PUCRS

ESCOLA POLITÉCNICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
DOUTORADO EM CIÊNCIA DA COMPUTAÇÃO

ALINE ZANIN

TESTE BASEADO EM MODELOS EM PROJETOS ÁGEIS, UMA ABORDAGEM BASEADA EM LINGUAGEM DE DOMÍNIO ESPECÍFICO

Porto Alegre
2019

PÓS-GRADUAÇÃO - *STRICTO SENSU*

Pontifícia Universidade Católica
do Rio Grande do Sul

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
ESCOLA POLITÉCNICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

TESTE BASEADO EM MODELOS EM PROJETOS ÁGEIS, UMA ABORDAGEM BASEADA EM LINGUAGEM DE DOMÍNIO ESPECÍFICO

ALINE ZANIN

Tese apresentada como requisito parcial à obtenção do grau de Doutor em Ciência da Computação na Pontifícia Universidade Católica do Rio Grande do Sul.

Orientador: Prof. Avelino Francisco Zorzo

Porto Alegre
2019

Ficha Catalográfica

Z31t Zanin, Aline

Teste Baseado em Modelos em Projetos Ágeis, Uma Abordagem Baseada em Linguagem de Domínio Específico / Aline Zanin . – 2019.

139p.

Tese (Doutorado) – Programa de Pós-Graduação em Ciência da Computação, PUCRS.

Orientador: Prof. Dr. Avelino Francisco Zorzo.

1. Testes de Software. 2. Métodos Ágeis. 3. DSL. 4. DSL. 5. MBT. I. Zorzo, Avelino Francisco. II. Título.

Elaborada pelo Sistema de Geração Automática de Ficha Catalográfica da PUCRS
com os dados fornecidos pelo(a) autor(a).
Bibliotecária responsável: Salete Maria Sartori CRB-10/1363

Aline Zanin

Baseado em Modelos em Projetos ágeis, uma Abordagem Baseada em Linguagem de Domínio Específico

Tese apresentada como requisito parcial para obtenção do grau de Doutor em Ciência da Computação do Programa de Pós-Graduação em Ciência da Computação, Escola Politécnica da Pontifícia Universidade Católica do Rio Grande do Sul.

Aprovado em 21 de agosto de 2019.

BANCA EXAMINADORA:

Prof. Dr. Alexandre Lazaretti Zanatta- Avaliador (UPF)

Prof. Dra. Érika Fernandes Cota – Avaliador (PUCRS)

Prof. Dra. Sabrina Dos Santos Marczak – Avaliadora (PUCRS)

Prof. Dr. Avelino Francisco Zorzo (PPGCC/PUCRS - Orientador)

DEDICATÓRIA

Dedico este trabalho a minha prima Valentina, que com seu sorriso encantador sempre foi uma motivação para o segmento do meu trabalho.

“Me movo como educador, porque, primeiro, me movo como gente”
(Paulo Freire)

AGRADECIMENTOS

Cursar e concluir um doutorado só é possível quando ele é feito sobre o ombro de gigantes. “Um sonho que se sonha só é só um sonho, um sonho que se sonha junto se torna realidade”. Desta forma diversos são os agradecimentos que preciso registrar neste momento.

Primeiramente **Dani**, obrigada por estar do meu lado, obrigada por entender a minha ausência, gratidão por ser meu ponto de suporte, por me dar colo todas as vezes em que desanimei, por aguentar meu humor e nunca deixar de acreditar em mim, eu amo você. Depois, **Romulo e Adalto** obrigada por todas as manhãs, as tardes e as madrugadas estudando junto, ter vocês ao meu lado me motivou para seguir. **Taila** obrigada por todas as discussões, troca de conhecimentos e pelo suporte na realização do *focus group*. Obrigada também por todos os conselhos, por todos os colos que recebi e pelos milhares de momentos vividos juntas.

Ana e Pati: obrigado por terem sido as principais responsáveis pelos momentos de entretenimento e pela existência da minha vida social. Obrigada por terem me aguentado falando da tese mil vezes por dia e terem respeitado e apoiado o tempo todo. Ana, obrigada por trazer a Marina pra minha vida, **Marina** obrigada por me fazer dar as melhores risadas e me fazer criança todo dia. **Ale, Malu e Vivi**, obrigada por estarem na minha vida, torcendo por mim, me incentivando, confiando em mim sempre, desde sempre e pra sempre amo vocês.

Meu orientador professor **Avelino**, obrigada por seres um orientador no sentido mais literal da palavra, um exemplo de pessoa, um exemplo de profissional. Obrigada por entender e respeitar todas minhas limitações e me ajudar a evoluir sempre. Citando professor Avelino estendo meu agradecimento aos colegas do CONSEG e demais colegas de PPGCC. De forma muito especial agradeço ao Henry, que me auxiliou no desenvolvimento da Aquila, enquanto foi estagiário do grupo. Aos meus colegas **profs** da Universidade **La Salle**, minha profunda gratidão por cada palavra de incentivo. Aos colegas **tutores EAD** gratidão por cada abraço, cada carinho, cada risada e por uma acolhida indescritivelmente carinhosa. Prof. **Fabi**, obrigada por me mandar escrever a tese todas as noites, e por entender todos os ciclos loucos deste final de doutorado.

Carol e Diogo obrigada por todos os momentos divididos, pelas discussões, por tudo que aprendi dividindo apartamento com vocês, pelo carinho, pelo amor, pelo respeito, vocês moram em meu coração. Os **irmãos** da **academia Gracie Barra Cidade Baixa**, obrigada por todos os momentos de descontração, obrigada por todo treino pesado que aliviou a tensão e ansiedade. Especialmente, obrigada a galera que acorda de madrugada pra treinar de manhã cedo, vocês fazem meus dias muito melhores. **A família Viva e Deixe Viver**, obrigada por cada palavra de incentivo e especialmente obrigada por entenderem a minha ausência e acreditarem em mim. A psicóloga que me acompanhou nos últimos anos de doutorado, **Danielli Silva**, obrigada por ser uma profissional séria, competente e comprometida. O teu trabalho foi de extrema importância para conclusão do meu.

Por fim, porém de uma importância inestimável preciso agradecer a meus pais, **Rosane e Rudimar**, a minha mana **Angela** e cunhado **Diogo** e ao **Tio Hilario**. Obrigada por todas as orações, obrigada pelas milhões de vezes que me ofereceram ajuda, obrigada por toda a preocupação, e obrigada pelo incentivo de sempre. Citando vocês estendo o agradecimento aos demais tios e primos. Nona Angelina (*in memoriam*) obrigada por olhar por todos nós de onde você estiver.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal Nível Superior – Brasil (CAPES) – Código de Financiamento 001.

TESTE BASEADO EM MODELOS EM PROJETOS ÁGEIS, UMA ABORDAGEM BASEADA EM LINGUAGEM DE DOMÍNIO ESPECÍFICO

RESUMO

Teste baseado em modelos (MBT - Model-Based Testing) é uma das diversas técnicas que podem ser utilizadas para realização de testes de software. Esta técnica tem como diferencial, em relação às demais, o fato de mapear automaticamente todos os fluxos e com isso, gerar artefatos de testes que garantam cobertura total de um sistema. Esta característica vantajosa beneficia especialmente equipes que desenvolvem software em um ciclo de vida tradicional, isto porque, diversos trabalhos exploram a utilização de MBT neste contexto. Entretanto, no contexto de equipes ágeis de desenvolvimento de software MBT ainda é pouco explorado. Neste sentido, é proposto neste trabalho uma abordagem para aplicação de MBT em equipes ágeis (*Agile Teams* - AT). Esta abordagem se baseia na geração automatizada de modelos a partir da especificação de requisitos, realizada em cenários escritos em linguagem semi-natural. Desta forma, o trabalho de criar modelos é desonerado e MBT se torna flexível a mudanças de requisitos. Esta abordagem foi exemplificada através da criação de uma DSL, denominada Aquila, que estende a DSL Gherkin, adicionando a ela novas palavras chaves que representam de forma genérica, comportamentos do sistema e dados de testes. A abordagem proposta neste trabalho foi concebida a partir de um mapeamento de literatura e de um conjunto de entrevistas com especialistas. Estes estudos, forneceram embasamento para compreensão dos principais desafios na aplicação de MBT em AT e a partir disso, foi possível propor um conjunto de práticas para sanar estes desafios. Essas práticas foram validadas com uma *survey* em formato de questionário, e motivaram a criação da abordagem para aplicação de MBT em AT e da DSL Aquila. A abordagem e a DSL foram validadas por meio de um estudo de grupo focal (*focus group*) e de novo *survey* em formato de questionário, que foi respondido por profissionais que utilizaram a DSL Aquila em um ambiente controlado. Com a conclusão dos estudos foram obtidos resultados que permitem concluir que a utilização da abordagem proposta e da DSL Aquila, tornam viável a aplicação de MBT em equipes ágeis e podem trazer benefícios a estas equipes, em relação a produtividade (tempo e esforço) e a curva de aprendizado para automação de testes.

Palavras Chave: Testes de Software, Métodos Ágeis, DSL, MBT, Aquila, Agile, Gherkin.

MODEL-BASED TESTING IN AGILE PROJECTS: AN APPROACH BASED ON DOMAIN-SPECIFIC LANGUAGE

ABSTRACT

Model-based testing (MBT) is one of several techniques that can be used to perform software tests. This technique has as a differential, in relation to the others, the fact of automatically mapping all the flows and thus generate test artifacts that ensure total coverage of a system. This feature benefits teams that develop software in a traditional life cycle. Several studies explore the use of MBT in this context. However, in an agile development context, MBT is still little explored. In this sense, this work proposes an approach for the application of MBT in agile teams (*Agile Teams* - AT). This approach is based on the automated generation of models from the specification of requirements, performed in scenarios written in a semi-natural language. Hence, the work of creating models is exonerated and MBT becomes flexible to requirements changes. This approach was exemplified by the creation of a DSL, called Aquila, which extends another DSL called Gherkin. New keywords that represent in a generic way, system behaviors and test data area added to Gherkin. The approach proposed in this work was conceived based on a literature review and a set of interviews with experts. These studies provided the basis for understanding the main challenges in the application of MBT in AT and from this, it was possible to propose a set of practices to address these challenges. These practices were validated with a *survey* in the form of a questionnaire, and motivated the creation of the approach for the application of MBTs in AT and the DSL Aquila. The approach and the DSL were validated through a focus group study (*focus group*) and also through another *survey*, which was answered by professionals who used the DSL Aquila in a controlled environment. The obtained results allowed to conclude that the use of the proposed approach and the DSL Aquila alloes the application of MBT in agile teams. Furthermore, our approach can bring benefits to these teams, in relation to productivity and the learning curve for test automation.

Keywords: Software Testing, Agile, DSL, MBT, Aquila, Gherkin.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1. INTRODUÇÃO

Discutir alternativas aos processos e às práticas burocráticas utilizadas nas abordagens tradicionais de Engenharia de Software e Gerência de Projetos foram os principais fatores que motivaram a reunião de um grupo de profissionais da comunidade de software e com ela a emersão do manifesto ágil [5]. Este manifesto busca valorizar: “Indivíduos e interações mais que processos e ferramentas”; “Software em funcionamento mais que documentação abrangente”; “Colaboração com o cliente mais que negociação de contratos”; e “Responder a mudanças mais que seguir um plano”

Contudo, embora o manifesto ágil tenha surgido com objetivo de reduzir a burocracia no processo de desenvolvimento de software, a preocupação com a qualidade dos produtos entregues segue sendo uma constante. Inclusive, o manifesto ágil tem em sua cultura buscar a colaboração com o cliente, e, entregar um software em funcionamento, livre de falhas residuais aparentes, é um ponto de partida importante para que o cliente se sinta satisfeito, e o processo de colaboração possa fluir.

Neste sentido, diversas são as estratégias de controle de qualidade que podem ser utilizadas por empresas desenvolvedoras de software, sendo a principal, a realização sistemática de testes de software. Os testes podem ser realizados fazendo uso de diversas técnicas e aplicados em todas as etapas do processo de desenvolvimento.

Um exemplo de técnica que pode ser utilizada para realização de testes é a técnica de teste baseado em modelos (*Model-Based Testing* - MBT [13]). Esta técnica, busca estabelecer uma sinergia entre as atividades de análise e desenvolvimento e as atividades de testes de software. Esta sinergia é obtida através da reutilização de modelos comportamentais produzidos durante a análise do sistema, para a geração de artefatos de testes, por exemplo: *scripts* de teste

MBT pode ser utilizado para realização de diversos tipos de testes, por exemplo: teste de performance, teste estrutural e teste funcional. Independentemente do tipo de testes para qual se aplicar MBT, alguns benefícios desta técnica são comuns como é o caso do reuso de artefatos; do aumento da visibilidade dos requisitos; da redução dos ruídos de comunicação entre profissionais de desenvolvimento de qualidade de software e, da rastreabilidade entre o requisito documentado em modelo e o teste gerado.

Esta técnica é amplamente utilizada em equipes que trabalham com desenvolvimento de software seguindo o ciclo de vida tradicional (Cascata), contudo, até o momento tivemos contato com poucos trabalhos que exploram a sua aplicação no contexto de desenvolvimento ágil de software.

Visando atender a esta lacuna, buscamos propor neste trabalho uma abordagem para aplicação de MBT em AT. Para o desenvolvimento desta abordagem trabalhamos com a hipótese de que: *Gerar modelos a partir de cenários escritos em linguagem semi-natural, permitindo a aplicação*

de MBT a partir destes modelos, pode prover benefícios em relação a tempo e esforço para equipes de desenvolvimento ágil de software

A geração de modelos a partir de cenários permite que seja automatizada a etapa de criação de modelos, que é uma das etapas mais onerosas do processo de MBT. Além disso, possibilita a adaptabilidade dos *scripts* de teste as mudanças frequentes de requisitos que são característica de projetos ágeis. Isto acontece porque os modelos podem ser gerados de forma incremental a cada alteração efetuada nos cenários.

Na próxima seção, apresentamos o objetivo principal e os objetivos específicos desta pesquisa. Na seção 1.2, apresentamos uma síntese das contribuições desta pesquisa. Na seção 1.3 apresentamos a metodologia de pesquisa adotada neste estudo. Por fim, na seção 1.4 apresentamos a organização deste volume.

1.1 Objetivos

O objetivo principal desta tese é propor uma estratégia para aplicação de MBT em equipes ágeis, baseada na geração de modelos a partir de cenários escritos em linguagem seminatural.

São objetivos específicos deste trabalho:

- Compreender o estado da arte em testes de software no contexto de desenvolvimento ágil.
- Compreender o estado da arte em MBT no contexto de desenvolvimento ágil.
- Compreender as dificuldades, desafios, vantagens e desvantagens da aplicação de MBT no contexto de desenvolvimento ágil.
- Explorar a relação de requisitos escritos em linguagem seminatural com modelos comportamentais.
- Desenvolver uma estratégia para converter requisitos escritos em linguagem seminatural em modelos comportamentais.
- Criar uma prova de conceito para estratégia proposta.
- Realizar a validação da estratégia e da prova de conceito através da coleta de opinião de especialistas em grupo focal e em questionário.

1.1.1 Questões de Pesquisa

A fim de atender aos objetivos propostos e visando validar a hipótese de pesquisa estabelecida, formulamos as seguintes questões de pesquisa:

1. Quais informações de teste precisam ser incluídas nos cenários para que seja possível gerar modelos comportamentais que permitam a aplicação de MBT?

2. Quais informações de sistema (referente à interface dos software) precisam ser incluídas nos cenários para que seja possível gerar modelos comportamentais com informações suficientes para permitira a aplicação de MBT?

3. Qual será a relação entre as palavras chaves utilizadas nos cenários e os elementos que compõe os modelos comportamentais?

4. Como garantir que a adição de novas informações nos cenários não irá impactar na facilidade de uso e na aplicabilidade destes cenários no cotidiano das empresas?

1.2 Contribuições

Este trabalho apresenta contribuições para o segmento acadêmico e também para o segmento empresarial, apresentando para academia evolução do estado da arte e para indústria um novo recurso para facilitar o trabalho com testes em equipes ágeis.

