viabilizar tanto a descrição detalhada dos cenários quanto a geração automatizada. No contexto das tendências, é relevante destacar o movimento em direção ao suporte a diversos domínios por meio da mesma linguagem ou por meio de atualizações simples. Essa abordagem visa proporcionar uma maior adaptabilidade e versatilidade às DSLs no campo dos testes funcionais.*

3.4.7 Ameaças à validade

Esta seção aborda as ameaças à validade deste estudo, bem como as estratégias adotadas para mitigá-las, seguindo as diretrizes de (WOHLIN et al., 2012).

Validade da Construção: A principal ameaça à construção reside na possível exclusão de estudos relevantes. Para mitigar essa preocupação, foi conduzido um processo de seleção e extração de dados com a participação de dois pesquisadores distintos, seguindo critérios de inclusão e exclusão uniformes. Visando ampliar a abrangência, utilizamos seis base de dados científicos distintos. Além disso, para enfrentar a possibilidade de as palavras-chave não representarem completamente o campo, foram realizados vários testes preliminares antes da execução efetiva da pesquisa. Foi realizado também uma abordagem de *Snowballing* após o primeiro período, utilizando os estudos identificados como sementes para expandir as buscas em períodos anteriores e posteriores. No entanto, mesmo com um número considerável de estudos selecionados, não podemos garantir a inclusão de todos os estudos relevantes nesse conjunto de resultados. Por fim, no segundo período, optamos por realizar apenas o SMS, uma busca automatizada, deixando de lado outras estratégias de busca, como a *Snowballing* aplicada no primeiro período e buscas manuais.

Validade Interna: A análise dos estudos selecionados suscita a preocupação com a validade interna. Uma ameaça significativa é o viés de publicação, que poderia incluir estudos de qualidade inferior, contrariando o objetivo de pesquisa. Para mitigar esse risco, foram aplicadas classificações a todos os estudos por meio de perguntas de qualidade. Vale notar que nenhum estudo recebeu uma pontuação inferior a um, e, portanto, nenhum estudo foi excluído com base nos critérios de qualidade estabelecidos.

Outra ameaça a ser considerada é a possível falta de representatividade estatística na quantidade de resultados obtidos. Para enfrentar essa preocupação, empregamos a técnica de *snowballing* após o primeiro período e repetimos o protocolo SMS em 2023, visando capturar uma gama mais abrangente de estudos representativos. Por fim, a classificação incorreta dos estudos também é uma ameaça potencial. Para lidar com essa questão, optamos por utilizar uma abordagem mais simplificada em algumas análises, adotando a perspectiva do estudo. Isso se deu devido à ausência de recursos específicos sobre DSLs ou de uma taxonomia de tipos de dados de entrada em outros estudos, o que dificultou a classificação precisa desses dados.

Validade Externa: Uma das ameaças externas identificadas foi a possível ausência de estudos relevantes. Para enfrentar esse desafio, foram realizadas buscas manuais no banco de dados do *Google Scholar*, ampliando nossa abrangência na identificação de estudos. Quando as informações essenciais sobre o uso de DSLs não estavam claramente indicadas nos resumos, procedemos à busca em outras seções dos estudos, como conclusão e introdução, para determinar sua inclusão adequada. Por fim, vale ressaltar que pode haver DSLs adicionais destinadas a apoiar testes funcionais que não foram identificadas, possivelmente devido à falta de conhecimento do autor sobre o conceito de DSL, levando a uma não descrição explícita de sua solução como uma DSL.

Validade da Conclusão: A ameaça à validade das conclusões apresentadas na última seção é uma preocupação que exigiu estratégias específicas para mitigação. Para evitar qualquer viés de conclusão, seguimos rigorosamente o protocolo de pesquisa proposto por (PETERSEN et al., 2008), uma metodologia amplamente reconhecida para a condução de estudos de mapeamento sistemático. Outra ameaça abordada é o potencial viés estatístico decorrente da quantidade limitada de estudos selecionados. Para mitigar esse problema, ampliamos nossa busca para incluir seis base de dados, empregamos *strings* de pesquisa expressivas adaptadas para cada base e realizamos uma análise através de um *snowballing* após o primeiro período, além de uma execução do SMS segundo período em 2023. Adicionalmente, adotamos uma abordagem prudente ao tirar conclusões, aguardando a coleta completa dos resultados, o que contribui para evitar possíveis problemas de pesca e taxa de erro (WOHLIN et al., 2012).

3.5 Outras revisões

Vários estudos de Mapeamento Sistemático de Literatura (SMS) foram conduzidos para elucidar o panorama das *Domain-Specific Languages* (DSLs) e suas aplicações. Embora o escopo desses estudos de SMS possa não ter se restringido exclusivamente a DSLs para teste de software funcional, eles têm contribuído para uma compreensão abrangente das tendências e aplicações de pesquisa em DSL.

Em um SMS conduzido por (KOSAR; BOHRA; MERNIK, 2016), os autores tinham como propósito obter *insights* sobre o campo de pesquisa de DSLs no período de

2006 a 2012. O estudo abrangeu mais de mil estudos primários provenientes de fontes renomadas, como a *ISI Web of Science* e a *ACM Digital Library*. Seus principais objetivos eram delinear o panorama de pesquisa em DSLs e identificar as tendências predominantes no campo.

Os resultados desse SMS revelaram uma notável ênfase no desenvolvimento de novas técnicas e métodos relacionados a DSLs. Contudo, também apontaram para uma relativa falta de atenção à integração de DSLs com outros processos da engenharia de *software*, bem como a necessidade de pesquisas mais rigorosas de avaliação. Esse mapeamento chamou a atenção para fases específicas do desenvolvimento de DSLs que demandavam uma investigação mais aprofundada, incluindo análise de domínio, validação e manutenção. Além disso, destacou as instituições e publicações que contribuíram significativamente para a pesquisa em DSLs. Esses resultados oferecem um contexto valioso para compreender o cenário das DSLs.

Em outro SMS conduzido por (NASCIMENTO et al., 2012), os pesquisadores exploraram o extenso escopo da pesquisa sobre DSLs. Identificando um número significativo de estudos primários relacionados a DSLs, eles empregaram um abrangente esquema de classificação para categorizá-los. Apesar de não se concentrar especificamente em testes funcionais de DSLs, esse SMS proporciona uma visão holística dos aplicativos de DSL em diversos domínios.

O estudo aborda as ferramentas e ambientes de desenvolvimento de linguagem utilizados para criar DSLs, destacando as variadas técnicas e métodos empregados nos trabalhos relacionados a DSLs. Embora não tenha focado nas especificidades das DSLs em testes de *software*, contribui significativamente para uma compreensão mais ampla das aplicações de DSLs e das tendências de pesquisa.

Esses estudos de SMS, embora não especificamente direcionados a DSLs para testes funcionais, oferecem *insights* valiosos sobre o panorama de pesquisa das DSLs. Essas análises abarcam não apenas o desenvolvimento das próprias DSLs, mas também suas aplicações em diversos domínios e as metodologias empregadas em estudos relacionados a DSLs. O entendimento dessas tendências mais amplas pode enriquecer investigações futuras em domínios específicos de DSL, incluindo os testes funcionais de *software*.

3.6 Trabalhos relacionados à Teasy Mobile Language

Com base na análise realizada durante o SMS, apresentado no Capítulo 3, observa-se uma escassez de estudos centrados exclusivamente em dispositivos móveis. Dos quatro estudos mapeados, apenas o estudo S2 aborda uma linguagem voltada especificamente para aplicativos móveis, enquanto os demais utilizam DSLs que possuem a capacidade de descrever diversos domínios. Essa abordagem ampla pode resultar em uma possível perda de representatividade, uma vez que as ações executadas em aplicações *web* podem não ser equivalentes às realizadas em aplicações móveis.

A seguir, são apresentadas duas linguagens relacionadas identificadas por meio do SMS:

- **TBL (*Test-Based Language*):** Projetada para o domínio de aplicativos móveis, a linguagem TBL possui cerca de 50 instruções executadas por um intérprete. Essas instruções são categorizadas em duas principais: instruções gerais, que englobam fluxo de controle, operações aritméticas e relacionais, atribuições e acesso a campos; e instruções de teste, voltadas para a execução de testes, incluindo a invocação de drivers de teste, envio e recebimento de eventos. O enfoque específico em aplicativos móveis destaca a relevância da linguagem TBL para o contexto de testes funcionais nesse domínio (HARGASSNER et al., 2008). Vale salientar que a o estudo é datado de 2008 dessa forma as aplicações móveis atuais não são mais compatíveis com o a linguagem desenvolvida.

- **Gherkin:** Com um escopo mais amplo, aplicável a todos os domínios, a linguagem Gherkin destaca-se por sua capacidade de descrever o comportamento do *software* por meio de linguagem natural, parametrização de dados e integração com diversas estruturas de automação de testes. Sua facilidade de extensão a torna versátil. Embora seja frequentemente utilizada para testar aplicativos da *Web* e móveis, sua adaptação a diferentes domínios, aliada à ênfase na linguagem natural e na parametrização de dados, evidencia sua flexibilidade no contexto de testes funcionais (OLBERG; STREY, 2022).

Com base na análise realizada durante o SMS e nas informações apresentadas acima, foi identificada uma lacuna significativa no desenvolvimento de uma linguagem dedicada exclusivamente aos testes funcionais de *software* em aplicativos móveis. Essa lacuna representa uma oportunidade valiosa para a pesquisa, visando criar uma linguagem que seja verdadeiramente representativa desse domínio específico.

A necessidade de uma linguagem especializada surge da constatação de que as ações e palavras-chave relevantes para testadores de aplicativos móveis podem diferir substancialmente daquelas em ambientes *web* ou outros domínios. Portanto, desenvolver uma linguagem focalizada nesse contexto específico não apenas preencherá uma lacuna identificada no panorama atual, mas também proporcionará uma ferramenta mais eficaz e adequada para profissionais que atuam nesse domínio específico de aplicativos móveis. Essa iniciativa pode contribuir significativamente para a melhoria da eficiência e eficácia dos testes funcionais nesse cenário em constante evolução. Tornando assim, a Teasy Mobile Language com potencial relevante para o panorama geral de DSLs para testes funcionais de *software*.

3.7 Lições do capítulo

Neste capítulo, apresentamos o protocolo e os resultados de um SMS com foco em DSLs para testes funcionais de *software*. A partir dos resultados deste SMS, diversas contribuições foram identificadas:

i Foi conduzida uma análise abrangente da literatura atual sobre o escopo de DSLs focadas em testes funcionais de *software*. Este estudo abrangeu todo o desenvolvimento desse campo de pesquisa, considerando 107 artigos que mapearam o uso, benefícios, tendências e limitações das DSLs atuais.

ii Para profissionais do meio, o SMS contribui ao fornecer um mapeamento abrangente de todas as DSLs existentes para testes funcionais de *software*, destacando seus benefícios, limitações, tendências e funcionalidades. Com base nessas informações, é possível orientar empresas que pretendem adotar ou migrar parte de seu processo de testes funcionais, oferecendo suporte por meio das DSLs.

iii Este SMS também oferece suporte aos pesquisadores ao proporcionar uma compreensão aprofundada do corpo de conhecimento atual sobre DSLs voltadas para testes funcionais de *software*, abordando funcionalidades, limitações, tendências e benefícios associados à adoção dessas tecnologias. Além disso, são apresentados os desafios em aberto identificados, bem como oportunidades de pesquisa para incentivar novos estudos nessa área.

Este SMS desempenha um papel crucial neste trabalho por dois motivos: i) oferece informações essenciais para a definição e aprimoramento da Teasy Mobile Language; ii) serve como base para a identificação de trabalhos relacionados, abordados no Capítulo 3.6.

4 TEASY MOBILE LANGUAGE

Este capítulo apresenta a Teasy Mobile Language e sua integração com o Teasy Framework, detalhado em (LIMA, 2021). O conteúdo deste capítulo inclui os requisitos fundamentais para o desenvolvimento da linguagem, sua sintaxe e exemplos práticos de utilização.

A Teasy Mobile Language foi concebida como uma linguagem projecional que permite a programação orientada ao comportamento da aplicação sob teste. Essa abordagem proporciona a capacidade de gerar diversos artefatos, sendo que, na versão atual, destacamos a geração automatizada de *scripts* de testes executáveis.

Ao utilizar a Teasy Mobile Language, os desenvolvedores podem expressar o comportamento desejado da aplicação de forma mais intuitiva e direta. Isso resulta na geração automática de *scripts* de testes executáveis, simplificando significativamente o processo de teste. Vale ressaltar que, com a introdução dessa linguagem, o Teasy Framework amplia seu suporte para aplicações, abrangendo não apenas ambientes *web*, mas também dispositivos móveis.

4.1 Requisitos

A concepção da Teasy Mobile Language visa preencher uma lacuna identificada no contexto do SMS, que evidenciou a escassez de DSLs específicas para testes funcionais de *software* em dispositivos móveis. A proposta da linguagem é oferecer palavras-chave distintas, destinadas a auxiliar os testadores na otimização de seu desempenho ao automatizar testes nesse ambiente específico. Outro objetivo essencial é reduzir o tempo e esforço despendidos na criação de *scripts* de teste executáveis.

Ao alcançar esses objetivos, a Teasy Mobile Language se destaca como uma ferramenta versátil e adaptável, capaz de atender a uma variedade de cenários de desenvolvimento. Sua integração ao Teasy Framework amplia ainda mais suas capacidades, permitindo que o mesmo testador desenvolva testes tanto para dispositivos móveis quanto para aplicações *web*. Com o intuito de embasar essas premissas, são apresentados detalhadamente os Requisitos (Rs) da linguagem a seguir:

R01. Fornecer a capacidade de reutilização de componentes dentro da linguagem, com o propósito de minimizar o esforço demandado na elaboração de testes.

R02. Simplificar a manutenção dos *scripts* de teste automatizados, tanto aqueles gerados automaticamente quanto os elaborados utilizando a Teasy Mobile Language.

R03. Oferecer uma sintaxe intuitiva, alinhada aos padrões de projeto estabelecidos para testes, em consonância com a versão da linguagem destinada a testes em aplicações

web (LIMA, 2021). Isso possibilita a utilização da linguagem por usuários com diversos níveis de conhecimento, permitindo que o mesmo testador automatizar aplicações *web* e móveis com sintaxes e fluxos de desenvolvimento semelhantes.

R04. Produzir *scripts* de teste para uma estrutura escalável e de fácil manutenção similar a proposta em (LIMA, 2021), visando garantir a adaptabilidade e a eficiência no gerenciamento do código ao longo do ciclo de vida dos testes.

R05. Possibilitar a geração de relatórios contendo informações precisas e abrangentes, visando facilitar uma análise detalhada dos testes executados.

R06. Possibilitar a geração de artefatos de teste, como *scripts* automatizados, para aplicativos móveis em qualquer domínio de aplicação, assegurando a capacidade de automação de testes para diversos contextos.

4.2 Visão geral da nova versão do Teasy Framework

Com a introdução da Teasy Mobile Language, o Teasy Framework expandiu suas capacidades, permitindo que os testadores conduzam testes funcionais tanto em aplicações *web* quanto móveis. Esta seção destaca as melhorias incorporadas ao *framework* apresentado no estudo (LIMA, 2021). A partir da figura 3 é possível visualizar como as soluções se comunicam. A seguir, é apresentado os recursos atualizados do *framework*.

- **Teasy Web Language:** A Teasy Web Language[1] oferece uma abordagem eficiente para especificar as páginas *web* presentes em uma aplicação, bem como as ações associadas a cada página. Isso possibilita a criação de testes, representando os fluxos de execução dentro da aplicação. Para uma compreensão mais aprofundada sobre a Teasy Web Language e seu uso, detalhes adicionais podem ser encontrados em (LIMA, 2021).
- **Teasy Mobile Language:** A Teasy Mobile Language[2] foi projetada com o propósito de permitir que o testador consiga utilizar ambas as linguagens presentes no *framework*. Neste contexto, ela se destina a especificar as páginas em aplicações móveis, assim como as ações associadas a cada página. Essa especificação possibilita a criação de testes, representando os diversos fluxos de execução dentro da aplicação móvel. Vale destacar que as palavras-chave e códigos gerados pela Teasy Mobile Language são totalmente otimizados para dispositivos móveis, proporcionando uma abordagem especializada para testes nesse contexto. Detalhes mais abrangentes sobre a Teasy Mobile Language serão apresentados na Seção 4.3 e na Seção 4.5.
- **Teasy Generator:** O Teasy Generator[3] é uma aplicação *web* que desempenha um

[1] https://github.com/yuryalencar/teasy-language-web
[2] https://github.com/yuryalencar/teasy-language-mobile
[3] https://github.com/yuryalencar/teasy-generator

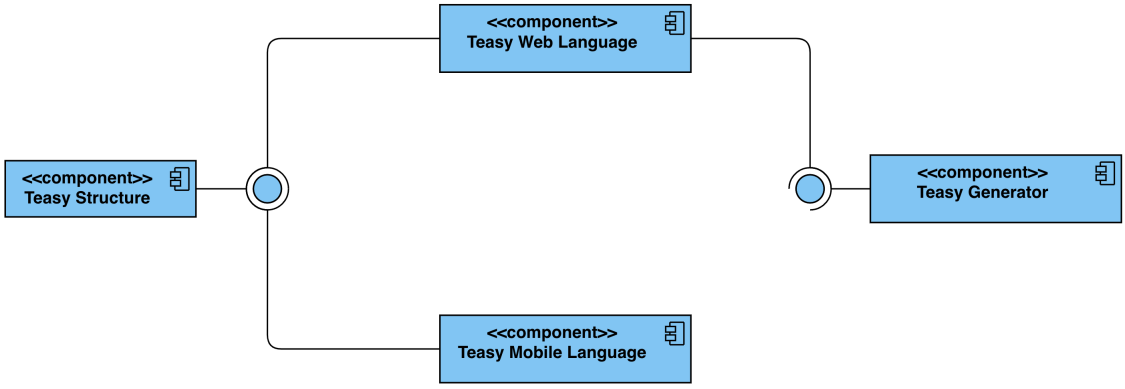


Figura 3 – Visão geral do Teasy Framework atualizada

papel crucial no ecossistema do Teasy Framework. Utilizando como entrada um arquivo *JavaScript Object Notation* (JSON) exportado pela Teasy Web Language, o Teasy Generator permite a definição de caminhos específicos dentro da aplicação. A partir desses caminhos, a aplicação emprega um algoritmo de busca em profundidade para gerar sequências de teste, visando aprimorar a cobertura dentro da aplicação, detalhes adicionais podem ser encontrados em (LIMA, 2021). É importante salientar que, embora a versão atual da Teasy Mobile Language ainda não ofereça suporte direto ao JSON para integração com o Teasy Generator, essa funcionalidade está sendo considerada para futuras atualizações. Essa integração representa um trabalho futuro que visa aprimorar ainda mais a eficiência e a abrangência dos testes definidos através da Teasy Mobile Language.

- **Teasy Structure:** A Teasy Structure[4] representa uma estrutura de diretórios eficiente para organizar projetos de automação de testes. Descrita em detalhes no estudo (LIMA, 2021), essa estrutura de pastas foi projetada para criar projetos escaláveis e de fácil manutenção. Utilizando a Teasy Web Language, é responsabilidade do testador organizar manualmente os arquivos dentro das pastas adequadas da Teasy Structure para garantir o funcionamento adequado do código. No entanto, uma melhoria significativa foi introduzida na Teasy Mobile Language com base no *feedback* dos usuários participantes do quasi-experimento, conforme discutido no estudo (LIMA, 2021). Agora, na Teasy Mobile Language, essa tarefa de organização manual não é mais necessária, proporcionando maior praticidade e eficiência ao usuário. Dessa forma, a Teasy Mobile Language oferece suporte à mesma estrutura escalável de diretórios apresentada na versão *web*, proporcionando uma experiência consistente e simplificada no gerenciamento de projetos de automação de teste.

[4] https://github.com/yuryalencar/teasy-structure

4.3 Teasy Mobile Language

A Teasy Mobile Language destaca-se como uma linguagem de domínio específico do tipo projecional, oferecendo uma abordagem versátil ao permitir a utilização tanto de elementos gráficos quanto textuais. Essa flexibilidade proporciona uma experiência rica ao usuário, permitindo a escolha entre representações visuais ou textuais conforme preferência ou necessidade.

Uma característica distintiva da Teasy Mobile Language é sua capacidade de separar a lógica e a interação com o usuário, graças à natureza projecional. Essa separação possibilita atualizações de usabilidade de forma independente da lógica subjacente, inclusive permitindo a substituição de textos por símbolos sem impactar na definição da linguagem em si. A flexibilidade proporcionada pela linguagem projecional permite uma adaptação mais dinâmica às necessidades evolutivas do projeto, otimizando a experiência do usuário.

Embora a Teasy Mobile Language permita a mescla de símbolos com textos, é importante ressaltar que ela projeta predominantemente textos. Essa escolha deliberada foi fundamentada no estudo de (TÖRSEL, 2013a), o qual destaca que, no domínio de testes, linguagens textuais tendem a ser mais produtivas quando comparadas às gráficas. Assim, a Teasy Mobile Language alinha-se a essa abordagem com o intuito de otimizar a eficiência e produtividade dos testadores no contexto de automação de testes para dispositivos móveis. Além de possibilitar uma usabilidade validada de forma positiva pelo estudo (LIMA, 2021).

Dessa forma, foi implementado um editor para a Teasy Language que adota uma abordagem exclusivamente textual. Essa escolha estratégica visa simplificar a criação e compreensão dos *scripts* de teste, promovendo uma experiência de desenvolvimento mais coesa e intuitiva para os usuários.

Com o intuito de implementar a Teasy Mobile Language, optou-se por utilizar o MPS como LW. Essa escolha foi respaldada por uma análise comparativa entre diversas tecnologias de desenvolvimento de DSLs, na qual o MPS demonstrou superioridade em termos de menor esforço e maior praticidade de uso, conforme destacado em (LIMA et al., 2019).

A adoção do MPS como ambiente de desenvolvimento traz benefícios adicionais, como a disponibilidade de atalhos e funcionalidades de preenchimento automático para diversos elementos da linguagem. Essas características aprimoram a produtividade durante a programação dos testes, proporcionando um fluxo de trabalho mais eficiente e intuitivo para os desenvolvedores que utilizam a Teasy Mobile Language.

4.3.1 Sintaxe

Assim como na versão *web*, a Teasy Mobile Language adota uma abordagem de subdivisão sintática com base em tipos de arquivos. Essa estratégia visa aprimorar a compreensão da linguagem, tornando-a mais acessível e intuitiva para os usuários. A organização desses arquivos segue o padrão de projeto conhecido como *Page Object Models* (RAGHAVENDRA, 2021), uma prática amplamente recomendada pelos próprios desenvolvedores do Selenium (DEV, 2023) para o desenvolvimento de testes automatizados, provendo o atingimento dos requisitos **R02** e **R03**. Nesse contexto, a instância dos diferentes tipos de arquivos dentro da Teasy Mobile Language é considerada uma instância do padrão de projeto *Page Objects*. Essa abordagem contribui para a estruturação modular e eficiente dos testes funcionais, promovendo boas práticas de desenvolvimento e facilitando a manutenção do código ao longo do tempo.

4.3.1.1 Tipos de Arquivos

Com o propósito de viabilizar a implementação eficiente de testes de aplicativos móveis utilizando a Teasy Mobile Language, foram delineados seis tipos fundamentais de arquivos. Esses arquivos interagem entre si, garantindo a coesão e o reúso de componentes, conforme estabelecido no requisito **R01**. A implementação correta de todos esses arquivos é essencial para a compilação bem-sucedida e a subsequente geração de *scripts* de teste executáveis.

É crucial observar que não há uma ordem fixa para a criação desses arquivos, conferindo ao usuário a flexibilidade de desenvolver seus elementos de maneira dinâmica, conforme suas necessidades específicas. A seguir, são apresentados os detalhes de cada arquivo, proporcionando uma visão abrangente dos componentes e dados essenciais para a construção dos testes.

4.3.1.1.1 Arquivo do tipo Configuration

O arquivo do tipo Configuration na Teasy Mobile Language deve ser singular em cada projeto; a presença de mais de um pode acarretar na sobreposição de arquivos durante a geração dos *scripts*. Sua função primordial consiste em garantir a configuração essencial para a execução adequada dos *scripts* de teste, tanto no emulador quanto no aplicativo móvel.

A seguir, são apresentadas as informações necessárias para a elaboração deste arquivo no contexto da Teasy Mobile Language. O cuidado ao configurar esse arquivo é crucial para assegurar a integridade e eficácia dos testes gerados.

- ***YOUR APK FILENAME***: Este campo requer a especificação do nome do arquivo que abriga o aplicativo a ser testado.

- ***YOUR APPIUM SERVER***: Este campo requer a URL local do servidor Appium (FOUNDATION, 2023), o qual é encarregado de realizar todas as interações com o aplicativo em teste no momento da execução.
- ***MAX TIME TO WAIT ELEMENTS (SECONDS)***: Este campo requer um valor numérico em segundos que representa o tempo máximo de espera para que um elemento esteja disponível para interação no aplicativo móvel sob teste.
- ***YOUR PLATFORM NAME***: Este campo requer a especificação da plataforma na qual o teste será executado, podendo ser Android ou iOS.
- ***YOUR DEVICE NAME***: Este campo requer o nome do dispositivo no qual os testes automatizados serão executados, podendo se referir a um dispositivo físico conectado ao computador que realizará a execução ou a um emulador.
- ***YOUR APPLICATION PACKAGE***: Este campo requer o nome do pacote da aplicação sob teste e é utilizado para identificar o aplicativo que será aberto durante a execução dos testes.
- ***YOUR APPLICATION ACTIVITY***: Este campo requer o nome da *activity* que diz respeito a página inicial do aplicativo móvel sob teste e é utilizado para identificar em página que será aberto o aplicativo durante a execução dos testes.
- ***YOUR AUTOMATOR NAME***: Este campo requer o nome do executor de testes que será utilizado durante o processo de execução.