No contexto acadêmico, apresenta-se uma evolução do estado da arte no que tange a aplicação de testes baseado em modelos aplicado ao contexto ágil de desenvolvimento de software. Esta evolução diz respeito a uma solução que integra cenários escritos em linguagem seminatural e geração automatizada de *scripts* de teste baseado em modelos. Ainda no contexto acadêmico evolui-se o estado da arte em linguagem de domínio específico apresentando uma DSL textual com domínio específico em testes funcionais de software e métodos ágeis de desenvolvimento de sistemas.

Já para o segmento empresarial a contribuição deste trabalho se dá pela geração de um protótipo de ferramenta que aplica a abordagem proposta para aplicação de MBT no contexto ágil e com ela, facilita a geração automatizada de artefatos de testes, proporcionando uma otimização de tempo e de recurso para as equipes que utilizam a ferramenta.

1.3 Metodologia

Primeiramente, a fim de compreender o estado da arte de MBT em métodos ágeis, bem como as suas vantagens, desvantagens e desafios, realizamos dois estudos preliminares: i) Revisão de Literatura e ii) entrevistas com especialistas. A partir destes dois estudos realizou-se a triangulação dos dados que deu origem a uma proposta de boas práticas para aplicação de MBT em métodos ágeis, a qual foi validada por meio de um questionário aplicado a profissionais da área.

Visando implementar as boas práticas levantadas na triangulação dos dados, propomos uma abordagem para aplicação de MBT em AT, e uma DSL que suporta esta abordagem (DSL Aquila). Para validar a abordagem e a DSL Aquila, realizamos um estudo de grupo focal e um questionário, com especialistas. A figura 1.1 sintetiza estas informações

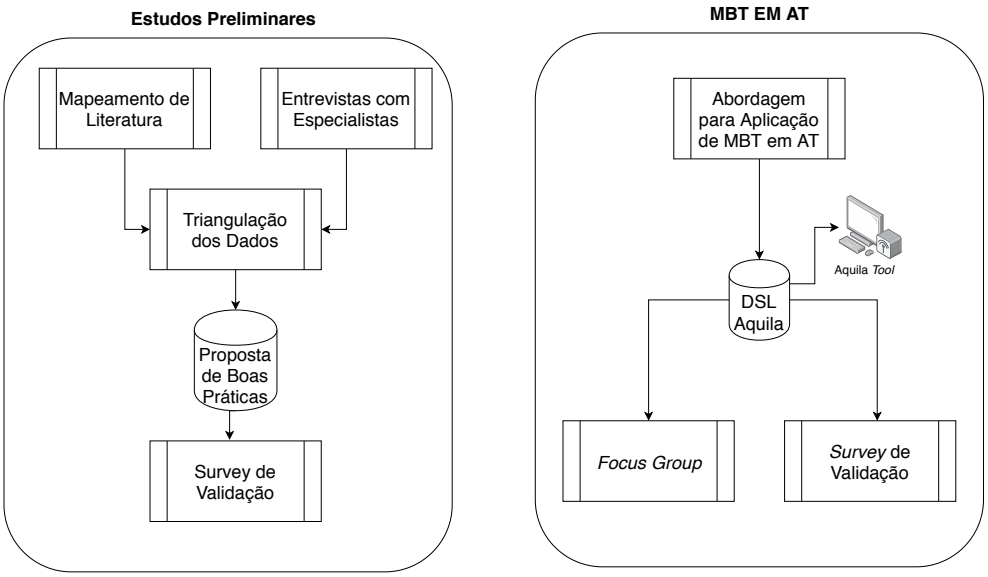

Figura 1.1 – Metodologia de Pesquisa

1.4 Organização do Volume

Este volume está organizado da seguinte forma: No Capítulo 1 são apresentados os conceitos que embasam este estudo. Em seguida, na Seção 2.8 apresentamos os trabalhos que se relacionam com esta tese e foram localizados em nossos trabalho de revisão de literatura. No Capítulo 3 apresentamos os estudos preliminares sobre MBT em métodos ágeis, que foram realizados com objetivo de conhecer o estado da arte. No Capítulo 4 apresentamos a nossa Abordagem para aplicação de MBT em AT, bem como exemplificamos esta abordagem através da proposta da DSL Aquila para a qual apresentamos um exemplo de uso como prova de conceito. Em seguida, no Capítulo 6 apresentamos os estudos de validação da abordagem e da DSL Aquila. Finalmente no Capítulo 7 apresentamos as considerações finais e trabalhos futuros.

2. FUNDAMENTAÇÃO TEÓRICA

Nesse capítulo, são apresentados alguns conceitos necessários à compreensão dessa tese, sendo eles estruturados da seguinte forma: Testes de Software (Seção 2.1); Automação de Testes (Seção 2.2); Teste Baseado em Modelos (Seção 2.3); Métodos Ágeis (Seção 2.4); Testes em Métodos Ágeis (Seção 2.5); Linguagem de Domínio Específico (DSL) (Seção 2.6); Gherkin (Seção 2.7 e os trabalhos que se relacionam com este estudo (Seção 2.8).

2.1 Testes de Software

O nome teste deriva do latin *Testum*, que significa panela de barro [12]. Historicamente, esta panela era usada para medir o peso de vários elementos, ou seja, pôr à prova [12]. Os testes estão presentes nos mais diversos segmentos, por exemplo: ao adquirir um carro é comum o consumidor efetuar um *test drive* para garantir a qualidade do veículo que está comprando. No segmento de software a recíproca se aplica igualmente, de forma que o cliente adquire um software com expectativa de recebê-lo funcionando e atendendo aos requisitos solicitados. Por este motivo adorar estratégias de verificação e validação no software, em momentos anteriores a entrega deste para o cliente é de extrema importância.

A verificação é o processo de verificar se o software está sendo construindo de acordo com os requisitos solicitados, e a validação o processo de verificar se o software está sendo construindo corretamente (Estamos construindo certo o software? Estamos construindo o software certo?) [48]. O Teste de software (ou simplesmente teste) é uma ferramenta que se aplica tanto para validação quanto para verificação e é um processo que visa, intencionalmente, encontrar defeitos de um programa ou sistema durante sua execução [39]. Nesse sentido, os testes se software são utilizados para provar a presença de falhas e não a ausência delas [26]. Esse é um dos sete princípios do teste de software estabelecidos pela *International Software Testing Qualification Board*, organização internacional que fornece certificações e materiais de estudo aos profissionais de testes.

O teste prova a presença de defeitos e não a ausência delas, uma vez que sistemas de software em geral são altamente complexos de forma que podem apresentar comportamentos distintos para uma mesma funcionalidade conforme os dados que o usuário fornecer ao sistema. Desta forma, validar exaustivamente todas as combinações de entradas disponíveis em um sistema é uma tarefa impossível [26].

Existem vários atributos que as organizações que concebem softwares devem considerar para garantir a alta qualidade desses sistemas. Para um software ser considerado confiável, precisa abranger seis atributos, que são [4]: disponibilidade, confiabilidade, confidencialidade, segurança, integridade e manutenibilidade. Além disso, também existem quatro principais técnicas que podem ser utilizadas para alcançar a confiabilidade em um sistema de software, sendo elas [4]:

- Prevenção de falhas: esta categoria foca-se na prevenção da ocorrência ou introdução de falhas;
- Tolerância a falhas: trata-se de garantir que o funcionamento do sistema não seja interrompido, causando problemas ao usuário, caso ocorra uma falha;
- Previsão de falhas: concentra-se na prevenção de prováveis falhas, para que elas possam ser removidas ou para que seu impacto para o sistema seja minimizado ou evitado;
- Remoção de falhas: foca-se em reduzir o número de falhas, detectando-as e removendo-as durante o desenvolvimento e uso do software.

2.1.1 Conceitos e Terminologias em Testes

Nesta seção, são apresentados alguns conceitos que tangem a área de testes de software e que são de grande relevância para o entendimento e exploração do assunto. Inicialmente, são contextualizados três termos que são diariamente utilizados pelos profissionais que atuam na área: erro, defeito e falha. Tratando-se desses termos, não há um consenso único sobre sua definição, visto que os pesquisadores que focam esforços nesse segmento divergem sobre tais definições.

Este trabalho tem como base os conceitos definidos por Avizienis *et al.*. [4] em que falha, erro e defeito são definidos da seguinte forma: falhas estão associadas ao universo do usuário, erros ao universo da informação e defeitos ao universo físico [4].

Outros termos importantes no segmento de testes de software, extraídos do glossário da *International Software Testing Qualification Board* [27], podem ser visualizados a seguir:

- Caso de Teste (*test case*): documento que contém um conjunto de valores de entrada, pré condições de execução, resultados esperados e pós condições. Em outras palavras, esse documento descreve as ações que devem ser feitas para testar o sistema e o resultado esperado para cada uma delas;
- Caso de Teste Abstrato: é um caso de teste de alto nível, que embora descreva entradas e saídas esperadas, não está aliado a nenhuma ferramenta ou linguagem de programação específica, um exemplo de caso de teste abstrato são as sequências de testes que descrevem um caminho que deve ser percorrido em um sistema;
- *Script* de Teste: é um caso de teste automatizado, em que as ações são executadas e os defeitos são apontadas automaticamente;
- Suíte de Teste: é um conjunto de casos de teste ou *scripts* de teste que serão executados em um mesmo momento;

A fim de atingir com excelência os objetivos dos testes de software, diversos tipos de testes podem ser empregados, entre eles os testes funcionais. Os testes funcionais de software são executados considerando o comportamento do sistema. Nesse tipo de teste, o testador avalia se o sistema está se comportando de acordo com aquilo que foi especificado nos requisitos funcionais, sob o ponto de vista técnico e também sob o ponto de vista de negócio [26].

Diversas são as técnicas que podem ser empregadas para a realização de teste funcional de software podendo estas serem executadas de forma manual ou automatizada. São exemplos de técnicas de testes funcionais: Automação de testes e Teste Baseado em Modelos. Todas as técnicas de teste funcional têm por característica comum o fato de não serem aplicadas diretamente no código fonte da funcionalidade que está sendo testada, e por isso, esse tipo de teste é chamado de teste de caixa preta.

2.2 Automação de Testes

Diversas são as tecnologias e ferramentas que podem ser utilizadas para automação de testes. No contexto de testes funcionais, a automação pode ser realizada de três principais formas: por meio de captura e repetição, de *screenshot* de telas e de programação de *scripts* de testes.

As ferramentas de captura e repetição funcionam por meio da gravação da utilização de um sistema em funcionamento. Esse processo é realizado para que, após feita alguma alteração ou incluída uma nova funcionalidade, seja possível executar a gravação e verificar se o sistema ainda está funcional. Um exemplo de ferramenta que utiliza esta técnica é o Seleniun IDE (https://www.seleniumhq.org/selenium-ide/).

As ferramentas que capturam *screenshots* da interface gráfica das aplicações comparam essas interfaces, por exemplo, com um protótipo definido para a tela. Um exemplo de ferramenta que utiliza essa técnica é a Sikuli (http://doc.sikuli.org/). Ela pode ser usada para testes multiplataforma, uma vez que pode automatizar testes de todo tipo de aplicação que possuam interface gráfica.

O tipo de ferramenta de automação de testes mais recomendado e mais utilizado atualmente é o que contempla as ferramentas que baseiam a automação de testes na programação de *scripts*. Um exemplo de ferramenta deste tipo é o Seleniun Webdriver[1]. O Seleniun Webdriver é utilizado especificamente para sistemas Web. Com ele, pode-se programar a interação com componentes de páginas web, sendo possível, por exemplo, localizar um campo na tela, clicar nesse campo, inserir valores ou apenas posicionar o mouse sobre ele.

A linguagem de programação utilizada no Seleniun Webdriver independe da linguagem utilizada pela ferramenta que está sendo testada. Dessa forma, é possível criar, por exemplo, um *script* em java para testar uma página em PHP. Isso é possível porque o Seleniun Webdriver interage com o HTML das páginas web não precisando ter acesso ao código fonte de *back-end*.

[1] https://www.seleniumhq.org/projects/webdriver/

2.3 Testes Baseados em Modelos

Testes Baseados em Modelos *Model Based Test (MBT)* é uma técnica utilizada para gerar artefatos de testes com base em modelos. O princípio básico da aplicação dessa técnica considera que os modelos são criados naturalmente de forma mental pelos envolvidos no projeto. Entretanto, quando estes modelos são formalizados podem tornar-se um guia para todo o processo de desenvolvimento do software. Além disso, a referida técnica prevê que os modelos podem ser utilizados para inserir informações de testes e, com isso, tornar-se agentes da geração semiautomatizada de artefatos[41].

Dentre os modelos que são comumente criados ao empregar essa técnica estão *Finite State Machine, Machine Statecharts, Markov Grammars* e *Unified Modeling Language (UML)*. Entre esses, a UML é o que adquiriu maior visibilidade por ser uma forma de representação de modelos completa (que suporta modelos estruturais e comportamentais) e também por apresentar conceitos do paradigma de orientação a objetos facilmente representados na estrutura dos modelos propostos por essa linguagem. [40].

Independente do modelo escolhido, a técnica de MBT segue um processo padrão que pode ser dividido em cinco etapas [16]:

- Entender o sistema em teste (*Understanding the System under Test*): A necessidade de entender o funcionamento do software e do negócio que ele representa é uma característica da maioria das técnicas de testes e, com a técnica de MBT, não é diferente. O testador precisa, antes de iniciar seus testes, ser capaz de montar um modelo mental das funcionalidades da aplicação que está testando para depois elaborar o modelo formal do software.

- Escolher o Modelo (*Choosing the Model*): Sommerville *et al.* [49] apresentam diversas técnicas para escolha de modelos em projetos de engenharia de software. Existem vários tipos de modelos com aplicações específicas, cabe ao profissional, no momento em que for iniciar o seu projeto de MBT, analisar para o contexto do seu projeto qual o modelo mais adequado.

- Construir o Modelo (*Building the Model*): Esta é a principal etapa do processo de MBT, pois é a partir dela que os artefatos de teste serão gerados nas etapas posteriores. O modelo deve ser construído levando em consideração os requisitos solicitados pelo cliente. Para que a aplicação de MBT seja possível, nesta etapa devem ser inseridas no modelo informações de testes, por exemplo, dados referentes a entradas (que refletem ações em um caso de teste).

- Gerar Testes (*Generating Tests*): A execução dessa etapa varia de acordo com o projeto e o modelo escolhido, uma vez que a geração dos modelos pode considerar diversas técnicas, como utilização de métodos de geração de sequência de testes. Exemplo: W [8], Wp [24], HSI [37] ou técnicas de caminhamento em grafos, por exemplo Algoritmo de Busca em Profundidade (*Depth-First Search - DFS* [50].

- Executar os Testes (*Running the Tests*):

  Esta etapa consiste na execução propriamente dita dos testes. Ela depende da estruturação do projeto, uma vez que, pode ser executada de forma manual ou automatizada. Caso tenham sido gerados *scripts* automatizados de testes nesta etapa acontece a execução automatizada, senão, caso tenham sido criados casos de teste nesta etapa acontece a execução manual.

- Coletar os Resultados (*Collecting Results*):

  Nesta etapa de execução dos testes são localizadas falhas, essas falhas são armazenadas em algum repositório. No momento de coleta de dados são emitidos os relatórios para registro da execução dos testes.

Diversos trabalhos científicos já foram feitos estudando métodos, metodologias, processos e abordagens para aplicação de MBT em diversos contextos [54] [41]. Entretanto, nos últimos anos, um grupo de pesquisadores têm direcionado esforços em analisar a aplicação de MBT em projetos que envolvam metodologias ágeis [34] [35] [23].

2.4 Métodos Ágeis

O conceito de métodos ágeis está estritamente ligado ao pensamento de que o software deve ser construído com ênfase na qualidade e na entrega do produto com redução de processos burocráticos. Esse conceito já vinha sendo praticado por alguns profissionais, porém, em 2001, um grupo de especialistas uniu-se para tornar pública tal forma de desenvolver sistemas. O resultado dessa reunião foi a produção de um documento intitulado “Manifesto Ágil para o Desenvolvimento de Software”, que foi popularizado como “Manifesto Ágil” [5], tornando-se a principal referência para projetos que desejam seguir essa estrutura.