4.3.1.1.2 Arquivo do tipo PageRegisterConfig

O arquivo PageRegisterConfig também deve ser exclusivo para cada projeto, desempenhando um papel crucial para o correto funcionamento dos *scripts* de teste executáveis. Este arquivo deve conter os nomes de todas as páginas criadas (consulte a Seção 4.3.1.1.5). Essa informação é essencial para garantir o funcionamento adequado das importações dentro da Teasy Structure após compilação dos arquivos pela Teasy Mobile Language.

4.3.1.1.3 Arquivo do tipo Hooks

O arquivo Hooks é único para cada projeto e, ao contrário dos demais, não requer uma implementação específica. Sua única função é ser inicializado. A partir deste arquivo, são gerados os hooks dos *scripts* de teste. Os Hooks são responsáveis por abrir/fechar o navegador e capturar *screenshots* da tela ao final dos testes, aprimorando assim o relatório gerado ao final da execução e contribuindo para atender ao requisito **R05**.

4.3.1.1.4 Arquivo do tipo Components

O arquivo Components também deve ser exclusivo por projeto, nele são listados todos os componentes envolvidos nos testes realizados. Este arquivo é fundamental para realizar interações com os componentes, como inserção de valores, cliques, espera por algum estado e até mesmo alteração da estrutura do componente, se necessário. A seguir, são apresentadas as informações que devem ser incluídas para criar um componente.

- ***INSERT NAME COMPONENT***: Este campo requer o nome do componente, que não pode conter espaços e deve ser único, impedindo a duplicação entre os componentes. Esse dado é crucial, pois sempre que você usar este componente, fará referência a esse nome.
- ***INSERT SELECTOR***: Este campo requer um tipo de seletor usado para localizar o componente dentro do aplicativo móvel, oferecendo opções como: identifier, id, accessibility_id, xpath, class, android, ios, nsp, chain, css e name.
- ***INSERT SELECTOR VALUE***: Este campo requer o valor do seletor que é extraído diretamente do XML do aplicativo móvel. A Teasy Mobile Language utilizará esse valor em conjunto com o seletor escolhido para detectar o componente na página.

4.3.1.1.5 Arquivo do tipo Page

O arquivo do tipo Page não deve ser único dentro do projeto. Para cada página da sua aplicação móvel, é recomendado criar um arquivo deste tipo. Neste arquivo, são inseridas as ações que podem ser realizadas por meio da página específica, como realizar uma autenticação, efetuar o cálculo de soma, entre outras. As ações criadas neste arquivo consistem em um conjunto de manipulações realizadas nos componentes definidos no arquivo do tipo Components (Seção 4.3.1.1.4). A seguir, são apresentadas todas as informações necessárias para criar um arquivo do tipo Page dentro da Teasy Mobile Language.

- ***PAGE NAME***: O nome da página é único por arquivo, e ao criar um arquivo, é solicitado que o nome não contenha espaços, pois será utilizado para criar o nome do arquivo posteriormente na geração dos *scripts* executáveis. Além disso, esse nome deve ser registrado no arquivo do tipo PageRegisterConfig (Seção 4.3.1.1.2) após a finalização, garantindo que as ações criadas aqui funcionem corretamente nos *scripts* de testes automatizados gerados.
- ***ACTION***: Dentro de um arquivo do tipo Page, podem existir uma ou várias ações, e de preferência, elas devem ser isoladas, ou seja, sem dependência entre si, para

garantir uma melhor reusabilidade das ações criadas. Para criar uma ação dentro do arquivo, é necessário pressionar a tecla *ENTER* após a linha que solicita o nome da página. A seguir, são apresentadas as informações necessárias para a criação de uma ação:

- ***ACTION NAME***: O nome da ação é, por definição, livre, permitindo a inclusão de espaços. Esse valor será utilizado nos arquivos do tipo Flows (Seção 4.3.1.1.6) para referenciar as ações que devem ser executadas na criação de um fluxo de testes.
- ***INTERACTION***: Dentro de uma ação, é possível realizar diversas interações com os componentes definidos. Nesta versão da Teasy Mobile Language, são oferecidos 34 tipos de interações, 19 a mais do que na versão *web*, destacando a necessidade de uma linguagem especializada para esse domínio. Entre essas interações, incluem-se ações como clicar em um elemento, inserir texto, aguardar por um determinado estado do componente, realizar *scroll* no aplicativo, entre outras. Em cada interação, são especificados o nome do componente e informações adicionais de acordo com a natureza da interação. Duas exceções notáveis são a capacidade de executar *scripts Javascript* e *Adb Shell* (este último exclusivo para aplicativos Android). Tais interações não exigem um componente específico, apenas o código a ser executado. É importante ressaltar que não há limites definidos para a quantidade de ações ou tipos de interações dentro de uma ação, ficando a critério do testador determinar a quantidade necessária. Esta quantidade de interações foram mapeadas de acordo com a ferramenta Appium (FOUNDATION, 2023), dessa forma é possível atingir o requisito **R06**, pois a ferramenta tem como intuito viabilizar interações com todos os domínios de aplicação.

4.3.1.1.6 Arquivo do tipo Flows

O arquivo do tipo Flows também não precisa ser único dentro do projeto; para cada suíte de testes, é recomendado criar um arquivo desse tipo. Este arquivo é destinado à criação de sequências de teste a partir da página inicial do aplicativo móvel. As ações previamente criadas nos arquivos do tipo Page (Seção 4.3.1.1.5) são empregadas para construir fluxos de teste, proporcionando maior reutilização e eliminando a necessidade de duplicação de código (**R01**). A seguir, são apresentadas as informações necessárias para criar um arquivo do tipo Flows:

- ***FILENAME***: Representa a suíte de testes. Esse nome é utilizado tanto para a geração dos *scripts* de testes executáveis quanto como o nome do arquivo. Portanto, é recomendado evitar o uso de espaços.

- ***FLOW NAME***: Esse campo é livre, permitindo o uso de espaços e uma sintaxe flexível. Ele representa o nome do cenário ou fluxo de teste, sendo recomendável que seja único e descritivo para cada arquivo. Dentro de um arquivo do tipo Flows, podem existir um ou mais fluxos. Para criar um novo fluxo, basta pressionar a tecla *ENTER* após preencher o campo *FILENAME*.

- ***EXECUTE***: Dentro de cada fluxo, é possível ter uma ou mais execuções de ações, as quais são informadas sequencialmente para definir um fluxo de teste a ser executado. É crucial notar que cada ação utilizada neste arquivo deve estar presente em algum arquivo do tipo Page. Ao utilizar o atalho *CTRL+SPACE*, o editor incorporado no MPS fornecerá todas as opções possíveis. No entanto, para que as opções funcionem corretamente após a geração, o arquivo do tipo Page (Seção 4.3.1.1.5) que contém essa ação deve estar devidamente registrado no arquivo do tipo PageRegisterConfig (Seção 4.3.1.1.2).

4.4 Teasy Structure

A Teasy Structure é uma composição de diretórios e arquivos criada para estabelecer uma estrutura de testes escalável e de fácil manutenção, conforme validado no estudo (LIMA, 2021). Ao contrário da versão *web* da linguagem, na primeira versão da Teasy Mobile Language, não é mais necessário inserir manualmente os *scripts* de teste executáveis em seus respectivos diretórios. O uso da Teasy Structure é essencial para a execução adequada dos arquivos, pois eles são reutilizados entre si, permitindo que todas as importações funcionem conforme o esperado.

Devido aos *scripts* de teste gerados pela Teasy Mobile Language utilizarem o *Robot Framework*, seu código, em conjunto com a Teasy Structure, tende a ser mais modularizado, facilitando a manutenção direta no código final, se necessário, atendendo ao requisito **R04**. A seguir, são apresentados os diretórios presentes na estrutura:

- **resources:** Este diretório, denominado “resources”, é o único diretório que deve ser criado manualmente para a execução dos testes desenvolvidos com a Teasy Mobile Language. Ele deve ser criado diretamente na raiz do projeto e deve conter o arquivo *.apk* do aplicativo que será testado. Como esse arquivo não é inserido por meio do MPS, o testador deve criar essa pasta manualmente e incluir o arquivo *.apk*. Vale ressaltar que esse diretório foi introduzido exclusivamente para a Teasy Mobile Language, pois requer o arquivo *.apk* para a execução correta dos *scripts* automatizados.

- **config:** Neste diretório, encontram-se todos os arquivos relacionados à configuração para a execução dos testes automatizados, sendo eles: *config.robot* e o *page_register.config.robot*.

- **components:** Neste diretório encontram-se todos os componentes da aplicação sob teste. Estes componentes estão presentes no arquivo *components.robot*.
- **tests:** Neste diretório encontram-se todos os arquivos de teste, mapeados através dos arquivos do tipo Flows.
- **pages:** Neste diretório encontram-se todas as páginas da aplicação conforme o padrão *Page Objects*, mapeados através dos arquivos do tipo Page.
  - **commons:** Neste diretório, estão localizados os arquivos utilizados em todas as páginas para garantir a execução correta dos *scripts* automatizados. Até o momento, apenas o arquivo *hooks.pages.commons.robot* atende a esse requisito.

4.5 Exemplo de uso da Teasy Mobile Language

Com o intuito de fornecer um exemplo prático, optou-se por utilizar uma aplicação bancária da indústria, escolhendo assim um cenário crítico que pertence ao domínio de aplicações móveis. Dessa forma, busca-se representar um ambiente realista. No entanto, vale ressaltar que este exemplo abrange apenas uma ilustração simplificada do uso da Teasy Mobile Language. A seguir, serão detalhados os passos para criar testes em aplicações móveis utilizando essa linguagem. Neste exemplo específico, abordaremos um teste negativo de autenticação.

A Teasy Mobile Language, assim como sua outra versão *web*, gerencia todo o fluxo de testes, desde as configurações até a execução. Sendo assim, é aconselhável iniciar o projeto pelas configurações para garantir o funcionamento esperado. Conforme apresentado na Seção 4.3, o arquivo representado na Figura 4 é do tipo Configuration e pode ser interpretado da seguinte forma: os testes implementados serão executados em um arquivo *.apk* com o nome "app-hom-release". Implicitamente, será aguardado um período de até trinta segundos até que um elemento fique visível. As ações serão realizadas por meio do Appium, presente no servidor *http://localhost:4723/wd/hub*. Os testes serão executados em um dispositivo "Android" com o nome "emulator-5554", utilizando o executor "UIAutomator2". Por fim, o aplicativo deve ser aberto no pacote "br.com.app.package" na tela "br.com.app.activity". Vale ressaltar que os dados referentes ao pacote e às telas de início foram atualizados para preservar a confidencialidade do aplicativo.

Após a configuração dos testes, o próximo passo é identificar os elementos envolvidos e criar um arquivo do tipo Components. Este arquivo, único na aplicação, contém todos os elementos de todas as páginas. A Figura 5 apresenta os componentes envolvidos em uma autenticação com falha. Para encontrar o seletor, é necessário inspecionar os elementos no XML do aplicativo por meio de uma ferramenta externa.

Com os componentes definidos, é necessário criar um arquivo do tipo Page que representa uma página da aplicação sob teste. Neste caso, há apenas uma página de-

Figura 4 – Arquivo do tipo Configuration

```
CONFIGURATIONS

  YOUR APK FILENAME: app-hom-release
  YOUR APPIUM SERVER: http://localhost:4723/wd/hub
  MAX TIME TO WAIT ELEMENT (SECONDS): 30
  YOUR PLATFORM NAME: Android
  YOUR DEVICE NAME: emulator-5554
  YOUR APPLICATON PACKAGE: br.com.app.package
  YOUR APPLICATION ACTIVITY: br.com.app.activity
  YOUR AUTOMATOR NAME UiAutomator2 (Android)
```

Source: Autor

Figura 5 – Arquivo do tipo Components

```
SYSTEM COMPONENTS

  INSERT NAME COMPONENT: CpfField
  INSERT SELECTOR: xpath
  INSERT SELECTOR VALUE: //*[contains(@class, 'android.widget.EditText')[1] and (@text(),'000.000')]

  INSERT NAME COMPONENT: LoginButton
  INSERT SELECTOR: xpath
  INSERT SELECTOR VALUE: //android.widget.Button[@index='15']

  INSERT NAME COMPONENT: PasswordField
  INSERT SELECTOR: xpath
  INSERT SELECTOR VALUE: //android.widget.EditText[@index='13']

  INSERT NAME COMPONENT: DataNotFound
  INSERT SELECTOR: accessibility_id
  INSERT SELECTOR VALUE: Dados não encontrados

  INSERT NAME COMPONENT: EnterButton
  INSERT SELECTOR: accessibility_id
  INSERT SELECTOR VALUE: ENTRAR
```

Source: Autor

nominada “LoginPage”. Essa página contém ações relacionadas à página, como clicar no botão de autenticação, clicar no botão *ENTER*, verificar se é possível visualizar uma mensagem de erro, entre outras ações. A Figura 6 apresenta um exemplo completo da página mencionada. Devido à sintaxe livre, o testador pode descrever as ações da maneira que considerar mais fácil para a leitura; nesse caso, também foi introduzido um ator que realiza a ação para tornar a leitura mais intuitiva.

Sempre que um arquivo do tipo Page for criado, é necessário adicioná-lo ao arquivo do tipo PageRegisterConfig para garantir que todas as importações funcionem como esperado durante a execução dos testes. Se uma Page for criada e não for adicionada a

Figura 6 – Arquivo do tipo Page

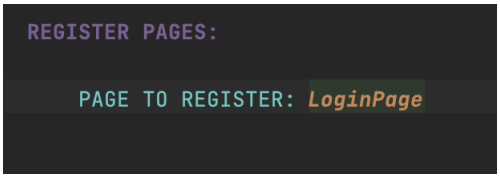
```
PAGE NAME: LoginPage

  ACTION: Alice fills form with invalid data $<
    WAIT UNTIL ELEMENT IS VISIBLE: CpfField TIMEOUT: 2

    CLICK ELEMENT: CpfField

    INPUT TEXT
    COMPONENT: CpfField
    TEXT: 000000000-00

    CLICK ELEMENT: PasswordField

    INPUT TEXT
    COMPONENT: PasswordField
    TEXT: password

  >$
  ACTION: Alice clicks to Login $<
    WAIT UNTIL ELEMENT IS VISIBLE: LoginButton TIMEOUT: 2

    CLICK ELEMENT: LoginButton

  >$
  ACTION: Alice sees data not found message $<
    SLEEP TIME: 1

    PAGE SHOULD CONTAIN ELEMENT: DataNotFound

  >$
  ACTION: Alice clicks to enter $<
    WAIT UNTIL ELEMENT IS VISIBLE: EnterButton TIMEOUT: 2

    CLICK ELEMENT: EnterButton

  >$
```

Source: Autor

este arquivo, não será possível utilizá-la para criar uma sequência de teste. A Figura 7 apresenta um exemplo deste arquivo contendo a única página mapeada na aplicação bancária escolhida.

Outro arquivo crucial de configuração é o arquivo do tipo Hooks. Ele deve ser criado no projeto para inicializar os aplicativos sob teste sempre antes de cada execução. Mesmo que não seja necessário escrever conteúdo específico, esse arquivo é de suma importância para a instalação, abertura e fechamento do aplicativo no dispositivo configurado. Além disso, ele contribui para a geração de um relatório final, incluindo capturas

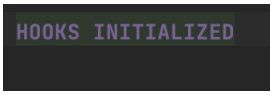
Figura 7 – Arquivo do tipo PageRegisterConfig



Source: Autor

de tela de cada execução, este arquivo pode ser visualizado através da Figura 8.

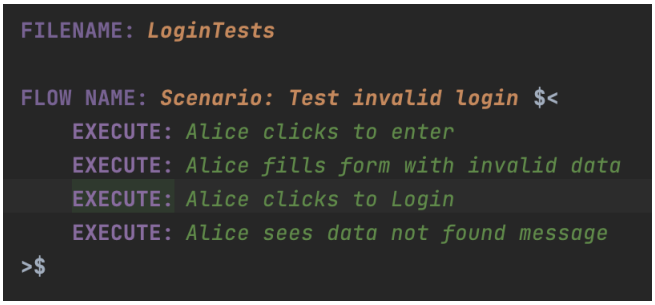
Figura 8 – Arquivo do tipo Hooks



Source: Autor

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 possuem vínculos com as ações cadastradas nos arquivos do tipo Page, assim, para criar um fluxo, o arquivo requisita um nome para a(s) sequência(s), além das ações que devem ser realizadas. É importante salientar que o fluxo deve ser criado a partir da primeira página da aplicação. A Figura 9 apresenta um exemplo de teste referente a uma autenticação realizada com falha.

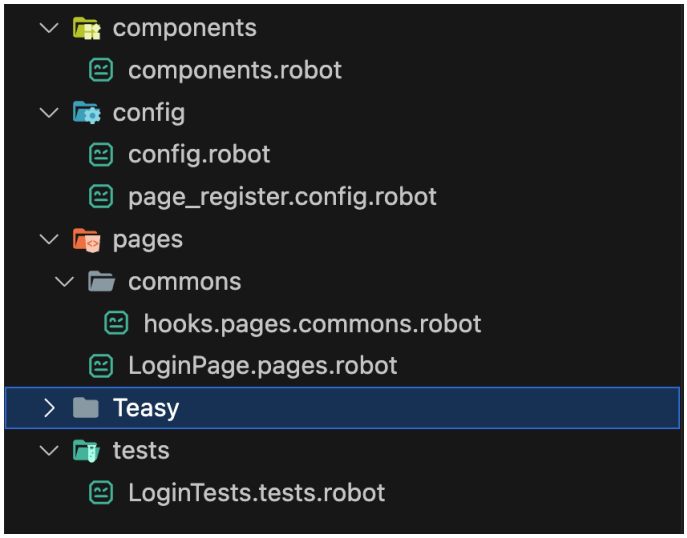
Figura 9 – Arquivo do tipo Flows



Source: Autor

Após especificar suas sequências e páginas utilizando a Teasy Mobile Language, a próxima etapa consiste na geração dos *scripts* executáveis. Essa ação é realizada por meio do MPS. Após a execução desse processo, será gerada uma estrutura de pastas e arquivos, conforme ilustrado na Figura 10.

Figura 10 – Teasy Structure gerada



Source: Autor

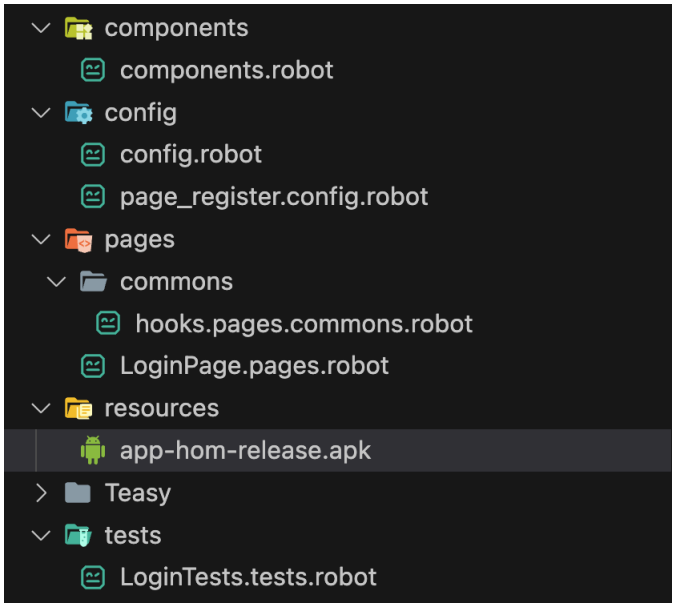
O acesso a esses arquivos é realizado por meio do diretório do projeto Teasy em seu computador. Atualmente, os arquivos ainda são gerados diretamente na Teasy Structure. Portanto, basta acessar o diretório Teasy do seu computador e navegar pelas pastas correspondentes: *languages* > *Teasy* > *sandbox* > *source_gen*. Caso seus arquivos não estejam presentes, é possível realizar um "Rebuild Model 'Teasy.sandbox'" para eliminar possíveis problemas relacionados ao cache do computador. Entretanto o diretório "resources" como mencionado anteriormente não é gerado diretamente na Teasy Structure, dessa forma é necessário que seja criado manualmente. Depois de ser criado manualmente é possível adicionar o arquivo *.apk* com a aplicação sob teste conforme apresentado na Figura 11.

Com os arquivos nos diretórios correspondentes, você pode executar os *scripts* de teste gerados usando o seguinte comando na raiz do projeto: "robot -d ./logs tests". Esse comando executará todos os fluxos especificados, instalando o aplicativo presente na pasta "resources" dentro do dispositivo informado, além de o abrir e fechando ao final da execução para cada cenário de teste, isolando assim cada sequência de testes. Ao término da execução, será gerado um diretório denominado "logs", que contém um relatório em HTML navegável com os resultados dos testes, incluindo capturas de tela.

4.6 Lições do Capítulo

Concluindo este capítulo, fica evidente que o Teasy Framework se tornou ainda mais robusto com a incorporação da Teasy Mobile Language. Essa adição não apenas amplia a abrangência do *framework*, mas também reforça seu compromisso com reúso máximo e flexibilidade para os testadores. A introdução da Teasy Mobile Language visa

Figura 11 – Resultado após adicionar o diretório resources à Teasy Structure



Source: Autor

significativamente reduzir o esforço necessário na criação de *scripts* de testes executáveis, proporcionando uma experiência mais eficiente.

É importante destacar que as soluções oferecidas pelo Teasy Framework, como a Teasy Web Language e a Teasy Mobile Language, podem ser utilizadas de maneira independente. Essa característica possibilita a aplicação dessas soluções em diversos contextos e domínios, eliminando uma limitação percebida em outras abordagens encontradas na literatura. A especialização da Teasy Mobile Language para testes em dispositivos móveis representa um avanço significativo, abordando uma lacuna observada em outras soluções disponíveis.

Os pontos positivos destacados neste capítulo resumem-se da seguinte forma:

- **Redução do esforço**: A utilização da Teasy Mobile Language facilita a automação em dispositivos móveis ao simplificar a projeção de informações automaticamente para o usuário com base no arquivo e contexto. Isso resulta em uma redução significativa no número de teclas necessárias para implementar uma sequência específica, tornando o processo mais eficiente e acessível para o testador.

- **Reúso**: A Teasy Mobile Language destaca-se pelo reúso eficiente de componentes e pela integração coesa com a Teasy Structure. Esse benefício é refletido na redução significativa da quantidade de código necessário, proporcionando maior facilidade de entendimento. Além disso, a automação do processo elimina a necessidade de configurações manuais nos arquivos gerados dentro da estrutura, simplificando ainda mais o fluxo de trabalho do testador.

- **Soluções Semelhantes**: A Teasy Mobile Language foi estruturada de forma a permitir que o testador crie cenários de teste utilizando o mesmo processo de desenvolvimento empregado na Teasy Web Language. Essa abordagem viabiliza que o testador possa criar testes automatizados para dispositivos móveis e aplicações web com um conjunto consistente de conhecimentos. Além disso, ao adotar o padrão *Page Objects*, a Teasy Mobile Language proporciona flexibilidade para sua aplicação em diferentes projetos, garantindo uma abordagem unificada.

Detalhes aprofundados sobre os benefícios e eventuais limitações da recém-criada linguagem para testes em aplicações móveis serão minuciosamente explorados na Seção 5. Esta análise mais específica proporcionará *insights* valiosos, consolidando as experiências adquiridas durante a implementação e utilização prática da Teasy Mobile Language por testadores da indústria.

5 ESTUDO DE CASO UTILIZANDO A TEASY MOBILE LANGUAGE

5.1 Introdução

Apesar da automação de testes contribuir significativamente para a eficiência na verificação de aplicações, sua implementação continua a ser uma tarefa manual exigente, sujeita a constante manutenção (TÖRSEL, 2013a). Essa exigência impõe desafios particulares no contexto do desenvolvimento ágil, onde a agilidade é crucial e os testes desempenham um papel vital na asseguração da qualidade e confiabilidade do *software* (TÖRSEL, 2013a). Nesse cenário, a Teasy Mobile Language surge como uma solução promissora, apresentando uma abordagem eficaz para a criação de *scripts* de teste automatizados, voltados para dispositivos móveis.

Dada a crescente quantidade de aplicativos móveis sendo disponibilizados nas lojas da Google Play e Apple Store na indústria, é evidente que esses produtos demandam uma atenção especial em relação à qualidade, especialmente quando se trata de soluções críticas, como aplicativos bancários. Nesse contexto, identificamos, por meio do SMS apresentado na Capítulo 3, uma lacuna de pesquisa relacionada à falta de DSLs especializadas que simplifiquem e otimizem o processo de automação de testes em aplicativos móveis. A Teasy Mobile Language, desenvolvida como uma extensão do bem avaliado Teasy Framework (LIMA, 2021), e surge como uma opção viável para atender a essa demanda.

Assim, com o propósito de avaliar empiricamente a eficácia e aplicabilidade da Teasy Mobile Language, definiu-se este estudo de caso. Com intuito de garantir uma representatividade em aplicações do mundo real, optou-se por uma aplicação bancária de uma empresa genuína. Este aplicativo funciona como um banco para condomínios, buscando simplificar as operações bancárias diárias dos síndicos. Ao selecionar casos de uso que representem as atividades cotidianas dessa empresa, nosso objetivo é analisar como a linguagem contribui para a eficiência, reusabilidade e manutenção dos *scripts* de teste no cenário real de desenvolvimento.

Ao explorar a Teasy Mobile Language no ambiente industrial, tem como foco compreender sua integração ao ciclo de vida de desenvolvimento de *software*. Um dos objetivos foi buscar identificar os benefícios tangíveis que ela proporciona aos testadores em comparação com soluções previamente utilizadas e avaliar sua adaptação aos desafios únicos enfrentados na indústria. Este contexto de estudo de caso visa fornecer *insights* valiosos sobre o papel da Teasy Mobile Language na automação de testes em ambientes industriais. Contribui para a compreensão do impacto dessa linguagem no aprimoramento da qualidade e eficiência dos processos de desenvolvimento de *software* móvel na indústria.