Esse documento é a base filosófica dos métodos ágeis, e diversos métodos de desenvolvimento de software estão alinhados a ele. Cada método possui um funcionamento peculiar, contudo, a maioria adota ciclos curtos de desenvolvimento, que duram em média duas semanas e são chamados de iterações. Por meio dessas iterações, as equipes de desenvolvimento garantem entregas rápidas ao cliente, o que provê avaliações contínuas do produto durante o desenvolvimento, permitindo ao projeto uma maior flexibilidade para responder a mudanças de escopo [25].

O Manifesto Ágil tem como ponto preponderante a definição de quatro valores, a saber:

- Considerar os indivíduos e a interação entre eles mais do que os processos e as ferramentas;
- Priorizar o software em funcionamento mais do que a documentação abrangente;
- Priorizar a colaboração com o cliente mais do que a negociação contratual;
- Responder a mudanças mais do que seguir um plano.

A partir desses valores, pode-se identificar de forma objetiva o que difere um processo ágil de um processo tradicional de desenvolvimento de software. No contexto ágil, destacam-se o envolvimento do cliente e a redução da documentação e do tempo entre uma entrega e outra (entrega contínua). Em complemento a esses valores fundamentais, o manifesto ágil propõe doze pilares. Estes pilares prover um maior detalhamento sobre as atividades, atitudes e comportamentos esperados de uma equipe ágil de desenvolvimento de sistemas e embasam a implantação dos quatro princípios. Os doze pilares são os seguintes [5]:

1. A maior prioridade é satisfazer o cliente por meio da entrega contínua de software com valor agregado;
2. Mudanças nos requisitos são bem-vindas, mesmo tardiamente, durante o desenvolvimento. Processos ágeis tiram vantagem das mudanças visando vantagem competitiva para o cliente;
3. Entregar continuamente software funcionando, de poucas semanas a poucos meses, com preferência à menor escala de tempo;
4. Pessoas de negócio e desenvolvedores devem trabalhar diariamente em conjunto por todo o projeto.
5. A construção de projetos deve ocorrer em torno de indivíduos motivados. Eles devem ter ambiente propício e suporte necessário, e acreditar que confiam em seu trabalho;
6. O método mais eficiente e eficaz de transmitir informações para e entre uma equipe de desenvolvimento é através de diálogo;
7. Software funcionando é a medida primária de progresso;
8. Os processos ágeis promovem desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter um ritmo constante indefinidamente;
9. Contínua atenção a excelência técnica e bom design aumenta a agilidade;
10. Simplicidade, a arte de maximizar a quantidade de trabalho não realizado, é essencial;
11. As melhores arquiteturas, requisitos e designs emergem de equipes auto-organizáveis;
12. Em intervalos regulares, a equipe reflete sobre como se tornar mais eficaz e então refina e ajusta seu comportamento de acordo;

Afim de facilitar a aplicação destes valores e princípios no cotidiano de projetos de TI, diversos métodos foram criados, os quais chamamos de métodos ágeis, são exemplos: eXtreme Programming (XP) [51] e Scrun [44]. Este último que é o mais lembrado entre os métodos ágeis não é tratado na literatura como um método e sim como um *framework*.

Contudo, para trabalhar de acordo com algum método ágil e especialmente de acordo com o próprio manifesto ágil, algumas adaptações precisam ser feitas e alguns processos tradicionalmente utilizados nas equipes de desenvolvimento de sistemas precisam ser reorganizados. Uma mudança que precisa ser feita para que as equipes possam se adequar ao manifesto ágil, torná-las multidisciplinares. A multidisciplinaridade permite que o trabalho aconteça de forma colaborativa, de modo que todos os sujeitos da equipe são responsáveis pelo todo do produto, ou seja, um indivíduo não é mais responsável apenas por uma parte específica.

Seguindo essa estrutura, costumeiramente em projetos ágeis desconstroem-se papéis que representam tarefas específicas, como: desenvolvedor, testador, analista. Essa desconstrução ocorre ao lado do envolvimento de toda a equipe em todas as atividades. Por exemplo: o profissional que antes era responsável apenas pela realização de testes de software no final do ciclo de vida de um projeto cascata, agora é envolvido em todo o processo de desenvolvimento, não apenas testando, mas também auxiliando o programador em todas as etapas [9].

Nesse sentido, busca-se, na próxima seção, descrever o conceito de testes em métodos ágeis e traçar um paralelo entre os testes realizados em abordagens tradicionais e os efetuados em métodos ágeis.

2.5 Testes Em Métodos Ágeis

Testes em métodos ágeis é um conceito que ganhou bastante espaço nos últimos anos. Diversos são os trabalhos que conceituam e propõem métodos e processos para realização de testes [55, 3]. A Figura 2.1 apresenta um comparativo sobre a realização de testes em times que adotam metodologias ágeis em contraponto aos que adotam metodologias tradicionais. Este comparativo pode ser visualizado na Figura 2.1.

Na Figura 2.1 pode-se observar, acima da linha do tempo um processo tradicional e, abaixo da linha do tempo, um processo ágil (interativo e incremental). Nesse exemplo, verifica-se que a principal diferença entre os dois processos está na estrutura das etapas que cada projeto adota. Na abordagem tradicional as etapas são divididas em requisitos, especificação, código, teste e entrega ao cliente. Todas elas se referem a um mesmo programa e, o entregável só é gerado na última etapa. Nos métodos ágeis, por sua vez, cada história é analisada, programada, testada e entregue ao cliente. Desta forma, o cliente recebe continuamente pequenas partes do projeto.

Devido a tal estrutura, a realização de testes em métodos ágeis acontece durante todo o ciclo de desenvolvimento. Assim, qualquer profissional da equipe pode e deve realizar testes, e todo e qualquer profissional da equipe pode e deve programar as funcionalidades do sistema.

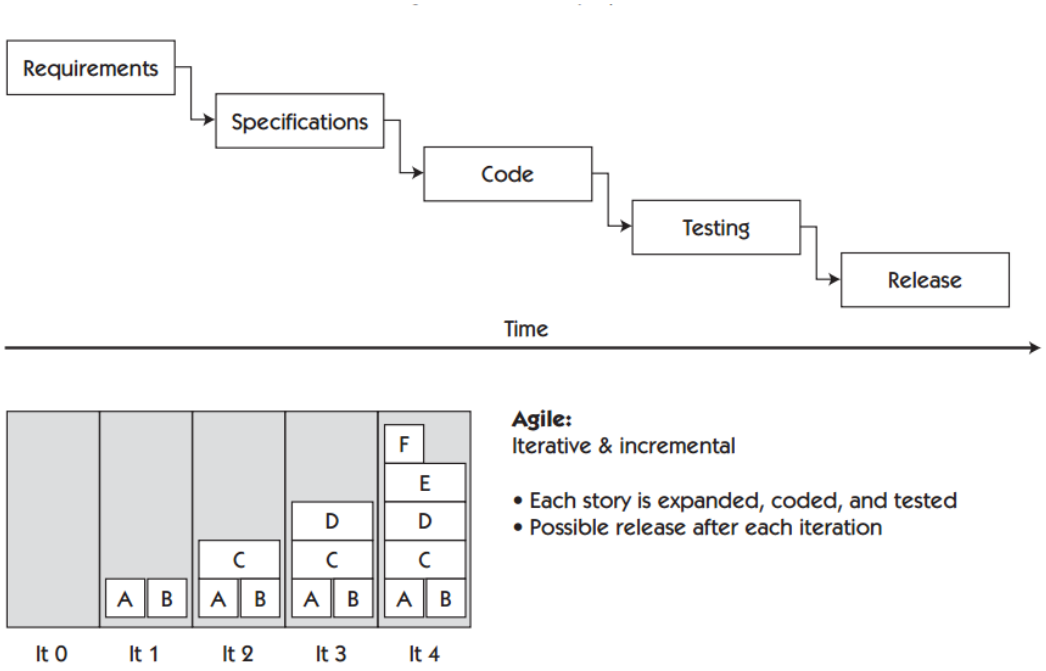

Figura 2.1 – Representação Processo Tradicional e Processo Ágil

2.6 Linguagem de Domínio Específico - DSL

Trabalhar com desenvolvimento de sistemas é uma tarefa que envolve um alto grau de complexidade. Contudo, atualmente os profissionais que atuam nesse segmento têm a possibilidade de se beneficiar de diversas linguagens de programação que podem ser utilizadas com um nível mais alto, ou seja, com uma linguagem mais próxima da humana, o que torna o desenvolvimento de sistemas mais rápido e menos oneroso.

Nesse contexto, assim como na programação utilizar linguagens de programação específicas pode beneficiar o programador; em outros segmentos da ciência da computação, a recíproca pode ser também verdadeira. Porém, para que uma linguagem específica possa ser criada, é necessário que exista um domínio específico, ou seja, um conjunto de aplicações que compartilhem características, como requisitos, objetivos, normas e procedimentos e problemas comuns que possam ser totalmente ou parcialmente solucionados com a criação e utilização de uma linguagem comum para efetuar uma determinada operação, como, por exemplo, programar, testar, analisar e implantar.

Essas linguagens são chamadas de linguagens de domínio específico (*Domain Specific Languages - DSL*) ou também conhecidas como linguagem de modelagem específica de domínio (*Domain-Specific Modeling Language - DSML*) ou modelagem específica de domínio (*Domain-Specific Modeling - DSM*). Essa nomenclatura se aplica justamente porque tais linguagens atuam em um problema específico de um determinado domínio, não podendo ser aplicadas de modo generalizado. São exemplos comuns de linguagens de domínio específicas a *Structured Query Language - SQL*, criada para realização de consulta em bancos de dados, a *HyperText Markup*

Language - HTML, utilizada para marcação de hipertextos em páginas web, a *eXtensible Markup Language - XML*, utilizada para estruturação de dados, a Gherkin, utilizada para documentação de histórias de usuários e aplicação da técnica de *Behavior Driven Development - BDD*, entre outros exemplos.

Essas DSLs, justamente por serem focadas em resolver problemas de um domínio específico, apresentam diversas vantagens para os projetos que as utilizam, uma vez que são desenvolvidas com um linguajar mais próximo da linguagem do domínio em que serão aplicadas, facilitando o aprendizado, aumentando a aplicabilidade e, consequentemente, a produtividade. Para usufruir dessas vantagens as DSLs podem ser criadas de três formas, que contemplam a forma de criação: internas, externas ou gráficas.

DSLs internas são aquelas que fazem uso de estrutura de uma linguagem de programação ou DSL (linguagem hospedeira), utilizando a infraestrutura da linguagem (ambientes de desenvolvimento e compiladores). DSLs externas, por sua vez, são aquelas desenvolvidas de forma independente contando com o desenvolvimento de uma ferramenta que possibilite traduzir, interpretar ou compilar suas diretrizes. Por fim, DSLs gráficas são aquelas que utilizam normas, linhas e elementos gráficos em vez de uma linguagem textual para representar informações do domínio.

Cada um dos tipos de DSL possui suas próprias vantagens e desvantagens na utilização. Assim, o criador da DSL deve analisar qual das três se adequa melhor ao contexto de seu projeto, estudo ou empresa. Todavia, existem vantagens e desvantagens que são comuns a todos os tipos de DSL, que serão descritas a seguir [6].

- Vantagens:
  - Melhor expressividade nas regras do domínio. DSLs permitem que soluções sejam expressas no nível de abstração do domínio do problema. Consequentemente, especialistas do domínio podem compreender, validar, modificar ou mesmo desenvolver suas próprias soluções;
  - Melhora a comunicação e colaboração entre as partes interessadas do projeto de software. No contexto de teste, otimiza a comunicação entre desenvolvedores e testadores no entendimento dos requisitos do sistema;
  - Possibilitam validação e otimização em nível de domínio.
- Desvantagens:
  - Alto custo ao projetar e manter uma DSL, principalmente, em casos de maior complexidade;
  - Alto custo na capacitação de usuários;
  - Dificuldade em definir um escopo adequado para uma DSL, necessidade de entender o padrão de domínio do problema ou envolvimento de especialistas a fim de garantir um adequado nível de abstração e expressividade da linguagem;

- Tendência em usar múltiplos idiomas: se não for cuidadosamente controlado, esta programação poliglota pode resultar em um projeto demasiadamente grande. Isto ocorre quando, visando facilitar a escrita de acordo com a linguagem vernácula de cada usuário, opta-se por traduzir sua sintaxe em diversos idiomas;
- Existe o risco de uma empresa adotar uma linguagem própria e, com isso, a posteriori, ter dificuldades de encontrar novos integrantes para o time, bem como de manter e de atualizar as tecnologias;
- Em casos de DSLs executáveis, outro problema é a perda potencial de desempenho quando comparado com código-fonte desenvolvido manualmente, entretanto, esse fator varia de acordo com a qualidade do código gerado.

Nesse trabalho considera-se o Processo de construção de uma DSL de forma incremental. Utilizar uma abordagem incremental permite que a linguagem esteja disponível para uso em um período mais curto de tempo, com os mesmos custos iniciais [7]. Mesmo efetuando a criação de uma DSL de forma incremental, o processo precisa considerar diversas etapas, que variam de acordo com o autor que as define. De acordo com Mernik *et al.* [38], o processo de construção de uma DSL é constituído por cinco etapas principais, sendo elas: decisão, análise, projeto, implementação e implantação. essas etapas são descritas a seguir.

A etapa de decisão consiste em realizar uma análise nessa fase devem ser analisados os benefícios da construção da DSL para o cotidiano de trabalho dos profissionais que irão utilizá-la e os custos e esforços necessários para a sua construção.

Depois de concluída a fase de decisão, caso a equipe considere viável e opte pela criação da DSL, é realizada a etapa de análise. Nessa etapa busca-se compreender o domínio e os problemas que este apresenta e que podem ser solucionados parcial ou totalmente pelo uso de uma DSL. Em tal fase também se define que tipo de DSL os usuários do domínio específico esperam receber para sanar seus problemas.

Na etapa de projeto, por sua vez, são tomadas decisões referentes ao desenvolvimento da DSL, como a tecnologia que será empregada no desenvolvimento e o tipo de DSL (interna ou externa). Além disso, nessa etapa, o desenvolvedor da DSL deve especificar o vocabulário, a gramática, as regras de conversão (*parsers*), a arquitetura da DSL, bem como bibliotecas ou ferramentas adicionais que serão utilizadas no projeto.

Na etapa de implementação acontece a programação da DSL. Nessa fase é desenvolvido todo o software da DSL, incluindo o compilador ou interpretador, e toda a arquitetura DSL. Por fim, é executada a fase de implantação, em que a DSL é entregue para utilização dos usuários.

2.7 Gherkin

Gherkin é uma DSL que tem por objetivo permitir a descrição do comportamento de um software, sem necessidade de detalhar como esse comportamento é implementado [11]. Essa linguagem pode ser suportada pela ferramenta Cucumber [10] e utilizada como *input* para aplicação da técnica de *Behavior Driven Development* (BDD).

A escrita de requisitos funcionais em formato de cenários Gherkin é feita em arquivos separados por funcionalidades, chamados de *features*. Os cenários Gherkin podem ser utilizados para realizar o detalhamento de *user stories*. Uma *user story* diz respeito à descrição de o que o usuário espera de uma funcionalidade, em alto nível de abstração e com baixo nível de detalhamento.

As *User stories* utilizam a seguinte estrutura: palavras-chave AS A [tipo de usuário], I WANT [alguma meta], SO THAT [por algum motivo]. Por exemplo: AS A vendedor, I WANT Um sistema de registro de clientes SO THAT registrar os clientes que efetuaram compras na loja.