5.2 Design do estudo de caso

5.2.1 Questões de pesquisa

A condução deste estudo visa responder a perguntas fundamentais que permeiam a eficácia e aplicabilidade da Teasy Mobile Language no contexto industrial. As perguntas de pesquisa apresentam as informações que serão analisando, proporcionando *insights* valiosos sobre a contribuição da linguagem nos processos de automação de testes. A seguir são apresentadas as Questões de Pesquisa (QPs) mapeadas para este estudo de caso:

QP01. Como a Teasy Mobile Language impacta na redução do tempo e esforço dedicados ao desenvolvimento de casos e *scripts* de teste em comparação com outras tecnologias de mercado?

QP02. A curva de aprendizado para o uso da Teasy Mobile Language é mais acessível em comparação com outras tecnologias similares?

5.2.2 Seleção dos casos de uso

Nesta subseção, foi detalhado o processo de seleção dos casos de teste e dos sujeitos participantes do estudo de caso, visando proporcionar uma análise representativa e aprofundada do impacto da Teasy Mobile Language.

A seleção das histórias de usuário destinadas à criação dos cenários de teste foi fundamentada na relevância e complexidade das funcionalidades do aplicativo bancário em análise. Optamos por escolher três casos de uso que abrangem diferentes aspectos da aplicação, considerando, além disso, a variação nas complexidades dos cenários para a automação. Essa abordagem visa proporcionar uma avaliação abrangente e representativa da capacidade da Teasy Mobile Language em lidar com diferentes desafios presentes na aplicação real. A seguir são apresentados todos os cenários mapeados:

- **Nome do cenário**: Autenticação do Usuário
  - **História de usuário**: Eu como Síndico gostaria de acessar o meu aplicativo através de um usuário e senha para verificar o status da conta do condomínio.
  - **Complexidade**: Baixa.
  - **Motivo**: É uma funcionalidade tradicional de diversos aplicativos móveis existentes.
- **Nome do cenário**: Transferência de valores (TED)
  - **História de usuário**: Eu como Síndico gostaria de realizar um TED para pagar as contas do meu condomínio.

  - **Complexidade**: Média.
  - **Motivo**: É uma funcionalidade representativa do domínio bancário, que representa um grande problema caso não funcione corretamente.

- **Nome do cenário**: Aceitar condômino
  - **História de usuário**: Eu como Síndico gostaria de poder aceitar um condômino para que o mesmo consiga visualizar a conta bancária do condomínio a fim de possibilitar uma maior transparência.
  - **Complexidade**: Alta.
  - **Motivo**: É uma funcionalidade que depende de espera de carregamento e também fatores externos para funcionarem como uma solicitação realizada através de outro aplicativo.

5.2.3 Seleção dos participantes

Com o objetivo de assegurar representatividade dentro do contexto industrial, a totalidade da equipe de testadores, composta por quatro profissionais, foi selecionada para participar do estudo. A escolha de participantes abrangendo diferentes níveis de experiência tem como intuito garantir uma análise abrangente da adoção da linguagem, avaliando a usabilidade e eficácia da Teasy Mobile Language independentemente do nível de conhecimento dos usuários.

5.2.4 Procedimento de coleta de dados

A coleta de dados para este estudo de caso foi conduzida em duas fases principais: a criação dos casos de teste utilizando a Teasy Mobile Language e a aplicação de um questionário aos testadores. Abaixo, foi descrito detalhadamente cada fase do procedimento de coleta de dados:

- **Fase 1**: Criação dos casos de teste
  - **Atividade**: Os testadores realizaram o uso da Teasy Mobile Language para criar casos de teste automatizados para as três histórias de usuário específicadas na Subseção 5.2.2.
- **Fase 2**: Aplicação do questionário
  - **Atividade**: Após a criação dos casos de teste, os testadores responderam a um questionário estruturado, abordando diferentes aspectos da experiência deles com a Teasy Mobile Language.
  - **Tópicos abordados no questionário**:

- Avaliação da curva de aprendizado.
- Percepção sobre a eficiência no desenvolvimento de *scripts* de teste.
- Comparação com outras tecnologias anteriormente utilizadas.
- Identificação de desafios específicos encontrados.
- Sugestões de melhorias para a linguagem.

A combinação desses métodos de coleta de dados proporcionará uma compreensão abrangente da eficácia e usabilidade da Teasy Mobile Language no contexto da automação de testes em aplicações móveis na indústria.

5.2.5 Procedimento de análise dos dados

Após a coleta de dados, a análise foi conduzida em uma etapa abrangente, com o objetivo de aprofundar a compreensão sobre a eficácia e usabilidade da Teasy Mobile Language pelos testadores participantes. A principal abordagem utilizada foi a análise qualitativa, centrada nas respostas obtidas por meio do formulário enviado a cada testador. A seguir são apresentadas as etapas da análise realizada.

- **Tabulação das Respostas**: As respostas fornecidas pelos testadores foram tabuladas de forma sistemática para facilitar a compilação e compreensão dos dados.
- **Medição da Satisfação**: A análise qualitativa priorizou a mensuração da satisfação dos testadores no uso da Teasy Mobile Language. Com isso, foi possível identificar pontos de destaque e áreas que necessitam de aprimoramento.
- **Identificação de Benefícios e Limitações**: Foram identificados benefícios específicos percebidos pelos testadores durante o uso da linguagem, assim como eventuais limitações que impactaram a experiência.
- **Sugestões de Melhoria**: As sugestões apresentadas pelos testadores foram categorizadas, proporcionando *insights* valiosos sobre possíveis aprimoramentos na Teasy Mobile Language.

A análise qualitativa permitiu uma avaliação holística da linguagem, fornecendo *insights* cruciais para a compreensão da experiência dos testadores no mercado de trabalho. Os resultados destacaram áreas de sucesso, desafios enfrentados e oportunidades de melhoria, contribuindo para uma visão abrangente sobre a aplicabilidade da Teasy Mobile Language na automação de testes em ambientes industriais.

5.2.6 Procedimentos de validade

A validade do estudo de caso é um elemento crucial para assegurar que os resultados e conclusões sejam confiáveis e aplicáveis ao contexto em questão. Diversos procedimentos foram adotados para garantir a validade do estudo:

- **Validade Interna**: A escolha de histórias de usuário relevantes e representativas contribui para a validade interna, garantindo que os cenários de teste abordaram funcionalidades significativas do aplicativo bancário representando um cenário real para os participantes.
- **Validade Externa**: A participação de toda a equipe de testadores, composta por quatro membros, busca refletir a diversidade de perspectivas, contribuindo para a validade externa e generalização dos resultados.
- **Validade de Construção**: A utilização do formulário estruturado e objetivo contribui para a validade de construção, assegurando que as respostas dos testadores estejam alinhadas com os objetivos específicos deste estudo.
- **Validade de Conclusão**: A análise qualitativa abrangente, incluindo a tabulação das respostas, a mensuração da satisfação e a identificação de benefícios e limitações, contribui para a validade de conclusão ao proporcionar uma compreensão profunda dos resultados.
- **Triangulação de Dados**: A triangulação de dados, utilizando respostas do formulário e *feedbacks* direto dos testadores, fortalece a validade dos resultados ao corroborar informações provenientes de diferentes fontes.
- **Coleta de *feedback* Pós-Análise**: Após a análise inicial, os participantes foram convidados a revisar os resultados, promovendo a validação da análise realizada por meio da perspectiva dos próprios testadores.

Esses procedimentos em conjunto buscam assegurar que o estudo de caso seja robusto, confiável e ofereça resultados que possam ser generalizados e aplicados em cenários semelhantes na indústria de desenvolvimento de software.

5.3 Ameaças à validade

Embora o estudo de caso tenha sido planejado meticulosamente, algumas ameaças à validade podem influenciar os resultados e as conclusões. É crucial reconhecer essas ameaças para interpretar os resultados com cautela e considerar suas limitações. A seguir são apresentadas as ameaças à validade mapeadas até o momento:

- **Validade Interna**: A subjetividade associada à complexidade das histórias de usuário selecionadas pode impactar a validade interna do estudo, uma vez que a escolha de casos específicos pode não abranger todos os níveis possíveis de complexidade. A avaliação da complexidade é inerentemente subjetiva e pode variar entre os participantes, o que pode influenciar a generalização dos resultados para além dos

casos específicos escolhidos. Essa consideração é essencial ao interpretar os resultados e reconhecer possíveis limitações na representação abrangente da complexidade das funcionalidades do aplicativo.

- **Validade Externa**: Apesar de contar com um grupo de testadores de uma empresa na indústria, o tamanho da amostra ainda é considerado baixo, o que pode limitar a generalização dos resultados para além da equipe de testadores participantes. Isso impacta a validade externa do estudo, uma vez que diferentes contextos e equipes podem ter experiências diversas e tamanhos distintos, dificultando a extensão dos resultados para um escopo mais amplo na indústria de desenvolvimento de *software* móvel. Essa limitação deve ser levada em consideração ao interpretar e aplicar os resultados em diferentes cenários organizacionais.

- **Validade de Construção**: A interpretação subjetiva das respostas dos testadores pode introduzir viés, afetando a validade de construção. Diferentes interpretações das perguntas podem levar a respostas variadas, mesmo que o formulário tenha sido elaborado de forma simples e objetiva. Para mitigar esse desafio, foram fornecidas instruções claras junto ao formulário, buscando minimizar ambiguidades e assegurar uma compreensão uniforme por parte dos participantes. Por fim, outra ameaça de construção detectada é o que os participantes do estudo conheciam o desenvolvedor da linguagem, entretanto o primeiro contato dos participantes foram através do estudo de caso.

- **Validade de Conclusão**: Mudanças no ambiente de teste ou no contexto da equipe durante a coleta de dados podem impactar a validade de conclusão, pois fatores externos têm o potencial de influenciar as percepções dos testadores. Esse impacto pode ser ainda mais evidente em situações em que problemas de produção ocorram, levando a um certo nível de exaustão durante o uso da Teasy Mobile Language. Essas possíveis influências externas foram registradas e consideradas durante a análise dos resultados para fornecer uma visão mais precisa e contextualizada.

- **Validade Temporal**: Alterações na Teasy Mobile Language ou nas ferramentas relacionadas após a conclusão do estudo podem afetar a validade temporal, podendo tornar os resultados menos aplicáveis a versões futuras.

Reconhecer essas ameaças à validade é essencial para interpretar os resultados de maneira crítica. O estudo busca minimizar essas ameaças por meio de procedimentos apresentados na Subseção 5.2.6, mas é fundamental reconhecer que algumas limitações podem persistir.

5.4 Análise dos resultados

Este capítulo tem como objetivo fornecer uma análise abrangente dos resultados obtidos a partir dos formulários preenchidos pelos participantes. A análise é estruturada em diversas subseções, abordando aspectos-chave, incluindo: 1) Perfil dos Participantes (Subseção 5.4.1): Nesta Subseção, serão apresentadas informações sobre o perfil dos participantes, destacando a diversidade de experiências e níveis de conhecimento dentro da equipe de testadores. 2) Avaliação da Linguagem (Subseção 5.4.2): A análise dos resultados gerados a partir da aplicação do formulário apresentando os benefícios, limitações e sugestões de melhoria identificados pelos participantes será abordada. 3) Conclusão (Subseção 5.4.3): A análise geral será concluída com uma subseção dedicada à síntese dos principais resultados. Aqui, será possível destacar tendências, resumir os *insights* mais significativos e oferecer uma visão consolidada dos resultados. Essa estrutura foi elaborada para garantir uma abordagem abrangente e organizada da análise dos resultados, permitindo uma compreensão clara e detalhada do impacto da Teasy Mobile Language na experiência dos testadores. Por fim, todos os dados respondidos no formulário analisado nessa seção são públicos para o acesso[1], os pessoais dos participantes como e-mail foram atualizados com o fim de garantir a anonimidade. Entretanto, por fins de confidencialidade os códigos automatizados pela equipe de participantes não pode ser público, decorrente a ser uma aplicação bancária considerada crítica.