Para o detalhamento de uma *User Story* são utilizados cenários Gherkin. Gherkin é uma linguagem orientada por linhas, ou seja, uma linha marca o final de uma instrução, e tem como palavras-chave principais: GIVEN, WHEN, THEN e AND. A palavra GIVEN representa o estado atual do sistema que será testado, por exemplo: GIVEN [a página de cadastro de clientes]. A palavra WHEN representa um comportamento que será realizado no sistema, por exemplo: WHEN [clicar em novo]. Já a palavra THEN representa o resultado do sistema para esse comportamento THEN [será exibida uma nova página]. A palavra AND, por sua vez, é um conectivo que mais de uma ação será executada ou que o sistema apresentará mais de um resultado.

2.7.1 Desenvolvimento Orientado por Comportamento

Desenvolvimento Orientado por Comportamento (*Behavior Driven Development* - BDD) é uma técnica de desenvolvimento de sistemas que foca no comportamento que o usuário espera do sistema que está descrevendo. A técnica de BDD objetiva descrever o que o sistema deve fazer e não como o sistema deve fazer [47], por isso é constituída essencialmente utilizando uma linguagem seminatural que permita ao cliente conversar com a equipe de desenvolvimento e ter pleno entendimento a respeito dos requisitos que estão sendo escritos. A técnica de BDD utiliza cenários escritos empregando a DSL Gherkin para estruturar projetos de desenvolvimento de sistemas e, especialmente, de testes. Isso permite a realização de uma rastreabilidade entre os requisitos (descrição do comportamento esperado do sistema) e a implementação do software. Diversas são as ferramentas utilizadas para suportar o desenvolvimento guiado por comportamento, dentre elas encontra-se a ferramenta Cucumber, que é a mais utilizada e difundida [57].

2.8 Trabalhos Relacionados

Katara *et al.* (2006) [30] apresentam uma abordagem baseada em caso de uso informal para efetuar a geração de *scripts* de testes. Essa abordagem é diferente da Aquila DSL porque a DSL Aquila usa linguagem semi-natural para gerar modelos automaticamente para aplicar MBT. Katara *et al.* não utilizam MBT, geram *scripts* a partir de caso de uso informal.

O trabalho de Entin *et al.* (2011) [19] combina as técnicas de Captura e Repetição e Testes Baseados em Modelos. A referida pesquisa se relaciona com este trabalho, já que ambos geram modelos e *scripts* de teste automaticamente. Entretanto, esta Tese considera a geração de modelos baseados em requisitos, e Entin *et al.* geram os modelos de geração baseados na técnica de captura e repetição. A técnica de captura e repetição requer que o sistema seja completamente implementado para capturar o fluxo funcional do sistema e, neste trabalho, a atividade de teste pode ser iniciada paralelo ao início do desenvolvimento do sistema.

Sivanandan *et al.* (2012) [46] apresentam uma estrutura de automação de teste orientada por comportamento usando o MBT e descrevem como ele pode ser efetivamente utilizado durante o desenvolvimento ágil. Para isso, o trabalho combina as ferramenta Graphwalker, o framework Robotiun, a técnica de desenvolvimento orientada por comportamento (*Behavior Driven Development -BDD*) e um framework Robot. A vantagem mais significativa desta Tese em relação ao trabalho de Sivanandan é que a perspectiva deste trabalho considera a geração automática dos modelos, enquanto a pesquisa de Sivanandan prevê a criação de modelos manualmente. Por outro lado, Sivanandan *et al.* geram cenários de Gherkin automaticamente a partir de modelos.

Törsel (2012) [53] apresenta um trabalho que usa Linguagem de Domínio Específico (DSL) para criar um modelo para gerar *scripts* de teste. No entanto, tal trabalho difere desta Tese porque a DSL usada para criar modelos não é linguagem natural, ou seja, tal DSL é semelhante a uma linguagem de programação.

Dwarakanath *et al.* (2017) [14] e Thummalapenta *et al.* (2012) [52] propõem abordagens em que as DSLs são usadas para gerar *scripts* de teste automaticamente. As DSLs usadas nesses trabalhos são semelhantes à DSL Aquila DSL porque todas utilizam linguagem seminatural. Contudo, Dwarakanath e Thummalapenta não usam a técnica de MBT, de modo que seus trabalhos não se beneficiam das vantagens dessa técnica. Utilizando a DSL Aquila, por exemplo, é possível perceber a combinação de fluxos alternativos do sistema para gerar *scripts* de teste que validam todos os fluxos possíveis do sistema, o que não seria possível sem o MBT.

Yue *et al.* (2015) [58] apresentam uma abordagem para realizar MBT com base em DSL. A principal vantagem da DSL Aquila, quando comparado a esse trabalho, é que a Aquila usa Gherkin para especificar cenários e gerar modelos automaticamente. Além disso, esta pesquisa deixa claro como a migração do cenário para o modelo é feita, sendo possível implementar uma extensão gráfica para exibição visual desse modelo. No trabalho de Yue, um modelo comportamental não é gerado, de modo que é difícil analisar os fluxos originados pela técnica de MBT.

Entin *et al.* (2015) [22] apresentam uma abordagem para usar linguagem natural e o DSL Gherkin para gerar modelos para aplicar o MBT. No entanto, os cenários escritos exclusivamente usando o Gherkin não possuem detalhes suficientes para gerar um modelo completo que permita a geração de *scripts* de teste. Por esse motivo, a interferência humana é necessária para criar o modelo e nenhuma estratégia é determinada para a geração automatizada de *scripts* a partir desses modelos.

Li *l.* (2016) [34] contextualizam uma abordagem para gerar cenários Gherkin a partir de modelos. Essa abordagem é semelhante à abordagem apresentada nesta Tese porque ambos trabalhos usam a DSL Gherkin. Contudo, na abordagem apresentada nesta Tese são gerados modelos a partir de cenários e *scripts* de modelos.

Albiero (2017) [2] propõe uma abordagem que faz uso de cenários BDD para a geração de *script de testes* funcionais. O trabalho do autor é focado em testes de aplicativos móveis, o que difere da DSL Aquila por esta ter seu domínio estabelecido em testes de sistemas web. Além disso, embora o trabalho do autor gere grafos para facilitar a visualização dos cenários, o autor não aplica a técnica de MBT gerando os *scripts* de teste com base em métodos de geração de sequências de testes.

A Tabela 2.1 sintetiza os trabalhos relacionados citados e expõe uma comparação sobre as características que cada trabalho apresenta e as características do trabalho proposto nesta tese (Zanin 2019). As características consideradas são: O trabalho utiliza cenários escritos em linguagem natural? O trabalho permite realizar a geração automática de modelos? O trabalho faz uso de MBT? O trabalho realiza a geração de *scripts* de teste? O trabalho permite realizar a geração de cenários a partir de modelos?

Foram elencadas estas características por serem as principais abrangidas pelo trabalho proposto nesta Tese. Analisando a tabela é possível perceber que o trabalho de Zanin é o único que aborda as quatro primeiras características, sendo que, os demais possuem uma ou mais destas mas nenhum o conjunto de todas elas. A última características, que diz respeito a geração de cenários a partir de modelos, não é parte do escopo do trabalho proposto nesta tese, sendo inclusive a engenharia reversa desta. Contudo, esta característica foi elencada por ser utilizada em diversos trabalhos e analogamente ao trabalho proposto nesta tese, utilizar também modelos e cenários.

Tabela 2.1 – Comparação entre trabalhos relacionados

	Cenários em Linguagem Natural	**Geração Automática de Modelos**	**MBT**	**Geração de scripts**	**Geração de Cenários a Partir de Modelos**
Katara *et al.* (2006) [30]			■	■	
Entin *et al.* (2011) [19]		■	■		
Sivanandan *et al.* (2012) [46]			■		■
Törsel (2012) [53]		■	■		
Thummalapenta *et al.* (2012) [52]				■	
Dwarakanath *et al.* (2017) [14]				■	
Yue *et al.* (2015) [58]	■	■		■	
Entin *et al.* (2015) [22]	■	■	■		
Li *et al.* (2016) [34]			■		■
Albiero (2017) [2]			■		■
Zanin 2019	■	■	■	■	

3. ESTUDOS SOBRE MBT EM MÉTODOS ÁGEIS

A fim de propor uma solução que possa reduzir a lacuna existente no uso de MBT em times ágeis, realiza-se, no âmbito deste trabalho, dois estudos preliminares que permitem identificar quais dificuldades ou desafios as equipes ágeis encontram na aplicação de MBT em AT. Além disso, com esses estudos, busca-se identificar pontos que podem ser melhorados na técnica de MBT para que ela se torne aplicável no contexto de AT. Os estudos preliminares foram então divididos em duas partes: 1) Mapeamento sistemático de literatura; 2) Entrevista com especialistas.

As entrevistas têm caráter exploratório, e contemplam perguntas sobre dificuldades encontradas e melhorias sugeridas para o processo de MBT, além de apresentarem algumas comparações entre MBT e outras técnicas de testes. Já no que tange ao mapeamento de literatura, além de identificar os desafios e dificuldades relatados, também se busca identificar informações a respeito das Linguagens de Domínio Específico (*Domain-Specific Language* - DSL) disponíveis na literatura para aplicação de MBT em AT. Tal estudo aconteceu porque se trabalha com a hipótese de que utilizar DSL para aplicação de MBT em AT pode tornar esse processo mais fácil e aplicável.

3.1 Estudo 1 - Mapeamento Sistemático de Literatura

Nesta seção, apresenta-se o mapeamento sistemático de literatura realizado, evidenciando o planejamento, a execução e os resultados encontrados.

3.1.1 Planejamento do Mapeamento Sistemático de Literatura

Mapeamentos de literatura são realizados com o intuito de mapear e identificar uma determinada área de pesquisa, no caso deste trabalho, a área de testes baseados em modelos. Contudo, a fim de garantir a eficácia da pesquisa e a localização correta dos artigos relacionados à área foco do estudo, utiliza-se um processo sistemático.O processo desta pesquisa baseia-se em três etapas principais, que são [32]: plano de mapeamento, execução do mapeamento e documentação do mapeamento.

O planejamento do mapeamento de literatura detalha como o processo macro será executado, de forma que ele possa ser reproduzido por outros pesquisadores. Nessa seção, apresenta-se o protocolo definido para o estudo, compreendendo informações a respeito das bases de dados utilizadas, os termos e sinônimos que formaram a *String* de busca, os critérios de inclusão, exclusão e garantia de qualidade e as estratégias de extração e análise de dados.

3.1.2 Escopo e Objetivo

O principal objetivo deste trabalho é promover uma visão geral, amparada nos trabalhos publicados entre 2001 e 2019 sobre a área de teste baseado em modelos aplicado em métodos ágeis. Objetiva-se identificar o que a academia tem pesquisado e como a indústria tem se comportado a respeito deste tema. Esta pesquisa poderá auxiliar profissionais e pesquisadores a definir que técnica, processo, método ou ferramenta utilizar e a identificar lacunas que ainda precisam ser exploradas cientificamente.

3.1.3 Definição das questões de pesquisa

A definição das questões de pesquisa deste estudo foi feita com base nos critérios PICO [32] (Population, Intervation, Comparison, Outcome), ou seja, população, intervenção, comparação e saída. As definições desses termos são as seguintes:

- População (*Population*): Pesquisas na área de Testes de Software em Desenvolvimento Ágil.
- Intervenção (*Intervention*): Abordagens Baseadas em Modelos.
- Resultados (*Outcome*): Os resultados esperados são abordagens, métodos, metodologias, técnicas, especificações e ferramentas que são usados atualmente em MBT por times ágeis de desenvolvimento de software.

A partir desses critérios, são definidas as seguintes questões de pesquisa:

QP1. Quais são as Metodologias, as ferramentas, os processos e as técnicas disponíveis na literatura que apresentam soluções para realização de teste baseado em modelos para Times Ágeis?

QP2. Entre os trabalhos selecionados na QP1, quais fazem uso de linguagem de domínio específico (Domain-Specific Language - DSL) textual, escrita em linguagem seminatural?

QP3. Quais DSLs são utilizadas para aplicação de MBT em AT?

QP4. Quais desafios são apontados pela literatura na utilização de testes baseados em modelos em times ágeis?

3.1.4 Seleção da base de dados

Para esse trabalho, optou-se por utilizar as bases de dados ACM, Compendex, IEEE, SCOPUS e SpringerLink, porque elas indexam artigos publicados nos principais periódicos e eventos da área de testes de software, como: IEEE Conference on Software Testing Validation and Verification

(ICST); International Conference on Software Engineering (ICSE); Simpósio Brasileiro de Engenharia de Software (SBES); IEEE Software (Journal); Empirical Software Engineering (Journal).

3.1.5 Seleção dos estudos

Nesta seção, são apresentados detalhes dos critérios adotados para a seleção dos estudos que formaram esta pesquisa. Tais critérios são de inclusão, de exclusão e de qualidade.

Critérios de Inclusão e Exclusão

Critérios de Inclusão e Exclusão são importantes mecanismos utilizados para identificar os estudos primários que fornecem evidências diretas sobre uma ou mais questões de pesquisa [32]. Esses critérios permitem reduzir a quantidade de artigos selecionados para análise em revisões sistemáticas de literatura, uma vez que possibilitam classificar adequadamente os trabalhos filtrados nas buscas. Como sugere o nome, trabalhos que são enquadrados nos critérios de inclusão serão incluídos no estudo, e trabalhos enquadrados nos critérios de exclusão são desconsiderados no estudo.

Para tanto, foram definidos os seguintes critérios de inclusão (CI) e de exclusão (CE):

CI1 O estudo apresenta uma contribuição ao tema de teste baseado em modelos em times ágeis;

CI2 O estudo descreve algum processo, metodologia, ferramenta ou técnica utilizada para testes de software baseado em modelos em times ágeis;

CI2 O trabalho descreve em detalhes a metodologia, ferramenta, processo ou técnica proposta, de forma que seja possível compreender esses elementos e reproduzi-los;

Critérios de Exclusão

CE1 Estudos que não se concentram explicitamente em métodos ágeis, mas que citam o desenvolvimento ágil de software em algum outro contexto (por exemplo, estudos que citam ágil como um adjetivo).

CE2 Artigos que expressam opinião ou ponto de vista, palestras, discussões, editoriais, comentários, tutoriais, prefácios, apresentações em formatos de slides sem quaisquer documentos associados, resumos estendidos e *position papers*.

CE3 Artigos que falam de testes ágeis, mas que não se aplicam a software.

CE4 Artigos que foram escritos antes de 2001.

CE5 Estudos que não estão disponíveis de forma completa e gratuita (pelo convênio CAPES e PUCRS) para consulta e extração de dados.

CE6 Artigos que não falam de testes funcionais e sim de outro tipo de teste ou então que não tem foco em testes de software.

CE7 Artigos que falam de teste baseado em modelos em times ágeis ágeis, mas que não descrevem nenhuma abordagem, limitando-se, por exemplo, à revisão de literatura.

CE8 Artigos que não estão no idioma inglês.

Critérios de Garantia de Qualidade

Critérios de qualidade (CQ) são utilizados para garantir a avaliação adequada dos estudos, como forma de mensurar a relevância de cada um deles. Para esse estudo, os critérios de qualidade definidos são:

CQ1 O estudo apresenta uma contribuição ao tema de teste baseado em modelos em times ágeis.

CQ2 O trabalho apresenta alguma validação formal dos resultados apresentados.

CQ3 O trabalho descreve em detalhes a metodologia, a ferramenta, o processo ou a técnica proposta, de forma que seja possível compreender esses elementos e reproduzi-los.

Para cada uma das questões referentes aos critérios de qualidade é utilizada a seguinte pontuação: Y (sim)= 1; P(parcialmente) = 0.5; N (não) = 0. Assim, a pontuação total (soma das três questões) pode resultar em: 0 a 1 (limitado); 1 a 1,5 (regular); 2 (bom); 2,5 (Muito Bom) e 3 (ótimo). Os artigos serão incluídos no trabalho apenas se apresentarem uma soma total maior ou igual a dois. Dessa forma, a classificação tem como aspectos de avaliação:

QA1. Y: a contribuição está explicitamente definida no estudo; P: a contribuição está implícita; N: a contribuição não pode ser identificada e não está claramente estabelecida;

QA2. Y: o estudo apresenta explicitamente a avaliação formal aplicada; P: existe uma avaliação não formal ou a avaliação formal foi considerada parcial (como nos casos em que o exemplo aplicado é muito pequeno para validar o problema); N: nenhuma avaliação foi apresentada;

QA3. Y: O trabalho descreve detalhadamente a metodologia, a ferramenta, o processo ou a técnica proposta; P: a metodologia, ferramenta, processo ou técnica proposta são descritos, mas não com detalhamento suficiente para serem reproduzidos ou compreendidos em sua totalidade; N: a metodologia, ferramenta, processo ou técnica não podem ser identificados.