5.4.1 Perfil dos participantes

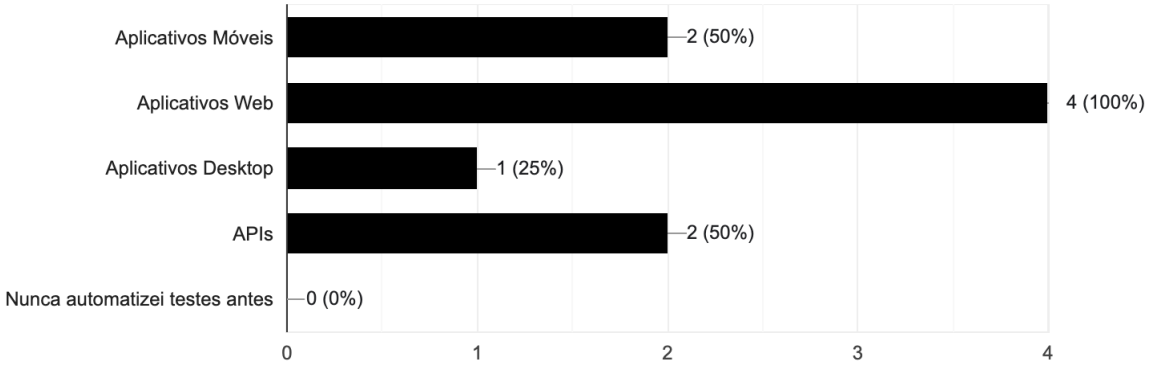
O estudo de caso envolveu todos os membros disponíveis na equipe da empresa, abrangendo testadores de diferentes níveis de conhecimento com o intuito de avaliar a Teasy Mobile Language. Dos quatro participantes, dois possuem graduação completa, um possui graduação incompleta e outro possui pós-graduação, todos na área de tecnologia. Quanto à experiência profissional, a diversidade foi evidente, com dois participantes acumulando mais de nove anos de experiência na indústria, um com entre 1 e 3 anos, e um com até 1 ano de experiência em qualidade de *software*. Todos os participantes trabalham atualmente com testes em dispositivos móveis, refletindo a relevância direta da Teasy Mobile Language para o contexto de suas atividades.

Na experiência dos participantes em atividades de automação de testes, todos os participantes já possuíam experiência anteriormente, conforme evidenciado pela Figura 12, que destaca os tipos de *software* anteriormente automatizados pelo grupo de participantes antes da introdução da Teasy Mobile Language. Entretanto, é importante salientar que todos os participantes possuem experiência em testes de aplicativos móveis realizados manualmente. Essa diversidade de experiências prévias permite uma comparação rica

[1] https://docs.google.com/spreadsheets/d/1EKoFD-3ZejiMkF7QZCpCEAUgK0qrZQ1i7vD-pzWVHxs/edit?usp=sharing

entre as ferramentas utilizadas anteriormente e a Teasy Mobile Language.

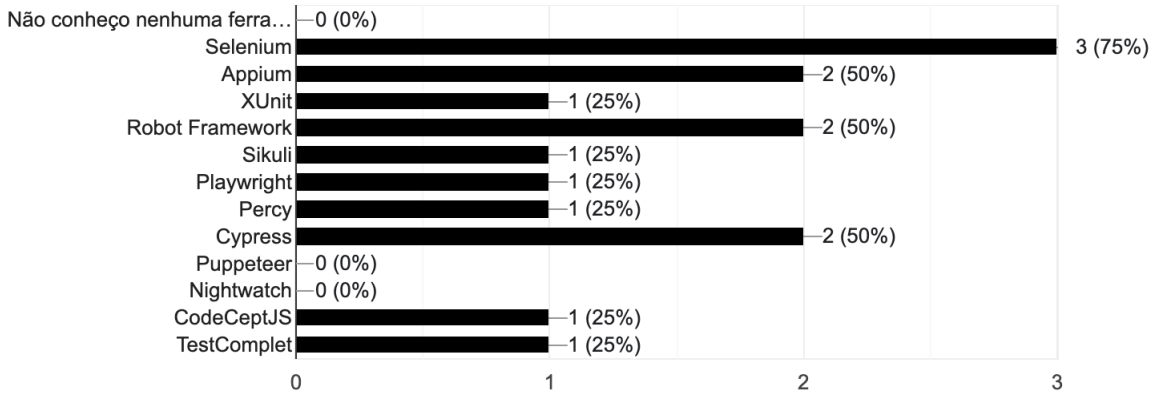
Figura 12 – Experiência dos participantes em atividades de automação de testes



Source: Autor

Além disso, a Figura 13 apresenta um gráfico que destaca as ferramentas conhecidas pelos testadores, evidenciando o conhecimento prévio do grupo em relação às tecnologias de automação de testes. Essa análise indireta permite uma compreensão mais profunda das percepções dos participantes sobre a Teasy Mobile Language em comparação com as tecnologias previamente conhecidas.

Figura 13 – Ferramentas conhecidas pelos participantes



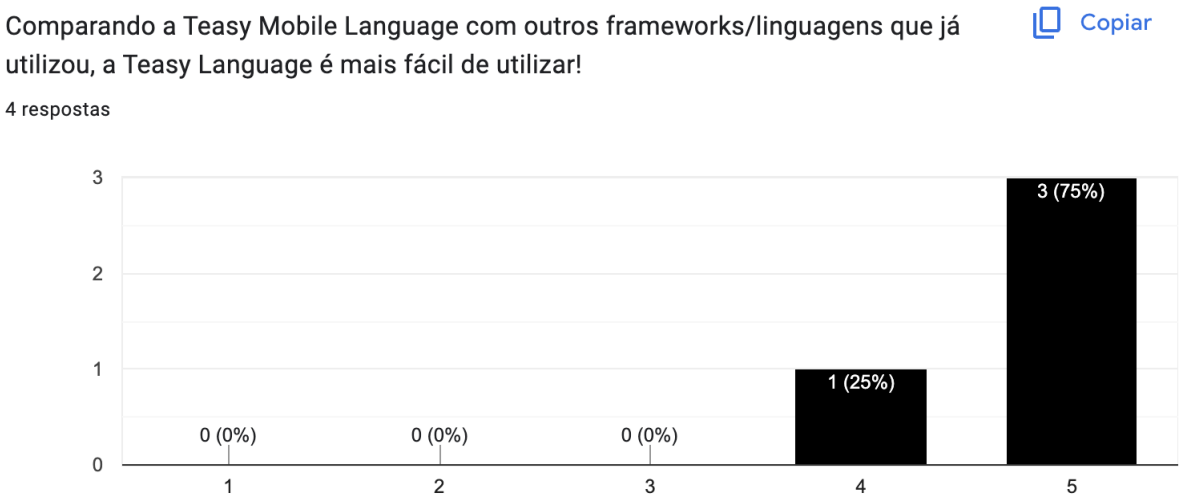
Source: Autor

5.4.2 Avaliação da linguagem

Com o objetivo de abordar a QP02, a equipe foi questionada através do formulário sobre a facilidade de uso da Teasy Mobile Language em comparação com os *frameworks* ou linguagens que já conheciam. Dos quatro participantes, três expressaram total concordância, atribuindo uma pontuação de 5 na escala *Likert*, enquanto o quarto

participante atribuiu uma pontuação de 4, indicando uma concordância acima da média, conforme ilustrado na Figura 14.

Figura 14 – Comparando a Teasy Mobile Language com outros *frameworks* e ferramentas no quesito facilidade de uso



Source: Autor

Quando questionados sobre os cenários específicos em que a Teasy Mobile Language se mostrou mais fácil de utilizar, os participantes destacaram a criação de cenários, a estruturação dos testes, a configuração inicial do projeto, o uso de palavras-chave diretamente no MPS sem a necessidade de consultar documentação externa, além da facilidade para criar componentes e descrever ações.

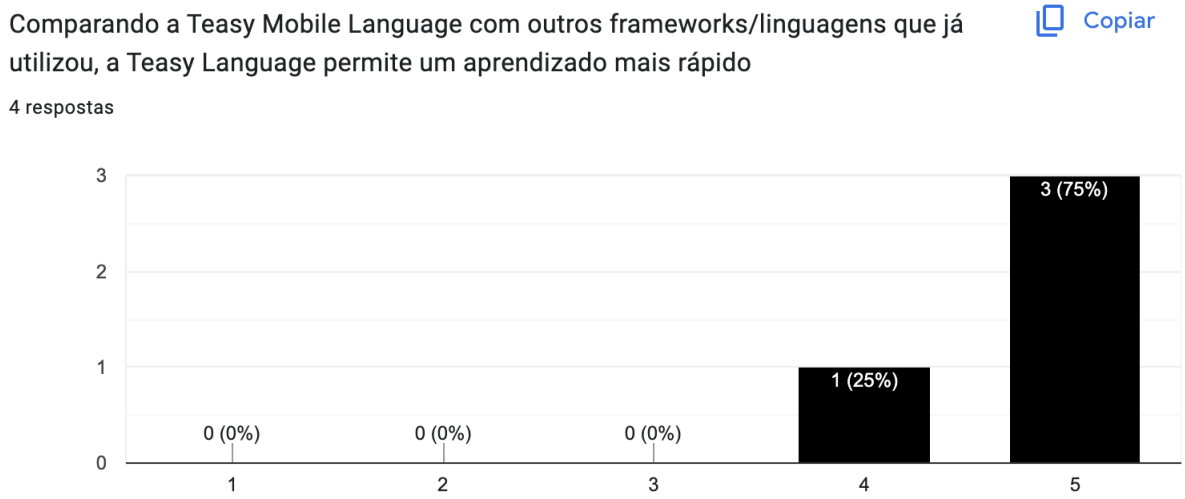
No entanto, alguns pontos foram identificados como desafiadores pelos testadores. A parametrização de dados, a depuração de código e a reutilização de algumas palavras-chave foram destacados como áreas em que a linguagem apresentou maior complexidade, especialmente no contexto do aprendizado. Esses *insights* fornecem uma compreensão abrangente das percepções dos participantes sobre os aspectos mais acessíveis e desafiadores referentes ao aprendizado da Teasy Mobile Language.

Ainda com intuito de de abordar a QP02, foi incorporada outra pergunta ao formulário, abordando a facilidade de aprendizado para a utilização da linguagem. É relevante destacar que os participantes receberam apenas a linguagem, um arquivo README contendo boas práticas e instruções de instalação da DSL, além de uma reunião para apresentar os cenários a serem desenvolvidos. Não foram realizadas múltiplas reuniões de treinamento ou sessões para esclarecimento de dúvidas durante o período de avaliação.

Mesmo sem um treinamento formal, três dos participantes atribuíram uma pontuação máxima de 5, concordando totalmente que a Teasy Mobile Language foi mais fácil de aprender em comparação com outras soluções. O quarto participante também votou

positivamente, com uma pontuação de 4, indicando uma facilidade acima da média no aprendizado, conforme evidenciado na Figura 15.

Figura 15 – Comparando a Teasy Mobile Language com outros *frameworks* e ferramentas no quesito aprendizagem



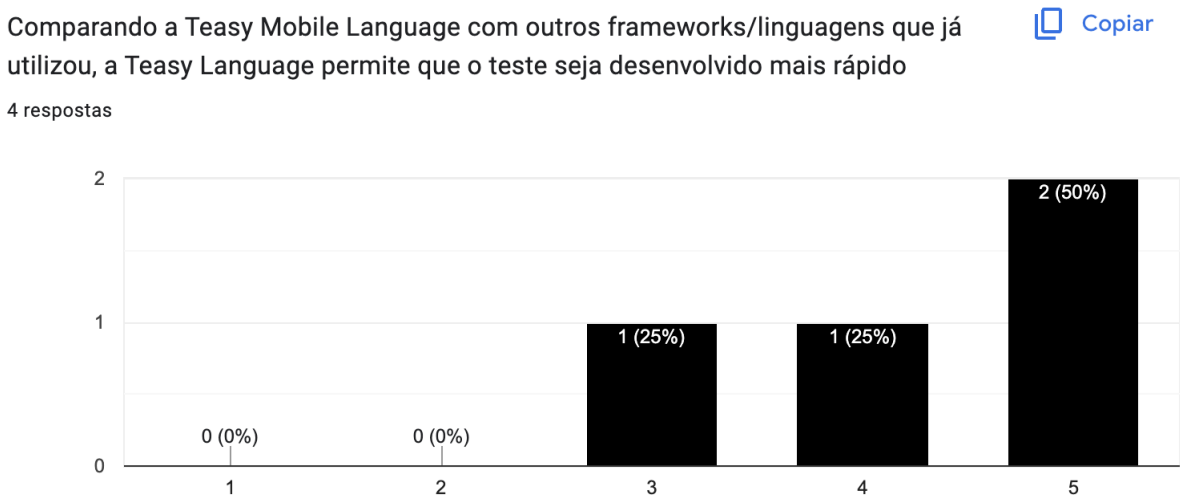
Source: Autor

Os participantes destacaram vários pontos positivos em relação à facilidade de aprendizado, incluindo a estrutura do projeto, a criação de cenários de teste, a clareza das funcionalidades, a completude e simplicidade do README. Esses aspectos ressaltam a acessibilidade e eficácia da Teasy Mobile Language no que diz respeito à facilidade de aprendizado, mesmo sem um treinamento formal.

Com o intuito de abordar a QP01, foram incluídas duas perguntas adicionais no formulário. A primeira delas explorou a velocidade necessária para criar testes em comparação com as ferramentas/*frameworks* conhecidos. Dois participantes atribuíram uma pontuação máxima de 5 na escala Likert, concordando totalmente que a Teasy Mobile Language reduz significativamente o tempo de desenvolvimento de cenários. Um participante concedeu uma pontuação de 4, indicando uma concordância acima da média, enquanto outro atribuiu uma pontuação de 3, sugerindo que, para essa pessoa, o tempo geral foi semelhante ao uso de outras tecnologias, conforme ilustrado na Figura 16.