3.1.6 Definição dos artigos de controle

Artigos de controle são artigos que atendem a todos os critérios de inclusão e exclusão estabelecidos para pesquisa. Apresentam relação direta com o tema que se objetiva desenvolver e trazem contribuição significativa para responder às questões de pesquisa. Esses artigos foram localizados por meio de buscas não sistemáticas na base de dados Scopus. Assim, os seguintes trabalhos são utilizados, nesta pesquisa, como artigos de controle:

Tabela 3.1 – Artigos de Controle Utilizados no Mapeamento Sistemático de Literatura

Referência	Artigo	Ano
[34]	Skyfire: Model-Based Testing with Cucumber	2016
[21]	Introducing model-based testing in an industrial scrum project	2012
[46]	Agile development cycle: Approach to design an effective Model Based Testing with Behaviour driven automation framework	2014

3.1.7 Definição da *String* de busca

Para a concepção da *String* de busca, utilizou-se um conjunto de termos e sinônimos que, de acordo com o conhecimento prévio da autora, são capazes de refletir sobre as diferentes terminologias utilizadas pela comunidade para se referir a testes e métodos ágeis.

A *String* utilizada neste trabalho é dividida em quatro quartos: G1, G2, G3 e G4. Nesse contexto, G1 se refere a termos e sinônimos utilizados para referenciar a técnica de testes baseados em modelos, considerando as diversas variações de grafia e abreviações. G2 diz respeito a termos e sinônimos utilizados para referenciar abordagens, métodos, metodologias, técnicas e processos, G3 se refere à software e G4 sinaliza termos e sinônimos utilizados para representar métodos ágeis e DSL. Optou-se por utilizar um mesmo grupo para métodos ágeis e DSL porque era objetivado utilizar uma clausula OR para que um artigo selecionado pudesse abordar métodos ágeis, DSLs ou ambos.

Devido ao número de artigos retornados pelas buscas não ser excessivo, foi possível incluir os estudos que apresentavam as condições estabelecidas pela *string* independente do local em que estivessem no texto. A formação dos grupos é a que segue:
G1: MBT OR “model-based testing” OR “model based testing” OR “modelbased test” OR “model based test” OR “model-based software testing” OR “model based software testing”
G2: approach OR method OR methodology OR technique
G3: software
G4: agile OR scrum OR xp OR “Extreme Programming” OR fdd OR “Feature-Driven Development” OR lean OR kanban OR Gherkin OR BDD OR “domain specific language” OR “domain-specific language” OR DSL

De acordo com os critérios supracitados, a *String* de busca ficou genericamente definida como:

(MBT OR “model-based testing” OR “model based testing” OR “modelbased test” OR “model based test” OR “model-based software testing” OR “model based software testing") AND (approach OR method OR methodology OR technique) AND (software) AND (agile OR scrum OR xp OR “Extreme Programming” OR fdd OR “Feature-Driven Development” OR lean OR kanban OR Gherkin OR BDD OR “domain specific language” OR “domain-specific language” OR DSL)

3.1.8 Execução do mapeamento

A execução do mapeamento sistemático seguiu um processo que compreende cinco etapas distintas, que são: Busca nas bases de dados (Seção 3.1.9), Eliminação das Redundâncias (Seção 3.1.9), Seleção Intermediária (baseada nos critérios de inclusão e exclusão) (Seção 3.1.9), Seleção Final (baseada nos critérios de Qualidade) (Seção 3.1.9) e Análise dos Dados (Seção 3.1.9). Essas etapas são sintetizadas e descritas a seguir:

3.1.9 Busca nas bases de dados

A primeira etapa do mapeamento sistemático de literatura consistiu em executar a *string* de busca nas bases de dados selecionadas. A primeira busca retornou 297 distribuídos da seguinte forma entre as bases de dados pesquisadas:

- ACM (https://dl.acm.org/): 12 trabalhos
- IEEE (https://ieeexplore.ieee.org/Xplore/home.jsp): 22 trabalhos
- Scopus (https://www.scopus.com/home.uri): 45 trabalhos
- Compendex (https://www.engineeringvillage.com/home.url): 46 trabalhos
- SpringerLink (https://link.springer.com/): 172 trabalhos

Eliminação das Redundâncias

No processo de eliminação de redundâncias, foram removidos os artigos que listados em duplicidade na busca efetuada nas bases de dados. Isso acontece porque algumas bases de dados indexam artigos publicados por outras, de modo que tais trabalhos aparecem em mais de uma base. Nesse processo, foram eliminados 73 trabalhos, restando uma lista de 224 artigos.

Seleção Intermediária (baseada nos critérios de inclusão e exclusão)

Nesta etapa, faz-se a leitura do título e do resumo de cada estudo retornado e, com base nos critérios de inclusão e exclusão, os trabalhos são incluídos ou não no estudo. No que tange aos artigos encontrados, em alguns casos foi necessário efetuar a leitura, também, da introdução, para obter maiores esclarecimentos. Após esse processo, foram selecionados 13 artigos que podem responder a uma ou mais das questões de pesquisa e que atendem aos critérios de inclusão e exclusão definidos.

Esses artigos podem ser visualizados na tabela 3.2 Observa-se que a base de dados eletrônica que concentra o maior número de trabalhos publicados é a base IEEE.

Tabela 3.2 – Artigos Selecionados na Seleção Intermediária do Mapeamento Sistemático de Literatura

Base de Dados	**Ano de Publicação**	**Titulo**	**Autor**
IEEE	2016	Skyfire: Model-Based Testing with Cucumber	Li *et al.* [34]
IEEE	2015	A test automation language framework for behavioral models	Li *et al.* [35]
IEEE	2015	A process to increase the model quality in the context of model-based testing	Entin *et al.* [22]
IEEE	2015	Automated model-based testing based on an agnostic-platform modeling language	Sanz *et al.* [45]
IEEE	2014	Agile development cycle: Approach to design an effective Model Based Testing with Behaviour driven automation framework	Sivanandan *et al.* [46]
IEEE	2014	Formal test-driven development with verified test cases	Aichernig *et al.* [1]
IEEE	2013	A Testing Tool for Web Applications Using a Domain-Specific Modelling Language and the NuSMV Model Checker	Törsel [53]
SCOPUS	2012	Towards model-based testing patterns for enhancing agile methodologies	Jalalinasab *et al.* [29]
ACM	2012	Introducing model-based testing in an industrial scrum project	Entin *et al.* [21]
IEEE	2011	Combining Model-Based and Capture-Replay Testing Techniques of Graphical User Interfaces: An Industrial Approach	Entin *et al.* [20]
SCOPUS	2007	Making model-based testing more agile: A use case driven approach	Katara *et al.* [30]
Compendex	2004	Model based testing in incremental system development	Pretschner *et al.* [43]
Compendex	2001	Model based testing in evolutionary software development	Pretschner *et al.* [42]

Seleção Final (baseada nos critérios de Qualidade)

Nesta etapa, os artigos são analisados com mais cautela por meio da leitura do texto completo (introdução, conclusão e partes específicas associadas à contribuição principal). Novamente com base nos critérios de inclusão e exclusão, e agora também nos critérios de qualidade, define-se se os trabalhos serão incluídos ou removidos do escopo da pesquisa.

Para cada artigo lido e identificado como apto para integrar o trabalho, cria-se uma ficha de leitura, buscando responder individualmente às questões de pesquisa. Nenhum artigo foi eliminado nessa etapa. A Tabela 3.3 demonstra a pontuação obtida por cada artigo.

Tabela 3.3 – Pontuação dos Artigos por Critério de Qualidade no Mapeamento Sistemático de Literatura

Trabalho	CQ1	CQ2	CQ3	Total
Skyfire: Model-Based Testing with Cucumber	1	1	1	3
A test automation language framework for behavioral models	1	1	1	3
A process to increase the model quality in the context of model-based testing	1	1	1	3
Automated model-based testing based on an agnostic-platform modeling language	1	0,5	0,5	2
Agile development cycle: Approach to design an effective Model Based Testing with Behaviour driven automation framework	1	1	1	3
Formal test-driven development with verified test cases	1	1	1	3
A Testing Tool for Web Applications Using a Domain-Specific Modelling Language and the NuSMV Model Checker	0,5	1	0,5	2
Towards model-based testing patterns for enhancing agile methodologies	1	1	1	3
Introducing model-based testing in an industrial scrum project	1	1	1	3
Combining Model-Based and Capture-Replay Testing Techniques of Graphical User Interfaces: An Industrial Approach	1	1	1	3
Making model-based testing more agile: A use case driven approach	1	1	1	3
Model based testing in incremental system development	0,5	1	0,5	2
Model based testing in evolutionary software development	0,5	1	0,5	2

Análise dos Dados

Nesta etapa, são analisados os trabalhos selecionados na etapa anterior, a fim de responder às questões de pesquisa propostas para o estudo. São elas:

1. **Quais são as Metodologias, as ferramentas, os processos e as técnicas disponíveis na literatura que apresentam soluções para realização de Testes Baseados em Modelos para Times Ágeis?**

   Foram identificados, no contexto deste estudo, treze trabalhos que apresentam doze metodologias, ferramentas, processos e técnicas que utilizam *Model Based Test* aplicado a métodos

ágeis e que atendem aos critérios de inclusão, exclusão e qualidade. A seguir, descreve-se brevemente cada um deles.

(a) Skyfire: Model-Based Testing with Cucumber - Li *et al.* [34] apresentam uma abordagem para a geração de cenários BDD automaticamente. Nessa abordagem, o testador deverá criar um modelo UML comportamental (por exemplo, um diagrama de máquina de estado). A partir disso, são gerados casos de testes, e estes são convertidos ao formato BDD.

(b) A test automation language framework for behavioral models - Li *et al.* [35] apresentam uma estratégia para a geração de casos de testes concretos a partir de casos de testes abstratos. Esse trabalho utiliza uma linguagem de domínio específica (*Domain Specific Languages - DSL*) chamada "Test Automation Language STAL" para realizar um mapeamento de elementos de diagramas de máquina de estados da UML para grafos. A partir desses grafos são gerados casos de testes abstratos e, a partir destes, uma ferramenta chamada STALE efetua a geração dos casos de teste concretos.

(c) A process to increase the model quality in the context of model-based testing - Entin *et al.* [22]. Neste trabalho é descrita criação automatizada de modelos com base nos requisitos escritos na linguagem Gherkin. A pesquisa não explora a geração de *scripts* automatizados, encerrando o processo na criação dos modelos. Tais modelos têm um alto nível de abstração porque se baseiam exclusivamente em Gherkin.

(d) Automated model-based testing based on an agnostic-platform modeling language - Sanz *et al.* [45] apresentam uma DSL para criação de modelos de teste genéricos (independente de plataforma), que podem, por meio de um *framework*, ser automaticamente transformados em testes executáveis para plataformas específicas, como por exemplo: JUnit, Selenium e QTP-HP.

(e) Agile development cycle: Approach to design an effective Model Based Testing with Behaviour driven automation framework - Sivanandan *et al.* [46] apresentam um framework para realização de teste baseado em comportamento, integrado com *Model Based Test*. Por meio desse *Framework* são utilizados modelos para geração de *Scripts* de teste e descrição de requisitos em formato de BDD.Nesse *framework*, o modelo precisa ser escrito manualmente na ferramenta YeD. A geração de *Scripts* de teste e de BDD é feita pelas ferramentas Graphwalker e Robot Framework.

(f) Formal test-driven development with verified test cases - Aichernig *et al.* [1] apresentam uma estratégia para realização de *Test Driven Development (TDD)* baseado em modelos. Nessa estratégia são criados modelos formais e, com base nesses, é realizada a geração de casos de testes. Posteriormente, os casos de teste gerados são validados e utilizados como base para a aplicação de TDD em sua estrutura tradicional, sendo que nesse momento os casos de testes gerados são implementados utilizando TDD.

(g) A Testing Tool for Web Applications Using a Domain-Specific Modelling Language and the NuSMV Model Checker - Törsel [53] apresenta uma DSL e uma ferramenta para geração de *scripts* de testes a partir de modelos. A DSL utilizada pelo autor é textual, porém não utiliza linguagem natural e é escrita na linguagem Java.

(h) Towards model-based testing patterns for enhancing agile methodologies - Jalalinasab *et al.* [29] apresentam uma estratégia para aplicação de padrões e prática utilizados em projetos que seguem um tradicional em projetos ágeis. Dentre esses padrões se encontra a técnica de MBT. Os autores explicitam a aplicação desses padrões em projetos que utilizam especificamente a metodologia ágil *Feature Driven Development*.

(i) Introducing model-based testing in an industrial scrum project - Entin *et al.* [21] apresentam um relato de experiência da aplicação de MBT em um projeto Scrum. A abordagem dos autore define todo processo de identificação e refinamento de requisitos. Inicialmente, são criados protótipos de tela navegáveis em papel. Depois, são definidas histórias de usuários que mapeiam esses protótipos. Ao iniciar uma sprint, são selecionadas as histórias que ela cobrirá e, para essas histórias,l são localizados os cenários que podem ser mapeados em modelos. Para esses cenários são criados os modelos, de forma manual. A partir desses modelos é feita a geração de *scripts* automatizados de testes.

(j) Combining Model-Based and Capture-Replay Testing Techniques of Graphical User - Entin *et al.* (2011)[20] Nesse trabalho os autores apresentam uma abordagem para combinar duas técnicas já difundidas na literatura e na academia e então aplicá-las em um contexto ágil. Tais técnicas são: Captura e Repetição (CR) e MBT. Essa abordagem consiste em criar um conjunto reutilizável de caminhos de teste, por meio de uma ferramenta de CR, e posteriormente integrar esses caminhos a modelos comportamentais que serão a base para aplicação da técnica de MBT e geração de *script* de teste. Nesse trabalho, os autores evidenciam como dificuldades para o uso de MBT em projetos ágeis que empregam a metodologia Scrum o fato de a maioria das abordagens de MBT não deixar claro o processo de migração de um caso de teste abstrato (gerado por meio dos caminhos percorridos nos modelos) para um caso de teste executável. executável. Esse processo é, em muitos casos, realizado de forma manual, o que gera um consumo excessivo de tempo e de recursos, tornando, por vezes, inviável a realização de MBT em uma *Sprint*.

(k) Making model-based testing more agile: A use case driven approach - Katara *et al.* [30] apresentam um método para realização de testes baseado em casos de uso. Os casos de uso utilizados nesse trabalho são de uso textual, e não o diagrama de casos da UML. A partir destes casos de uso são geradas sequências chamadas de *action words*, que correspondem às ações do sistema descritas com um alto nível de abstração.

(l) Pretschner *et al.* [42] [43] apresentam uma técnica para a derivação automática de casos de teste com base em *Constraint Logic Programming* (Programação de lógica de restrição).

2. **Quais DSLs são utilizadas para aplicação de MBT em AT?**

   Dos treze trabalhos selecionados acima, seis utilizam, em sua estrutura, DSLs para aplicação de MBT.

   - Skyfire: Model-Based Testing with Cucumber [34]:esse trabalho utiliza a DSL Gherkin em conjunto com a ferramenta cucumber.
   - A test automation language framework for behavioral models [35]:esse trabalho propõe uma DSL própria, a DSL STAL (*Test Automation Language*)
   - A process to increase the model quality in the context of model-based testing [22]: esse-trabalho utiliza a DSL Gherkin
   - Automated model-based testing based on an agnostic-platform modeling language [45]: esse trabalho propõe uma DSL própria para a qual não foi dado um nome específico; é chamada apenas de “modelling language”
   - Agile development cycle: Approach to design an effective Model Based Testing with Behaviour driven automation framework [46]: esse trabalho utiliza a DSL Gherkin
   - A Testing Tool for Web Applications Using a Domain-Specific Modelling Language and the NuSMV Model Checker [53]: esse trabalho também propõe uma DSL própria para a qual não foi dado um nome específico.