Os participantes destacaram vários pontos em que a Teasy Mobile Language se mostrou mais rápida. Isso inclui a criação de componentes, a definição das ações dos testes, a configuração inicial do projeto e a sintaxe simples, eliminando a necessidade de grandes trechos de código. No entanto, alguns aspectos foram identificados como menos eficientes. A falta de parametrização exigiu a replicação de cenários com diferentes valores, e a execução dos testes foi apontada como uma atividade que demandou mais tempo. Esses *insights* oferecem uma visão detalhada das percepções dos participantes sobre a eficácia da Teasy Mobile Language em termos de velocidade de desenvolvimento

Figura 16 – Comparando a Teasy Mobile Language com outros *frameworks* e ferramentas no quesito tempo de criação de cenários



Source: Autor

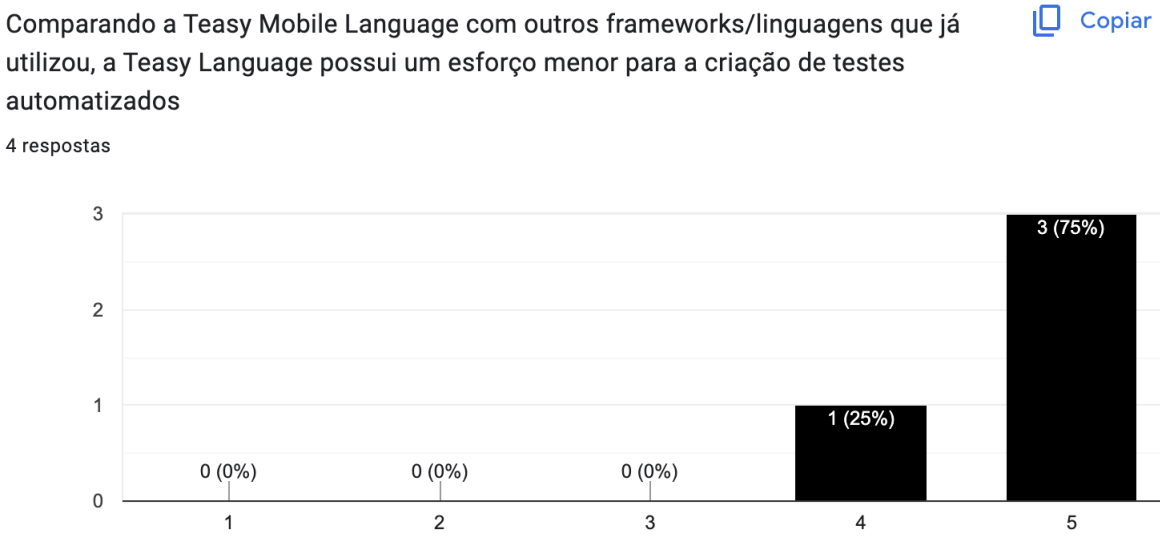
de testes.

Ainda visando responder à QP01, foi solicitado aos participantes, por meio do formulário, que opinassem sobre se a linguagem exigia menor esforço para a criação de testes automatizados, desconsiderando o tempo de desenvolvimento. Três dos quatro participantes atribuíram uma pontuação máxima de 5 na escala Likert, concordando totalmente que a Teasy Mobile Language reduziu significativamente o esforço necessário para criar testes automatizados. O quarto participante concedeu uma nota de 4, ainda acima da média, conforme ilustrado na Figura 17.

Os participantes destacaram vários pontos positivos em favor da Teasy Mobile Language, indicando um esforço reduzido para a criação de testes automatizados. Esses aspectos incluem a facilidade na redação de cenários, a simplicidade da sintaxe, a rapidez na criação de testes positivos e a eficácia das validações por meio de *xpaths*, que contribuíram para a redução do esforço. Contudo, o único ponto mencionado como demandante de mais esforço foi a necessidade de definir diversas ações com valores diferentes para contornar a falta de parametrização e possibilitar o uso de variáveis globais.

Concluindo a avaliação da Teasy Mobile Language, dos quatro participantes, três afirmaram que certamente utilizariam a linguagem, enquanto apenas um indicou que talvez a utilizaria. Os pontos positivos levantados incluíram a facilidade de uso, a agilização de processos e na criação de componentes, bem como uma sintaxe simples que proporciona um excelente primeiro contato para testadores que ainda não automatizam cenários de teste. Vale ressaltar que nenhum ponto detrator foi identificado durante a avaliação. No entanto, foi observado que, dependendo do tamanho do projeto e do nível de conhecimento em programação da equipe de testadores, outras soluções poderiam ser

Figura 17 – Comparando a Teasy Mobile Language com outros *frameworks* e ferramentas no quesito esforço



Source: Autor

mais flexíveis.

Como sugestão de melhoria, o grupo de participantes destacou a necessidade de aprimorar a Teasy Mobile Language para que ela possa gerar testes automatizados em diversas tecnologias para automação diferentes, visto que atualmente a linguagem produz testes apenas para o Robot Framework. Esse *feedback* valioso fornece uma direção clara para futuras atualizações e aprimoramentos da Teasy Mobile Language.

5.4.3 Conclusão

Neste estudo de caso, buscamos avaliar empiricamente a eficácia e aplicabilidade da Teasy Mobile Language na indústria, utilizando uma aplicação bancária real como contexto. Com a participação de toda uma equipe de testadores, obtivemos *insights* valiosos sobre como a linguagem se integra ao ciclo de vida de desenvolvimento de *software*, os benefícios percebidos pelos testadores em comparação com outras soluções e sua adaptação aos desafios enfrentados na indústria.

No que diz respeito ao perfil dos participantes, a diversidade de experiência e formação proporcionou uma representatividade abrangente. Os resultados sugerem que a Teasy Mobile Language foi bem recebida por testadores com diferentes níveis de conhecimento, desde os mais experientes até aqueles que estão começando na área.

Ao analisar a facilidade de aprendizado, a maioria dos participantes concordou que a Teasy Mobile Language é mais fácil de aprender em comparação com outras tecnologias. Destacaram-se pontos como a estrutura do projeto, criação de cenários de teste e funcionalidades claras como facilitadores desse processo.

Quanto à eficiência, os participantes indicaram que a Teasy Mobile Language reduz significativamente o tempo necessário para criar cenários de teste, destacando a criação de componentes, definição de ações e a sintaxe simples como pontos positivos. Contudo, alguns participantes apontaram que a parametrização de dados e a reutilização de algumas palavras-chave podem ser áreas de melhoria.

Na questão do esforço necessário, a maioria dos participantes concordou que a Teasy Mobile Language demanda menos esforço para criar testes automatizados, enfatizando a facilidade na escrita de cenários, a sintaxe simples e a rapidez na criação de testes positivos. No entanto, alguns participantes observaram que a falta de parametrização e a necessidade de definir várias ações com valores diferentes podem aumentar o esforço em determinados contextos.

E possível analisar que a satisfação dos participantes com a Teasy Mobile Language foi positiva, com três dos quatro indicando que certamente utilizariam a linguagem em seus projetos. Os benefícios percebidos incluíram facilidade de uso, agilização de processos e uma sintaxe acessível, enquanto as sugestões de melhoria foram direcionadas principalmente para a expansão da geração de testes automatizados para outras tecnologias além do Robot Framework.

Esses resultados fornecem uma visão abrangente da Teasy Mobile Language na prática, evidenciando seus pontos fortes e identificando áreas que podem ser aprimoradas. A linguagem demonstrou ser uma opção viável e eficaz para a automação de testes em dispositivos móveis na indústria, promovendo uma experiência positiva para os testadores. As sugestões apresentadas pelos participantes oferecem orientações valiosas para futuras atualizações e refinamentos, contribuindo para o contínuo desenvolvimento e aprimoramento da Teasy Mobile Language.

5.5 Lições do Capítulo

Considerando os *insights* extraídos do formulário, este capítulo permite concluir que a Teasy Mobile Language, mesmo em sua versão inicial, obteve uma aceitação positiva entre os profissionais da indústria. A linguagem demonstrou possuir diversas funcionalidades que facilitam a navegação no código, proporcionando aos testadores uma compreensão mais clara e eficiente, além de capacidades para identificar e resolver potenciais problemas.

É relevante destacar que o estudo de caso também revelou áreas de aprimoramento para a próxima versão da linguagem. Esses desafios identificados representam oportunidades valiosas para elevar ainda mais a qualidade e eficácia da Teasy Mobile Language.

Os benefícios evidenciados no decorrer do estudo, como a completude e simplicidade da linguagem, bem como a agilidade durante a implementação de casos de teste, foram aspectos positivos. Contudo, é crucial reconhecer que alguns pontos negativos, especialmente em relação à parametrização, também foram identificados como *feedbacks*

relevantes e são considerados para futuras melhorias.

Dessa forma, a análise do estudo de caso não apenas destaca as conquistas iniciais da Teasy Mobile Language, mas também fornece uma base sólida para direcionar esforços no sentido de superar desafios específicos. Este capítulo reforça a importância contínua do desenvolvimento da linguagem, alinhando-se às demandas e expectativas dos profissionais da indústria, visando aprimorar ainda mais sua utilidade e eficácia em ambientes de automação de testes em dispositivos móveis.

6 CONSIDERAÇÕES FINAIS

Neste estudo foi apresentada a Teasy Mobile Language como uma extensão do Teasy Framework, proporcionando uma abordagem eficaz para a criação de *scripts* de teste automatizados voltados para dispositivos móveis. O desenvolvimento dessa linguagem foi impulsionado pela necessidade de simplificar e otimizar o processo de automação de testes em aplicativos móveis, reconhecendo a crescente importância desses dispositivos na indústria e a demanda por soluções ágeis e eficientes. A identificação dessa lacuna específica foi possível por meio da análise dos resultados do SMS realizado, revelando a carência de soluções direcionadas a esse domínio específico. Essa lacuna apresentou a falta de uma quantidade representativa de DSLs focadas em dispositivos móveis, sendo que apenas as linguagens Gherkin e a TBL apareceram em estudos com o foco em testes em dispositivos móveis. Como a TBL é uma linguagem datada de 2008, os aplicativos atuais não são mais compatíveis com a versão da linguagem apresentada no estudo, mostrando assim a importância da proposta da Teasy Mobile Language para o cenário de testes automatizados em dispositivos móveis.

A análise minuciosa da Teasy Mobile Language revelou diversos pontos positivos que respaldam sua eficácia e aceitação no ambiente industrial, conforme demonstrado pelo estudo de caso. A linguagem oferece uma sintaxe clara e intuitiva, facilitando tanto a criação quanto a manutenção de casos de teste. A integração direta com o Teasy Framework proporciona uma experiência unificada, especialmente para os testadores familiarizados com a versão *web* da linguagem. Isso amplia as possibilidades, permitindo que os testadores automatizem testes tanto para o ambiente *web* quanto para dispositivos móveis.

O estudo de caso conduzido em um ambiente industrial validou a eficácia da Teasy Mobile Language. Os participantes, abrangendo diversos níveis de conhecimento e experiência, enfatizaram a agilidade na criação de cenários de teste e a facilidade de aprendizado como aspectos positivos da linguagem. No entanto, identificaram áreas que podem ser aprimoradas, como a demanda por maior flexibilidade na parametrização e a expansão para a geração de testes automatizados em diversas tecnologias de automação diferentes.

As lições aprendidas com a implementação e avaliação da Teasy Mobile Language proporcionaram *insights* valiosos para o desenvolvimento futuro da linguagem. A coleta de *feedbacks* dos testadores da indústria permitiu identificar desafios específicos e oportunidades de melhoria, orientando as próximas versões da linguagem. Além disso, uma melhoria futura mapeada é a integração completa com o Teasy Framework. Atualmente, a Teasy Mobile Language ainda não gera um arquivo utilizado pelo Teasy Generator para a criação de cenários de teste com base em buscas em um grafo. Portanto, além das sugestões fornecidas pelos participantes, essa integração também está planejada como trabalho futuro. Outro trabalho futuro mapeado é a execução de um experimento contro-

lado com mais participantes, com o intuito de comparar de forma direta a Teasy Mobile Language com outras soluções de mercado, validando assim a sua eficácia para o avaliar o seu uso na indústria.

É crucial destacar que a Teasy Mobile Language não visa substituir soluções existentes, mas sim complementar o ecossistema de ferramentas disponíveis para a automação de testes móveis. Sua integração contínua com o Teasy Framework reforça a abordagem de reúso de componentes e simplifica a transição para testadores que já estão familiarizados com a versão *web* da linguagem.

Por fim, com base nos resultados deste estudo, a Teasy Mobile Language se destaca como uma alternativa promissora para a automação de testes em dispositivos móveis, proporcionando agilidade, simplicidade e uma integração efetiva com uma estrutura de automação previamente validada (LIMA, 2021). Seu contínuo desenvolvimento, impulsionado por contribuições da comunidade, dada a natureza de código aberto, e pelas demandas da indústria, buscam aprimorar ainda mais a experiência dos testadores em automação de testes, especialmente em um cenário cada vez mais centrado em dispositivos móveis.

BIBLIOGRAFIA

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. Cited at page 30.

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. Cited at page 31.

ARTHO, C. et al. Domain-specific languages with scala. In: SPRINGER. **International Conference on Formal Engineering Methods**. [S.l.], 2015. p. 1–16. Cited at page 31.

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. Cited at page 31.