     O gráfico da Figura 2 apresenta a relação de DSLs utilizadas de acordo com o ano de publicação. Na Figura, os valores apresentados dentro das bolhas fazem a referência aos artigos. Pode-se notar, nessa figura que a DSL Gherkin é a única que aparece em mais de um estudo, tendo sido utilizada por trabalhos publicados em 2014, 2015 e 2016.

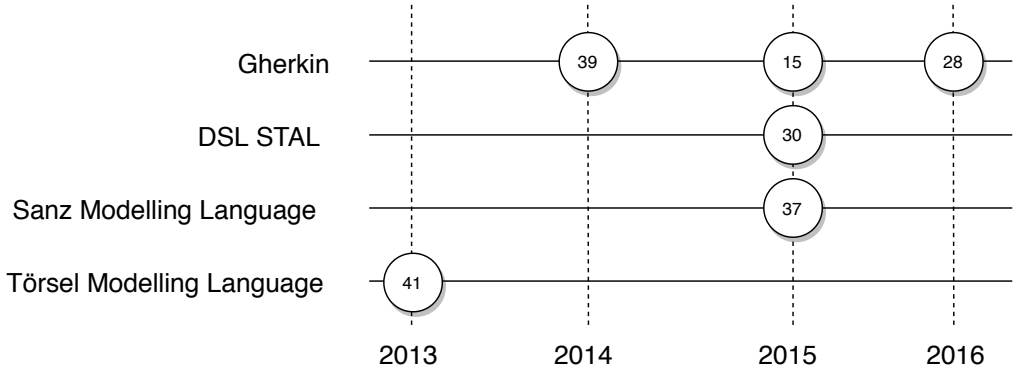

Figura 3.1 – DSLs Utilizadas Para MBT

3. **Dentre os trabalhos selecionados, quais utilizam, em sua estrutura, linguagem de domínio específica em escrita em linguagem seminatural?**

Foram localizados três trabalhos que usam os elementos supracitados, sendo eles os seguintes:

(a) Skyfire: Model-Based Testing with Cucumber [34].

(b) A process to increase the model quality in the context of model-based testing [22].

(c) Agile development cycle: Approach to design an effective Model Based Testing with Behaviour driven automation framework [46].

Analisando esses dados, pode-se perceber que, embora tenham sido localizados diversos trabalhos que apresentam a aplicação de MBT em AT, apenas três utilizam, em sua estrutura, uma DSL textual em linguagem seminatural. Os três trabalhos identificados utilizam Gherkin, sendo que os trabalhos 1 e 3 apresentam a geração de cenários Gherkin a partir de modelos, e o trabalho 2 emprega a geração de modelos a partir de Gherkin. Este, contudo, não considera a geração automatizada de *scripts*.

4. **Quais desafios são apontados pela literatura na utilização de testes baseados em modelos em times ágeis?**

   Grande parte dos trabalhos que apresentam a aplicação de testes baseados em modelos em times ágeis não descreve as dificuldades encontradas durante o processo. Contudo, as seguintes dificuldades foram identificadas:

   (a) Os modelos tradicionalmente criados por equipes ágeis normalmente não tem detalhes suficientes para aplicação de MBT [30] [29] [34].

   (b) Garantir que todos os requisitos sejam cobertos pelos artefatos de testes gerados [46] [30].

   (c) As equipes de desenvolvimento ágil, em geral, preferem ferramentas de teste fáceis de usar [30].

   (d) O processo ágil geralmente é baseado em ciclos de desenvolvimento curtos e em entregas contínuas [20] [22].

   (e) Curto tempo para criar modelos [20] [22].

   (f) Contínua mudanças de requisitos [20] [21] [46] [22].

   (g) Dificuldade de manutenção dos artefatos de testes [21] [22].

   (h) Os profissionais não possuem conhecimentos suficientes em linguagens de modelagem de software, como, por exemplo,: Unified Modeling Language (UML) [21] [22] [34].

3.2 Estudo 2 - Mapeamento Complementar de Literatura

Para a segunda revisão de literatura, utilizou-se a estratégia de busca *Snowballing* [56]. Essa estratégia é uma alternativa ao mapeamento sistemático de literatura e visa identificar trabalhos publicados, com base nas referências citadas por pesquisas importantes para a área em

estudo. Nesse contexto, inicialmente, busca-se identificar quais seriam esses trabalhos, e o ponto de partida é o artigo Skyfire: Model-Based Testing with Cucumber [34], que é o mais atual dentre os artigos de controle identificados anteriormente. A partir dele, realizou-se, primeiramente, uma Backward Snowballing e, posteriormente, um Forward Snowballing, quando foram examinados os trabalhos que citam esse artigo e os que apontam as pesquisas que foram sendo localizadas na busca anterior. Os trabalhos localizados nas duas estratégias de busca foram analisados quanto à título e *abstract* e, em caso de dúvidas, foi realizada a leitura da introdução ou até do artigo como um todo.

Além disso, foi efetuada uma busca exploratória na base de dados Google Scholar. Os artigos aí localizados foram base para novas iterações com a estratégia de *Snowballing*. A seguir, são listados os principais trabalhos encontrados:

Löffler *et al.* (2010) [36] no artigo "*Towards model-based acceptance testing for Scrum*", apresentam... apresentam uma estratégia para aplicação de MBT em projetos que utilizam a metodologia Scrum. Os autores sugerem a criação de modelos pelo product owner do projeto, por meio do refinamento de user stories, para que esses modelos sejam utilizados para geração de testes posteriores.

Elallaoui *et al.* (2015) [17] apresentam o trabalho *"Automatic generation of UML sequence diagrams from user stories in Scrum process"*. Nesse trabalho, os autores propõem gerar diagramas de sequências por meio de histórias de usuário. Embora esse trabalho se assemelhe com a linguagem Aquila por efetuar a geração de modelos, a linguagem Aquila propõe gerar diagramas de atividades com um nível de detalhamento elevado, permitindo a aplicação de MBT. Nesse trabalho os autores efetuam apenas a geração de diagramas de sequência, o que não compreende dados minimalistas sobre o sistema que permitam a aplicação de MBT.

Elallaoui *et al.* (2016) [18] no artigo "Automated Model Driven Testing Using Andro MDA and UML2 Testing Profile in Scrum Process", propõem a geração de *script* de teste baseado em diagramas de sequência. O processo apresentado nesse trabalho considera a criação de modelo de sequências simples migrando para um modelo de sequências com informações de testes. Posteriormente, os modelos completos que foram gerados são transformados em código de teste automatizado.

Dwarakanath *et al.* (2017) [15] apresentam o trabalho *"Accelerating Test Automation through a Domain Specific Language"*. Esse trabalho apresenta uma DSL para geração de *script* de teste funcional baseado em Gherkin. Essa DSL tem propósito similar à DSL Aquila, contudo, não é destinada à aplicação de MBT e, com isso , não permite que os projetos utilizem e se beneficiem das vantagens de MBT. Similar aesse trabalho King *et al.* (2014) [31] "Legend: an agile DSL toolset for web acceptance testing".esse também se baseia em linguagem natural para a geração automatizada de script de teste sem aplicação da técnica de MBT.

3.3 Considerações Sobre o Mapeamento de Literatura

O mapeamento sistemático de literatura teve como objetivo buscar os trabalhos científicos que foram desenvolvidos entre os anos de 2001 e 2019 no segmento de testes baseados em modelos no contexto de aplicação em projetos ágeis. Dessa forma, foram apresentados os trabalhos publicados no período mencionado, bem como o número de publicações por base de dados e os autores. Com esses dados, foi possível responder às questões de pesquisa e ainda ter uma visão macro do segmento de pesquisa no que diz respeito a quem são os pesquisadores que se destacam, quais linhas de pesquisa eles têm seguido e em quais bases de dados se encontra o maior número de publicações.

A partir dos dados supracitados, constata-se que a interseção entre testes baseados em modelos e métodos ágeis é uma área que teve um pico de pesquisas entre os nos anos de 2014 a 2016. Além disso, percebe-se que a maioria dos trabalhos foi publicada em conferências, o que leva a crer que são trabalhos em desenvolvimento e que, possivelmente, existirão trabalhos futuros.

Além da análise quantitativa, buscou-se responder algumas questões de pesquisa qualitativas. A primeira delas diz respeito a quais são as metodologias, as ferramentas, os processos e as técnicas disponíveis na literatura que apresentam soluções para realização de Testes Baseados em Modelos para Times Ágeis. Quanto a esse aspecto, identificou-se, em cada artigo selecionado, uma alternativa de trabalho diferente. Contudo, dentre todos os artigos analisados, foram encontrados apenas três que trabalham com DSLs e que fazem uso de linguagem seminatural, e um único que trabalha com a construção automatizada de modelos baseados nesse formato de requisito.

Entretanto, analisando os trabalhos selecionados e respondendo à questão de pesquisa número quatro, a criação manual de modelos foi apontada como o principal desafio para aplicação de MBT em AT. Isso se deve ao fato de que os projetos ágeis, em geral, trabalham com ciclos curtos de entrega contínua, o que inviabiliza destinar tempo e esforço da equipe para construir manualmente modelos para automação de testes.

Além disso, foi identificado, também, que os requisitos descritos nos projetos ágeis costumeiramente não possuem informações suficientes para criação dos modelos. A partir disso, verifica-se a necessidade de realizar mais pesquisas no que diz respeito à geração automatizada de modelos para execução de testes baseados neles e, principalmente, à geração automatizada de modelos baseados nos requisitos no formato em que as equipes de desenvolvimento ágil costumam utilizar, por exemplo: *Gherkin*.

3.4 Ameaças a Validade dos Estudos 1 e 2

A principal ameaça à validade deste estudo é a possibilidade de já existirem trabalhos publicados, porém em bases de dados cujos trabalhos não podem ser acessados publicamente.

Além disso, é uma ameaça à validade deste estudo a possibilidade de erros de interpretação na classificação dos artigos, dado que esse trabalho é feito de forma manual.

3.5 Estudo 3 - Entrevista com Especialistas

Este estudo consistiu em entrevistas realizadas com pessoas que trabalham com testes em equipes ágeis de desenvolvimento de sistemas. Esse estudo teve como objetivo principal entender a visão de profissionais que atuam com testes de software sobre a aplicação de testes baseados em modelos de desenvolvimento ágil. Mais especificamente, buscou-se entender as dificuldades e os desafios encontrados... por profissionais ao implantar testes baseados em modelos em projetos ágeis, bem como coletar opiniões acerca de como o processo de MBT pode ser melhorado para que sua aplicação seja viabilizada.

Dessa forma, foram estabelecidos os seguintes objetivos específicos para esta pesquisa:

- Entender as dificuldades e as vantagens de aplicar a técnica MBT em times ágeis
- Entender a diferença entre aplicar MBT em times ágeis e em times não ágeis
- Perceber melhorias a serem aplicadas no processo de MBT. para ser aplicado em times ágeis

Nesse contexto, foram estabelecidas algumas questões de pesquisa que embasaram a estruturação das perguntas que foram utilizadas nas entrevistas. As questões de pesquisa utilizadas são as seguintes:

RQ1: Quais as dificuldades encontradas para aplicação de MBT em times ágeis?

RQ2: Quais melhorias/adaptações podem ser feitas no processo de aplicação de MBT para que ele possa ser empregado em times ágeis?

3.5.1 Metodologia

Neste estudo, foram efetuadas entrevistas para analisar a percepção de profissionais de testes de software sobre os diversos aspectos já especificados no que tange à aplicação de MBT em AT. Essas entrevistas foram realizadas de forma semiestruturada, a partir de perguntas abertas e de perguntas específicas, o que permite que os respondentes possam trazer opiniões de amplo espectro sobre a aplicação de MBT em AT.

Foram entrevistados seis profissionais, cujos perfis são sintetizados na Tabela 3.4 e descritos individualmente na Tabela 3.5. Todos os profissionais entrevistados trabalham com testes em equipes ágeis, contudo solicitou-se que eles analisassem suas experiências anteriores nas atividades de automação de testes, desenvolvimento de software, análise de negócios, gestão ágil ou

nas atividades relacionadas com o Gherkin / BDD. Além disso, pediu-se que classificassem a sua senioridade nas seguintes categorias: iniciante (até 2 anos), intermediário (entre 2 e 5 anos) ou experiente (mais de 5 anos).

Tabela 3.4 – Perfil Profissional dos Sujeitos Participantes da *Survey* (Estudos Preliminares)

	Sem Experiência	**Iniciante**	**Pleno**	**Experiente**
Automação de Testes	0	1	4	1
Programação	2	4	0	0
Análise de Negócios	2	1	3	0
Gestão Ágil	3	0	2	1
Gherkin/BDD	0	2	3	1

Tabela 3.5 – Perfil Individual dos Sujeitos Participantes da *Survey* (Estudos Preliminares

S1	Profissional iniciante em automação de testes; pleno em análise de negócios e Gherkin/BDD; experiente em gestão de times ágeis e sem experiência com programação
S2	Profissional experiente em automação de testes e Gherkin/BDD e sem experiência em análise de negócios, gestão de times ágeis e programação
S3	Profissional pleno em automação de testes, análise de negócios e gestão de times ágeis e iniciante em programação e Gerkin/BDD
S4	Profissional pleno em automação de testes, análise de negócios, gestão de times ágeis e Gherkin/ BDD e iniciante em programação
S5	Profissional pleno em automação de testes; iniciante em Gherkin/BDD e programação e sem experiência em análise de negócios e gestão de times ágeis
S6	Profissional pleno em automação e gestão de times ágeis; iniciante em programação e análise de negócios e sem experiência em gestão de times ágeis

As entrevistas foram realizadas com uma amostra de conveniência, através de contatos pessoais e considerando os contatos pessoais da autora e o Grupo de Usuários de Testes de Software do Rio Grande do Sul (GUTSRS), um grupo ligado à SUCESU-RS. Tal grupo reúne profissionais que trabalham com testes, os quais realizam encontros mensais para troca de conhecimentos, além de interações virtuais sobre métodos, processos e ferramentas de Teste de Software.

As perguntas realizadas aos participantes foram as seguintes:

1. Sobre a criação de modelos para aplicação de Testes Baseados em Modelos (MBT)
   (a) De onde você acredita que poderiam ser extraídas as informações para inserir nos modelos?
   (b) Quem você acredita que deveria ser o responsável pela criação do modelo (PO, QA, Dev, todo time)? Por que?
   (c) Você acredita que em uma sprint, com base nos requisitos que você possui, você tem informações suficientes para efetuar a criação de modelos para aplicação de MBT?
2. Você acredita que durante uma *Sprint* é possível efetuar todo o processo de criação do modelo, de geração de casos de teste e de execução dos casos de teste gerados?

3. Em sua opinião, os baseados em modelos podem trazer algum benefício quando aplicados em um time de desenvolvimento ágil de software?

4. Em sua opinião, os testes baseados em modelos podem trazer alguma desvantagem quando aplicados em um time de desenvolvimento ágil de software?

5. Em relação ao esforço necessário para execução de MBT, você acredita que é mais trabalhoso efetuar a criação do modelo para geração automatizada de *scripts* de teste, ou executar a criação manual dos *scripts* ou casos de teste?

6. Sobre testes de regressão, em funcionalidades para as quais já exista um modelo criado, você acredita que a técnica de MBT pode trazer maiores vantagens quando comparada à execução dos testes em uma nova funcionalidade que precisa ser modelada?

7. Quais melhorias você sugere para que a técnica de MBT possa agregar maiores benefícios para o processo de desenvolvimento ágil de software?

A duração média das entrevistas foi de 20 a 30 minutos, sendo que, na maioria das vezes, foram realizadas de forma virtual. Todas as entrevistas foram gravadas e, por meio de um termo de consentimento livre e esclarecido, os envolvidos concordaram em participar e em ter suas respostas gravadas.

No início de cada entrevista explicou-se ao participante os objetivos do estudo e solicitou-se autorização para gravar a conversa. Após, efetuou-se uma demonstração do processo de MBT para os profissionais entrevistados. Essa demonstração abrangeu a criação dos modelos e a geração de casos de testes. Ao final da demonstração, foram respondidas dúvidas dos participantes em relação ao processo, e depois passou-se a realizar as perguntas. A demonstração foi feita para que os profissionais entrevistados pudessem conhecer a técnica, uma vez que não foram encontrados profissionais que tenham exercitado tal técnica na indústria. Dessa forma, as respostas foram pautadas opinião dos sujeitos sobre a possível aplicabilidade da técnica de MBT nos projetos em que cada entrevistado atua.

3.5.2 Resultados

Para analisar as entrevistas, efetuou-se a leitura das transcrições dos diálogos, visando a identificar os assuntos comuns que poderiam ser categorizados. Além disso, também voltou-se o olhar aos principais pontos do mapeamento de literatura e aos objetivos desta pesquisa. A partir disso, foram estipuladas seguintes categorias: dificuldades, melhorias e processos. No Apêndice E é possível visualizar a tabela que contém essa categorização.

A seguir, são apresentadas as respostas obtidas para cada questão de pesquisa estipulada. Tais respostas são baseadas na análise do conteúdo categorizado. Em cada resposta, são observados os principais pontos do relato do entrevistado que embasou a resposta da pesquisa.

As considerações apresentadas aqui foram citadas nas respostas de pelo menos três dos entrevistados

- Quais as dificuldades encontradas para aplicação de MBT em times ágeis?

  Em relação às dificuldades encontradas para aplicação de MBT em AT, diversos foram os pontos citados pelos entrevistados, sendo que, cada entrevistado respondeu olhando sob o ponto de vista do projeto em que atua.

  1. Disponibilidade de tempo para criação do modelo:
     S4: “quem trabalha com testes sabe que o nosso tempo de teste, de análise, enfim, todo processo, é o menor tempo. Geralmente. Então, eu acho que não, não é possível. Quer dizer, possível é, mas não é uma regra”.
     S5: “A desvantagem é que a criação do modelo é uma etapa a mais, um trabalho a mais que a equipe tem que fazer ao invés de ir na ferramenta e criar os casos, talvez demande um pouco mais de tempo”.
     S6: “Na minha realidade atual não mas acredito que no scrum bem robusto, bem feitinho é possível. Temos alguns times na empresa que são cases de sucesso onde acredito que o processo conseguiria ser executado por completo” (em resposta à pergunta sobre a possibilidade de executar todo ciclo de MBT em uma *sprint*)”.
  2. Falta de informações, dentro de uma sprint, para criar o modelo:
     S3: “Hoje eu diria que a resposta inicial seria não, porque os critérios de aceitação que nós temos nas histórias eles são critérios de aceitação de negócio e não necessariamente contemplam todos os cenários de teste” (Em resposta à pergunta “Você acredita que em um sprint com base nos requisitos que você possui, você possui informações suficientes para aplicação de MBT).”
     S5: “Não tem informações suficientes ainda, no período de baixa demanda as *stories* tem maiores informações e mesmo assim não são suficientes ”.
  3. Nível de dificuldade da criação de modelos superior a dificuldade de realização de testes manuais ou criação de *scripts* para automação:
     S1:“ Eu acho que mesmo que utilizando os modelos em relação aos testes manuais a automação sempre vai ser um pouco mais complicada”.
     S2: “é mais trabalhoso fazer um modelo que vai gerar automaticamente os casos de teste”.
  4. Necessidade de ferramentas de MBT que gerem código executável, que empreguem boas práticas e que façam isso em ferramentas comerciais e de forma automatica:
     S6: “Se o *script* for gerado com qualidade de código maneira não vejo porque ele não ser útil para o projeto neste caso, só se o código fosse muito ruim, alguma coisa que tivesse que modificar muito depois pra ter o mínimo de qualidade no próprio código de teste”.

S5: “Uma melhoria seria gerar testes automatizados, em vez de gerar excel, exportar para linguagem de programação por exemplo C, integrar com um request automatizado, com um ambiente de automação. (Adept).”

5. Aceitação da equipe para a aplicação de MBT:

   S1: “ único problema de sempre adicionar uma coisa nova é que sempre vai haver uma resistência por parte de qualquer pessoa no time de desenvolvimento”

   S2: “Porque se não tem benefício em manter a documentação, não tem como convencer uma equipe ágil a fazer isso.” (Em comentário referente a preocupação com manter os modelos).

6. Falta de conhecimento da equipe para aplicação de MBT em AT:

   S3: “A única questão que eu tenho em relação a isso, é da adoção que o mercado tem em relação com o uml, no sentido de que são poucas as pessoas que eu conheço, do meu círculo, que, de fato, usam, modelam, fazem modelagem com UML”

- Quais melhorias/adaptações podem ser feitas no processo de MBT para que ele possa ser aplicado em times ágeis?

  Para o processo de MBT se tornar aplicável em métodos ágeis, de acordo com os entrevistados, se faz necessário simplificar os processos empregados. A simplificação deve acontecer, principalmente através da automação de processos e da geração de artefatos que sejam fáceis de ser mantidos e que agreguem valor do projeto. Um exemplo desses artefatos gerados são os modelos, que precisam ser facilmente adaptáveis para se adequarem às mudanças de requisitos. Além disso o trabalho de construção dos modelos não pode ser maior do que os benefícios agregados pela técnica de MBT, por isso sugere-se a automação deste processo.

  S2: “a partir desse modelo, um sistema que a gente criou faz as verificações adequadas. Eu não preciso programar as verificações, eu não preciso ter um profissional de teste fazendo aquela customização daquelas verificações. ”

  S2: “Deveria, na minha cabeça... No papel ideal seria a equipe de desenvolvimento, equipe técnica, seja ela composta de um desenvolvedor, um *tester*, um PO (*product owner*) e tem que ser algo da equipe, não pode ser responsabilidade de um papel,” (Sobre a construção de modelos).

  S3: “UML deveria se modernizar. Eu acho que ela deveria acompanhar todas essas novas ferramentas que iam surgindo para apoiar o ágil. Talvez, tornar mais interativo essa construção ”.

  S3: “Eu acredito que qualquer um do time poderia fazer essas entradas. Se fosse considerar um modelo, assim, não vou dizer perfeito, mas, talvez, mais real, ao invés do PO registrar essas histórias na ferramenta, P.O. poderia já desenvolver a modelagem inicial, mas não vejo também problemas de qualquer pessoa do time, eventualmente, ter que fazer a tradução dessas histórias em modelagem (Sobre a construção de modelos)”.

S4: “Se não considerarmos o tempo de criação do modelo, apenas o tempo de aplicação da técnica, eu acho que eu... não vejo desvantagem. Eu acredito que daí a gente possa até encaixar a resposta da outra pergunta. Traria mais benefícios, na verdade”.

S6: “Uma melhoria seria gerar testes automatizados, em vez de gerar excel, exportar para linguagem de programação por exemplo C, integrar com um request automatizado, com um ambiente de automação. (Adept)”.

3.5.3 Ameaças a Validade do Estudo 3

A principal ameaça a validade deste estudo é o número de profissionais envolvidos. Entendemos que seis profissionais são um número pequeno dado o tamanho da comunidade de testes. Contudo, mesmo com a ampla divulgação do convite para participação não conseguimos atingir mais profissionais participantes. Reiteramos que o convite foi feito utilizando redes sociais, contatos pessoais e a comunidade de testes de software que tem sua representação ativa no Rio Grande do Sul através do Grupo de Usuários de Testes de Software - GUTS RS. Além disso, desconhecemos profissionais que trabalham com MBT em times ágeis, e por esse motivo foi necessário apresentar a técnica aos profissionais para que então eles pudessem responder. Contudo, acreditamos que pelos seis profissionais entrevistados trabalharem com metodologias ágeis, em empresas de porte grande (acima de 200 funcionários) e também localizados em locais distintos, sendo que, um dos profissionais entrevistados atua em uma empresa na Holanda e outro na Inglaterra temos dados representativos sobre aplicação de MBT em AT.

3.6 Triangulação dos dados dos estudos

Para triangulação dos dados destes estudos focamos na identificação das dificuldades que são comuns na literatura e nas entrevistas, desta forma identificamos que as seguintes dificuldades foram comuns nos dois estudos.

1. Falta de tempo para criar modelos [20] [22] e citado pelos profissionais S1,S3,S4, S5 e S6.
2. Contínua mudanças de requisitos [20] [21] [46] [22] e citado pelo profissional S5.
3. Dificuldade de manutenção dos artefatos de testes [21] [22] e citado pelo profissional S2.
4. Falta de informações para criação de modelos para aplicação de MBT [30] [29] [34] e citado pelos profissionais S1, S2, S3, S6.
5. O processo ágil geralmente é baseado em ciclos de desenvolvimento curtos e entregas contínuas (Falta de tempo em uma Sprint) [20] [22] e citado pelos profissionais S4, S5, S6.

6. Os profissionais não possuem conhecimentos suficientes em linguagens de modelagem de software, por exemplo: Unified Modeling Language (UML) [21] [22] [34] e citado pelo profissional S3.

Baseado nestes problemas comuns encontrados, e também nas melhorias sugeridas pelos profissionais entrevistados concebemos uma proposta de boas práticas para aplicação de MBT em AT.

3.7 Proposta de Boas Práticas para Aplicação de MBT em AT

Após os principais problemas e desafios encontrados pelas equipes de desenvolvimento, para cada problema identificado, propomos uma ou mais estratégias de solução através de boas práticas. Consideramos que ao seguir esse conjunto de boas práticas, as equipes de desenvolvimento obterão maior eficácia na implantação de MBT, e por esse motivo poderão usufruir melhor das vantagens que esta técnica proporciona e consequentemente agregar qualidade ao produto entregue ao cliente. Esses resultados serão atingidos através da prevenção de erros e problemas comuns nos trabalhos analisados na literatura e nos relatos dos entrevistados. Além de basear-se nos estudos preliminares, este conjunto de boas práticas também corrobora com o conjunto de boas práticas que foi proposto pela *Brazilian Software Testing Qualifications Board* [28], sendo que, o conjunto de boas práticas proposto nesta tese, tem ênfase em métodos ágeis e o proposto por [28] tem uma perspectiva genérica.

1. Conscientize a equipe da importância da criação de modelos e dê ciência dos benefícios da utilização de MBT:
    - Apresente resultados positivos reais de equipes que utilizam modelos para guiar o processo de desenvolvimento.
    - Fale sobre a técnica de MBT, apresente as vantagens, e permita a equipe que sane suas dúvidas de forma a agregar confiança na técnica.
2. Proporciona para toda a equipe de desenvolvimento capacitação para criação dos modelos.
3. Conscientize a equipe de desenvolvimento sobre a importância do trabalho em equipe e especialmente da importância dos testes de software.
4. Inicie o processo de criação de modelos juntamente com a escrita das histórias de usuário:
    - Quando são escritas as histórias de usuário pelo *Product Owner*, o mesmo efetua a criação do primeiro modelo, a exemplo do proposto por [22].

- Os modelos criados pelo *Product Owner* deverão ser evoluídos pela equipe de desenvolvimento. Esta evolução deverá ser feita sempre que forem alterados requisitos, seja esta alteração solicitada formalmente ou definida em reunião de equipe. Esta etapa visa solucionar a questão aponta na entrevista referente a falta de controle dos requisitos definidos em reuniões.
- No momento da realização dos testes de software, que pode ser paralelo ao desenvolvimento da funcionalidade, devem ser revisados e otimizados os modelos e realizada a geração dos artefatos de testes.

5. Priorize a geração automatizada de modelos e *scripts* evitando a criação manual de modelos e de casos de teste. Esta prática visa auxiliar na solução dos problemas I e IV, uma vez que gerando automaticamente os modelos e os demais artefatos de testes otimiza-se o tempo de trabalho da equipe

6. Inclua, no processo de MBT, uma forma de realizar a priorização dos *scripts* de testes gerados, de forma que o responsável pelos testes possa definir qual parte do sistema em teste deseja testar prioritariamente. Esta definição pode ser feita através de uma parametrização na ferramenta de MBT que permita definir quais atividades de um determinado modelo devem obrigatoriamente ser cobertas pelos testes. Esta prática visa atender ao desafio número V, uma vez que visa solucionar o problema de priorização dos casos de teste.

7. Busque alinhar a atividade de testes de software com os princípios do manifesto ágil, evitando que as equipes se segmentam por visualizarem a atividade de testes como um processo separado das demais atividades da equipe de desenvolvimento. Esta prática visa ajudar na solução de todos os desafios encontrados, uma vez que quando a equipe estiver alinhada aos princípios e valores ágeis, o time deverá trabalhar de forma mais integrada a fim de obter melhores resultados e se adequar melhor com as práticas supracitadas.

3.7.1 Avaliação da Proposta de Boas Práticas

Com objetivo de realizar uma validação inicial da nossa proposta de boas práticas, preparamos uma *survey* com profissionais que trabalham em equipes de desenvolvimento de sistemas para identificar qual a visão deles em relação às boas práticas propostas. Optamos pela realização desta *survey* porque acreditamos que as pessoas que estão diariamente trabalhando na indústria são as pessoas que melhor podem descrever como acontecem os processos cotidianos e que medidas podem ajudar ou não o seu trabalho.

Esta *survey* foi escrita em estrutura de questionário e distribuída de forma virtual. O questionário foi formado por perguntas para identificação do perfil do respondente e perguntas relacionadas diretamente com o processo de teste baseado em modelos e as sugestões de melhorias

propostas. A distribuição do questionário se deu através do Grupo de Usuário de Testes de Software do Rio Grande do Sul (GUTS-RS) no google groups; do facebook através da página pessoal e também do grupo TecnoTalks, do Linkedin pessoal, e através de compartilhamentos de colegas nas mesmas redes citadas. O questionário aplicado pode ser visualizado na íntegra em no link: https://goo.gl/iobm2e.

Obtivemos para esta pesquisa 30 respostas, sendo que, 11 dos respondentes trabalham com testes a mais de 6 anos, 8 responderam entre 4 e 6 anos, 7 responderam de 2 a 4 anos e 4 responderam até dois anos. Analisando as respostas obtidas podemos constatar que obtivemos um retorno positivo em relação a proposta apresentada, sendo que, a síntese das respostas obtidas será apresentada na sequência.

As propostas de melhoria número um e dois foram validadas da seguinte maneira: questionamos primeiramente se o respondente acreditava que caso uma equipe de desenvolvimento ágil seja conscientizada da aplicabilidade e benefícios de utilizar MBT, a equipe poderia motivar-se para aprender e explorar a técnica, e o resultado foi que 26 dos 30 entrevistados acreditam que sim. Além disso, perguntamos também se os respondentes acreditavam que a realização de capacitações para a criação de modelos auxilia ou viabiliza a criação de modelos durante o processo de desenvolvimento e testes de software e obtivemos para esta pergunta teve 28 respostas positivas. O conjunto destas duas respostas nos leva a crer que as boas práticas propostas números um e dois estão condizentes com a realidade das equipes.

A proposta de boas práticas número três, que diz respeito ao trabalho colaborativo, foi avaliada através da seguinte pergunta: Conscientizar a equipe da importância do trabalho coletivo, e da importância da realização de testes no processo de desenvolvimento de sistemas pode motivar os profissionais a envolver-se com a criação e edição de modelos que representem o funcionamento esperado para o software? Obtivemos, para esta pergunta 29 de respostas favoráveis e 1 contrária. O que nos leva a crer que os profissionais que foram questionados estão alinhados com a nossa proposta de que a conscientização da equipe auxilia o processo de implantação de MBT.

A proposta de boas práticas número quatro, que diz respeito a iniciar a criação dos modelos juntamente com a escrita das histórias de usuários, juntamente com as suas sub propostas, foi avaliada da seguinte forma: primeiramente questionamos os profissionais se as equipes em que eles atuam costumam construir modelos comportamentais e obtivemos 16 respostas sim.