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. Cited at page 30.

BETTINI, L. **Implementing domain-specific languages with Xtext and Xtend**. [S.l.]: Packt Publishing Ltd, 2016. Cited at page 19.

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. Cited at page 30.

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. Cited at page 30.

CAMILLI, M. et al. Online model-based testing under uncertainty. In: . [S.l.: s.n.], 2018. p. 36–46. Cited at page 30.

CAMILLI, M. et al. Hyppotest: Hypothesis testing toolkit for uncertain service-based web applications. In: _____. [S.l.: s.n.], 2019. p. 495–503. ISBN 978-3-030-34967-7. Cited at page 30.

CARVALHO, R. E. V. de. A comparative study of gui testing approaches. 2016. Cited at page 30.

CASADEI, R. et al. Scafi: A scala dsl and toolkit for aggregate programming. **SoftwareX**, v. 20, p. 101248, 12 2022. Cited at page 31.

CAVALCANTE, M.; SALES, J. The behavior driven development applied to the software quality test:. In: . [S.l.: s.n.], 2019. p. 1–4. Cited at page 30.

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. Cited at page 30.

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. Cited at page 30.

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. Cited at page 30.

COTRONEO, D. et al. Profipy: Programmable software fault injection as-a-service. In: . [S.l.: s.n.], 2020. p. 364–372. Cited at page 31.

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. Cited at page 31.

DELAMARO, M.; JINO, M.; MALDONADO, J. **Introdução ao teste de software**. [S.l.]: Elsevier Brasil, 2013. Cited 2 times at pages 14 and 20.

DEV, S. **Page object models**. 2023. Disponível em: <https://www.selenium.dev/pt-br/documentation/test_practices/encouraged/page_object_models/>. Cited at page 64.

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. Cited at page 30.

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. Cited at page 30.

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. Cited at page 30.

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. Cited at page 30.

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. Cited at page 30.

FOUNDATION, O. **Appium Documentation**. 2023. Disponível em: <https://appium.io/docs/en/2.1/>. Cited 2 times at pages 65 and 67.

FOWLER, M. Language workbench. 2005. Disponível em: <https://martinfowler.com/bliki/LanguageWorkbench.html>. Cited at page 19.

FOWLER, M. Internal dsl style. 2006. Disponível em: <https://martinfowler.com/bliki/InternalDslStyle.html>. Cited at page 18.

FOWLER, M. **Domain-specific languages**. [S.l.]: Pearson Education, 2010. Cited 3 times at pages 14, 18, and 19.

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. Cited at page 30.

GOSLING, J.; HOLMES, D. C.; ARNOLD, K. **The Java programming language**. [S.l.]: Addison-Wesley, 2005. Cited at page 18.

GRIEBE, T.; GRUHN, V. A model-based approach to test automation for context-aware mobile applications. **Proceedings of the ACM Symposium on Applied Computing**, 03 2014. Cited at page 30.

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. Cited at page 30.

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. Cited 2 times at pages 30 and 58.

HÄRLIN, M. **Testing and Gherkin in agile projects**. 2016. Cited at page 30.

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. Cited at page 30.

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. Cited at page 30.

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. Cited at page 30.

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. Cited at page 31.

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. Cited at page 30.

HESENIUS, M.; GRIEBE, T.; GRUHN, V. Towards a behavior-oriented specification and testing language for multimodal applications. In: . [S.l.: s.n.], 2014. Cited at page 30.

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. Cited at page 31.

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. Cited at page 30.

HU, G.; ZHU, L.; YANG, J. Appflow: using machine learning to synthesize robust, reusable ui tests. In: . [S.l.: s.n.], 2018. p. 269–282. Cited at page 30.

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. Cited at page 30.

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. Cited at page 30.

IUNG, A. et al. Systematic mapping study on domain-specific language development tools. **Empirical Softw. Engg.**, Kluwer Academic Publishers, USA, v. 25, n. 5, p. 4205–4249, sep 2020. ISSN 1382-3256. Disponível em: <https://doi.org/10.1007/s10664-020-09872-1>. Cited at page 19.

JETBRAINS. How does mps work?. 2019. Disponível em: <https://www.jetbrains.com/mps/concepts/index.html\#projection-editor>. Cited 3 times at pages 16, 19, and 20.

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. Cited at page 30.

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. Cited at page 31.

KESSERWAN, N. et al. From use case maps to executable test procedures: a scenario-based approach. **Software Systems Modeling**, v. 18, 04 2019. Cited at page 30.

KHORRAM, F. et al. Advanced testing and debugging support for reactive executable dsls. **Software and Systems Modeling**, v. 22, 09 2022. Cited at page 31.

KHORRAM, F. et al. Automatic test amplification for executable models. In: . [S.l.: s.n.], 2022. p. 109–120. Cited at page 31.

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. Cited at page 30.

KIRINUKI, H. et al. Nlp-assisted web element identification toward script-free testing. In: . [S.l.: s.n.], 2021. p. 639–643. Cited at page 31.

KIRINUKI, H. et al. Web element identification by combining nlp and heuristic search for web testing. In: . [S.l.: s.n.], 2022. p. 1055–1065. Cited at page 31.

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. Cited at page 31.

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. Cited at page 30.

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. Cited at page 31.

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. Cited at page 30.

KOSAR, T.; BOHRA, S.; MERNIK, M. Domain-specific languages: A systematic mapping study. **Information and Software Technology**, Elsevier, v. 71, p. 77–91, 2016. Cited at page 56.

LIMA, Y. A. Teasy framework: uma solução para testes automatizados em aplicações web. Universidade Federal do Pampa, 2021. Cited 11 times at pages 14, 15, 23, 29, 60, 61, 62, 63, 68, 76, and 91.

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>. Cited at page 63.

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. Cited at page 31.

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. Cited at page 30.

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. Cited at page 30.

MA, S.-P. et al. Graph-based and scenario-driven microservice analysis, retrieval, and testing. **Future Generation Computer Systems**, v. 100, p. 724–735, 11 2019. Cited at page 30.

MACIEL, D. A. M. Model based testing-from requirements to tests. 2019. Cited at page 30.

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. Cited at page 30.

MAKEDONSKI, P. et al. Test descriptions with etsi tdl. **Software Quality Journal**, Springer, p. 1–33, 2019. Cited at page 30.

MEIXNER, K. et al. Efficient test case generation from product and process model properties and preconditions. In: . [S.l.: s.n.], 2020. p. 859–866. Cited at page 31.

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. Cited at page 18.

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. Cited at page 30.

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. Cited at page 30.

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. Cited at page 31.

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. Cited at page 31.

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. Cited at page 30.

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. Cited at page 30.

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. Cited at page 30.

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. Cited at page 14.

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. Cited at page 31.

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. Cited at page 30.

MYERS, G. J.; SANDLER, C.; BADGETT, T. **The art of software testing**. [S.l.]: John Wiley & Sons, 2011. Cited at page 20.

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. Cited at page 30.

NASCIMENTO, L. M. do et al. A systematic mapping study on domain-specific languages. In: **The Seventh International Conference on Software Engineering Advances (ICSEA 2012)**. [S.l.: s.n.], 2012. p. 179–187. Cited at page 57.

NEGRINO, T.; SMITH, D. **JavaScript para world wide web**. [S.l.]: Pearson Education, 2001. Cited at page 18.

NIKIFOROVA, A. et al. Data quality model-based testing of information systems: the use-case of e-scooters. In: . [S.l.: s.n.], 2020. Cited at page 31.

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. Cited at page 30.

OLBERG, P.; STREY, L. Approach to generating functional test cases from bpmn process diagrams. In: . [S.l.: s.n.], 2022. p. 185–189. Cited 2 times at pages 31 and 58.

OLIVEIRA, D. F. et al. Development of a self-diagnostic system integrated into a cyber-physical system. **Computers**, v. 11, p. 131, 08 2022. Cited at page 31.

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. Cited at page 31.

OTHMAN, R.; ZEIN, S. Test case auto-generation for web applications: A model-based approach. In: . [S.l.: s.n.], 2022. Cited at page 31.

PAIVA, A.; SILVA, A.; SILVA, V. A test specification language for information systems based on data entities, use cases and state machines. **Communications in Computer and Information Science**, v. 991, 01 2019. Cited at page 30.

PAIVA, A. C.; VILELA, L. Multidimensional test coverage analysis: Paradigm-cov tool. **Cluster Computing**, Springer, v. 20, n. 1, p. 633–649, 2017. Cited at page 30.

PETERSEN, K. et al. Systematic mapping studies in software engineering. In: **Ease**. [S.l.: s.n.], 2008. v. 8, p. 68–77. Cited 2 times at pages 23 and 56.

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. Cited at page 30.

QUYET, T. H. et al. Generating test data for blackbox testing from uml-based web engineering content and presentation models. In: _____. [S.l.: s.n.], 2019. p. 207–219. ISBN 978-3-030-30148-4. Cited at page 30.

RAGHAVENDRA, S. Page objects. In: **Python Testing with Selenium**. [S.l.]: Springer, 2021. p. 143–150. Cited at page 64.

RAMLER, R.; KLAMMER, C. Enhancing acceptance test-driven development with model-based test generation. In: . [S.l.: s.n.], 2019. p. 503–504. Cited at page 30.

RENNOCH, A. et al. Using tdl for standardised test purpose definitions. In: . [S.l.: s.n.], 2020. Cited at page 31.

RIOS, E. et al. **Base de Conhecimento em Teste de Software. 2ª Edição–S. Paulo**. [S.l.]: Martins Fontes, 2007. Cited at page 20.

RODRIGUES, F. C. Pattern based usability testing. 2018. Cited at page 30.

SACRAMENTO, C. Reverse engineering of interaction patterns. 2014. Cited at page 30.

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. Cited at page 30.

SADOWSKI, C.; YI, J. Tiddle: A trace description language for generating concurrent benchmarks to test dynamic analyses. 08 2009. Cited at page 30.

SANTIAGO, D. et al. Towards domain-specific testing languages for software-as-a-service. In: CITESEER. **MDHPCL@ MoDELS**. [S.l.], 2013. p. 43–52. Cited at page 31.

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. Cited at page 30.

SCHNEID, K. et al. Automated regression tests: A no-code approach for bpmn-based process-driven applications. In: . [S.l.: s.n.], 2021. Cited at page 31.

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. Cited at page 30.

SILVA, T.; WINCKLER, M.; TRæTTEBERG, H. Ensuring the consistency between user requirements and graphical user interfaces: A behavior-based automated approach. In: . [S.l.: s.n.], 2019. p. 616–632. ISBN 978-3-030-24288-6. Cited at page 30.

SILVA, T.; WINCKLER, M.; TRæTTEBERG, H. Ensuring the consistency between user requirements and gui prototypes: A behavior-based automated approach. In: . [S.l.: s.n.], 2019. p. 644–665. ISBN 978-3-030-29380-2. Cited at page 30.

SILVA, T.; WINCKLER, M.; TRæTTEBERG, H. Ensuring the consistency between user requirements and task models: A behavior-based automated approach. v. 4, p. 1–32, 06 2020. Cited at page 31.

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. Cited at page 30.

SILVA, V. E. R. da. Model based testing-from requirements to tests. 2017. Cited at page 30.

SIPPL, C. et al. Scenario-based systems engineering: An approach towards automated driving function development. In: . [S.l.: s.n.], 2019. p. 1–8. Cited at page 30.

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. Cited at page 30.

SIRIUS. **Sirius. https://www.eclipse.org/sirius/**. 2019. Disponível em: <https://www.eclipse.org/sirius/>. Cited at page 19.

SNOOK, C. et al. Domain-specific scenarios for refinement-based methods. **Journal of Systems Architecture**, v. 112, p. 101833, 07 2020. Cited at page 31.

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. Cited 2 times at pages 14 and 15.

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. Cited at page 30.

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. Cited at page 30.

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. Cited 3 times at pages 14, 63, and 76.

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. Cited at page 30.

TUGLULAR, T.; ŞENSüLüN, S. Spl-at gherkin: A gherkin extension for feature oriented testing of software product lines. In: . [S.l.: s.n.], 2019. p. 344–349. Cited at page 30.

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. Cited at page 31.

UTTING, M.; LEGEARD, B. **Practical model-based testing: a tools approach**. [S.l.]: Elsevier, 2010. Cited 2 times at pages 14 and 21.

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. Cited at page 31.

VILELA, L. B. Test coverage analysis. 2013. Cited at page 30.

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. Cited at page 31.

WOHLIN, C. et al. **Experimentation in software engineering**. [S.l.]: Springer Science & Business Media, 2012. v. 1. Cited 4 times at pages 24, 29, 55, and 56.

WOLDE, B.; BOLTANA, A. Behavior-driven re-engineering for testing the cloud. In: . [S.l.: s.n.], 2020. p. 75–82. Cited at page 31.

ZANIN, A.; ZORZO, A. F.; NUNES, H. C. Model-based testing in agile projects: An approach based on domain-specific languages. In: **Conferencia Iberoamericana de Software Engineering**. [s.n.], 2020. Disponível em: <https://api.semanticscholar.org/CorpusID:229539961>. Cited at page 31.

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. Cited at page 30.

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. Cited at page 18.