Posteriormente, questionamos quem era responsável pela construção do modelo e nesta resposta os mais diversos papéis foram citados. Esses dados podem ser vistos na Figura 3.2. Esta questão foi de múltipla escolha, então o número de respostas pode ser mais alta do que o número de participantes porque um mesmo participante pode selecionar mais de uma opção. Neste sentido, 7 profissionais responderam que os modelos são construídos pelos desenvolvedores; 6 que os modelos são construídos pelo *Project Owner* (PO); 2 construídos pelo *Scrum Master*; 2 construídos pelo cliente. Além disso tivemos 11 respostas citando outros papéis *e.g.* testador, analista de testes e analista de sistemas. Esses dados podem ser vistos na Figura 3.2, no gráfico “Quem Constrói os Modelos”.

A diversidade na responsabilidade da construção dos modelos é um indício que esse trabalho pode ser dividido entre todos os profissionais e todo o processo de desenvolvimento de sistemas, sem sobrecarregar apenas a etapa de testes de software. Isto nos leva a crer que a proposta de boas práticas descrita no item quatro, está condizente com a realidade.

Além disso, questionamos os profissionais referente a viabilidade do PO ficar responsável pela criação inicial do modelo que representa o fluxo principal do software. Obtivemos uma aceitação de 25 dos profissionais entrevistados (Figura 3.2, no gráfico Criação do Modelo Inicial pelo PO) o que vem ao encontro ao proposto na proposta de melhoria número quatro.

A proposta de boas práticas número cinco foi validada através de duas perguntas. A primeira: A geração automatizada de *scripts* para automação de testes baseado nos modelos, pode apresentar vantagens quando aplicada em equipes ágeis? E a segunda: Uma vez criados os modelos iniciais pelo PO, se você fosse desafiado a realizar atualizações nos modelos, de acordo com a evolução do desenvolvimento, para posteriormente gerar automaticamente os artefatos de teste, qual seria a sua opinião? Para a primeira pergunta obtivemos uma aceitação de 28 profissionais (Figura 3.2 - Vantagem da Geração Automatizada) e para a segunda, 24 respondentes declararam que seriam favoráveis, 4 profissionais são favoráveis com ressalvas e apenas 1 profissional contrário (Figura 3.2 - “Realizar Atualização dos Modelos”).

Na sequência a fim de validar a proposta de boas práticas número seis, que diz respeito a gerar *scripts* de teste com prioridades distintas, ou então permitir a seleção da fração da funcionalidade que deseja testar, para que não sejam gerados *scripts* que representem toda a funcionalidade, quando deseja-se testar um fluxo específico, realizamos a seguinte pergunta: Considerando um diagrama de atividades, ou outro diagrama que representa o fluxo funcional do software de forma similar, você considera importante que possam ser gerados *scripts*, que representem apenas as funcionalidades pré-selecionadas do software representado no modelo? Obtivemos nesta pergunta 22 respondentes são favoráveis e 8 contrários. Esses dados podem ser vistos na Figura 3.2, no gráfico “Geração de *Scrips* por Funcionalidade”.

Analisando as respostas de todas as perguntas podemos constatar que com a aplicação destas boas práticas, pode-se propiciar uma facilitação da implantação de MBT em métodos ágeis, uma vez que as práticas propostas estão alinhadas com a visão dos entrevistados e resolvem lacunas específicas que foram comuns na literatura e na entrevista com profissionais especialistas na área.

3.7.2 Ameaças à validade do estudo de validação

Esse estudo de validação da proposta de boas práticas tem como principal ameaça a forma como as perguntas foram estruturadas. Uma das perguntas, que refere-se aos papéis responsáveis pela criação do modelo teve 11 respostas “outros” e esse “outros” não está descrito na pesquisa.

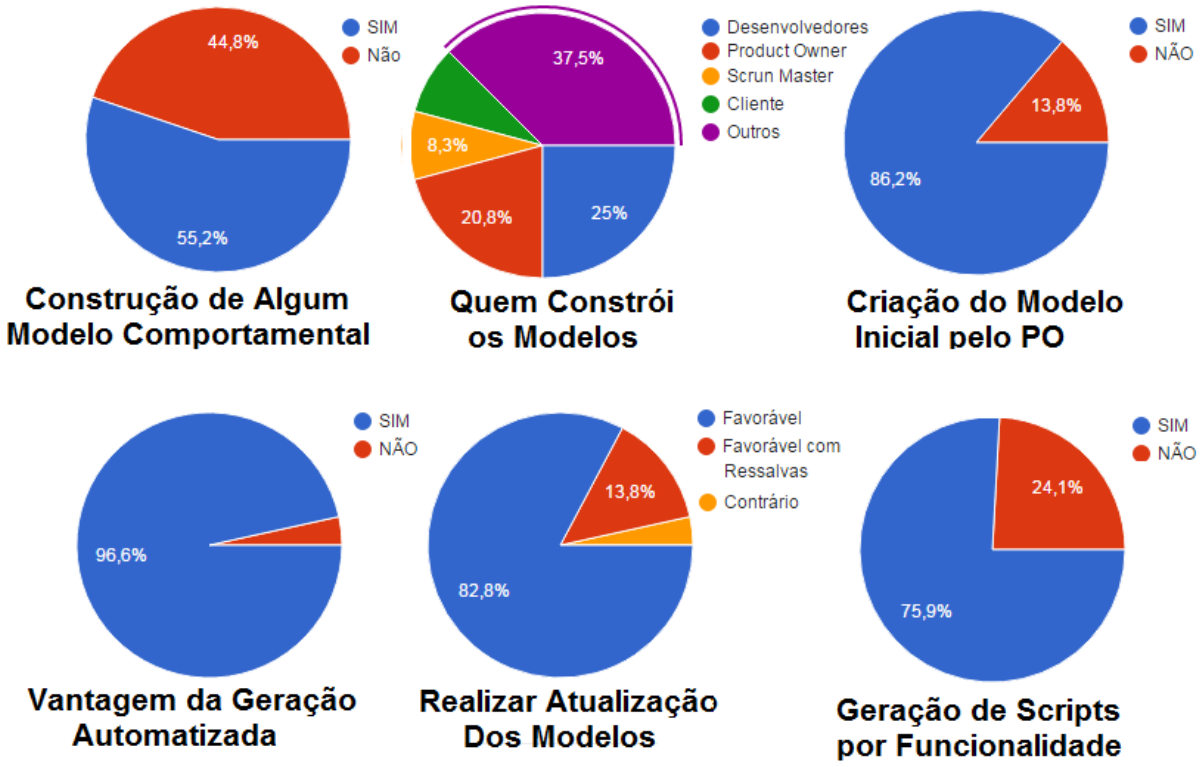

Figura 3.2 – Gráficos Contendo os Resultados da Pesquisa

Entende-se que outros envolvam profissionais de testes porque o papel do testador não foi listado entre as alternativas.

4. ABORDAGEM PARA APLICAÇÃO DE MBT EM AT

Na triangulação dos dados dos estudos preliminares, identificou-se que a principal dificuldade no processo de aplicação de MBT em AT está na criação dos modelos. Nesse contexto, propõe-se uma abordagem que possa mitigar essa dificuldade por meio da automação deste processo.

A automação do processo de criação de modelos, atende a alguns itens da proposta de boas práticas apresentada na sessão 3.7 e busca solucionar alguns dos problemas na aplicação de MBT em AT, que foram apresentados na Seção 3.6, por exemplo:

- A falta de tempo para criar os modelos é mitigada com a automação, uma vez que, o processo automatizado não consome tempo de criação de modelos;
- A dificuldade causada pela continua mudança de requisitos é solucionada porque com a automação do processo de criação de modelos, os modelos e os artefatos gerados a partir deles, podem ser atualizados facilmente adaptando-se aos novos requisitos;
- As dificuldades relacionadas a manutenção dos artefatos de testes são solucionas com a automação da geração dos modelos, porque a cada alteração nos requisitos todos os artefatos serão gerados novamente de forma automatizada mantendo-se atualizados;
- Os problemas relacionados a falta de informações para criação dos modelos são parcialmente solucionados com a adição de novas palavras chaves nos cenários que serão a base para automação da geração de modelos;
- A falta de conhecimento dos profissionais em modelagem de sistemas, também é resolvida, uma vez que, a criação do modelo é imperceptível ao profissional, este que, precisa apenas aprender escrever corretamente os requisitos em formato de cenário;

Para tornar aplicável a automação do processo de geração de modelos, a abordagem proposta nesse trabalho baseia-se em cenários escritos em linguagem seminatural, e a partir destes cenários, efetua a geração de modelos e a aplicação de MBT para geração de *scripts* de teste.

Optou-se por utilizar cenários escritos em linguagem seminatural como artefato de entrada para geração dos modelos, partindo do princípio de que usar requisitos em forma de cenários é uma estratégia já utilizada por equipes ágeis no processo de desenvolvimento de sistemas. Contudo, os cenários escritos precisam conter informações de teste e de sistema que possam embasar a geração de modelos com detalhamento suficiente para aplicação de MBT. Dessa forma, propõe-se que os cenários escritos possuam as seguintes características:

- Deve existir uma forma genérica de representar que todas as informações obrigatórias solicitadas pelo sistema serão preenchidas de forma correta pelo usuário

- Deve existir uma forma de representar ações que têm características específicas:
  - clique
  - movimentação de mouse
  - inserção de dados
  - ausência de dados
- Deve existir uma forma de representar resultados que são esperados do sistema para alguma ação:
  - Desabilitar ou habilitar campos
  - exibir mensagens
  - abrir páginas específicas
- Deve ser permitido informar mais de uma entrada para o mesmo cenário, para que por meio da técnica de MBT possam ser gerados múltiplos *scripts* combinando estas entradas

A partir destes cenários, considera-se um processo de MBT formado por quatro etapas, as quais são ilustradas na Figura 4.1,estas etapas tem um funcionamento cíclico, podendo ser reescritos os cenários ou criados novos cenários apos cada geração de *scripts*. Estas etapas são descritas a seguir:

1. Escrita dos cenários: inicia-se concomitantemente à análise do sistema é realizada pelos profissionais de testes e de negócio. Nessa etapa, são escritos os cenários contendo as informações necessárias para geração de modelos (conforme supradescrito).
2. Evolução dos cenários: é efetuada durante todo o ciclo de desenvolvimento, por todos os profissionais, nesta etapa são adicionados por exemplo: nomes de campos e possíveis dados entradas para cada campo ou ainda podem ser criados ou removidos cenários conforme necessidade do cliente.
3. Geração de modelos: Os modelos são gerados automaticamente por uma ferramenta que suporta a aplicação de MBT em AT (Aquila *Tool*, ver Seção 4.1).
4. Geração de *scripts* de testes: a partir dos modelos, são gerados *scripts* de testes.

4.0.1 Comparação do processo tradicional de MBT com o processo proposto neste trabalho

Conforme descrito na etapa de fundamentação teórica, o processo de MBT tradicionalmente proposto por[16] compreende cinco etapas, sendo elas: 1) Entender o sistema em teste

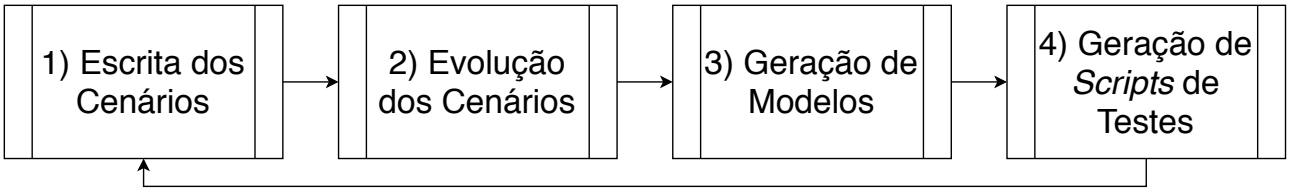

Figura 4.1 – Abordagem Para Aplicação de MBT em AT

(*Understanding the System under Test*), 2) Escolher o Modelo (*Choosing the Model*), 3) Construir o Modelo (*Building the Model*), 4) Gerar Testes (*Generating Tests*) e 5) Executar os Testes (*Running the Tests*.

Na abordagem proposta nesta tese, o processo de MBT passa a ter quatro etapas executadas de forma cíclica, contudo, a grande diferença entre os dois processos não diz respeito ao número de etapas, mas sim a forma de realização de cada uma das etapas e complexidade envolvida. As primeiras duas etapas do processo tradicional de MBT foram concatenas na etapa um do processo proposto nesta tese. Isto porque ao escrever os cenários o profissional toma conhecimento do sistema, e como o modelo será gerado baseado na descrição de cenários o tipo de modelo é impositivo (comportamental).

A etapa três do processo tradicional de MBT, que diz respeito a construção do modelo foi substituída pelas etapas dois e três do processo proposto nesta tese. Isto porque, agora os modelos não são mais construídos manualmente e sim gerados de forma automatizada a partir dos cenários. Contudo, para que isso se torne viável se faz necessário evoluir os cenários adicionando informações mais detalhadas de teste e de sistema.

A etapa quatro do processo tradicional de MBT é muito similar a etapa quatro do processo proposto nessa tese. A principal diferença entre elas está no fato de que apenas no processo proposto dessa tese são gerados *scripts* de testes. Para a geração destes *scripts* o modelo gerado na etapa anterior é percorrido gerando sequências de testes e então, considerando uma ferramenta de testes previamente escolhida pelo time de testes são convertidas as sequências de forma automatizada em *scripts*.

O modelo proposto por esta tese não considera a etapa de execução uma vez que esta etapa será feita de forma completamente automatizada com alguma ferramenta de testes disponível comercialmente.

4.1 Aquila - Uma DSL para testes funcionais ágeis baseados em modelos

Visando propor uma alternativa facilitada para a realização de MBT e considerando um enfoque principal em equipes que atuam com metodologias ágeis de desenvolvimento de software, propomos neste trabalho a DSL Aquila. Esta DSL propõe estender a DSL Gherkin para tornar possível a criação automática de modelos para aplicação de MBT a partir dos cenários escritos em

linguagem natural. Esta extensão deverá ser feita através da agregação de palavras chaves, que representam informações de sistema e são necessárias nos modelos para que seja possível aplicar MBT. Dentre essas informações se encontram referências às ações que precisam ser executadas no sistema para realizar um teste, por exemplo: preenchimento correto dos campos presentes na interface gráfica de determinada funcionalidade.

A geração automatizada de *script* de teste a partir de modelos é um processo já existente e aplicado por diversos pesquisadores. Contudo, estes trabalhos consideram a criação manual dos modelos ou a geração de modelos a partir de código fonte, não considerando a geração de modelos a partir de cenários Gherkin. A criação manual de modelos, em processo de desenvolvimento ágil, costuma ser citada como um dos principais pontos que dificultam a implantação de MBT, principalmente devido aos ciclos curtos de entrega contínua aplicados nas metodologias ágeis [19]. Já a geração de *script* de teste a partir de código fonte apresenta desvantagens uma vez que, pode ocasionar a realização de testes tendenciosos, sendo que, um dos princípios do teste de software segundo [27] é a realização dos testes com base nos requisitos do sistema e não com base no sistema implementado. Contudo, a desvantagem supracitada não se aplica nos casos de testes de regressão que visam garantir que um sistema continue em funcionamento após uma alteração de código [27].

Diferente do que propõem estes dois tipos de estratégias, a DSL Aquila permite a geração de *script* de teste baseado exclusivamente: nos requisitos especificados; nos padrões estabelecidos e no conjunto de entradas e saídas esperadas para o sistema. Esta característica faz com que os artefatos de teste gerados representem aquilo que o sistema deve fazer e não aquilo que o sistema implantado faz, aumentando assim a eficiência dos testes. Nas próximas seções apresenta-se a definição do domínio, arquitetura e requisitos desta DSL.

A principal DSL utilizada na automação de testes de software é a DSL Gherkin, contudo, para que cenários Gherkin possam ser utilizados para a geração de modelos com o nível de detalhamento necessário para aplicação de MBT, os cenários precisam ser excessivamente descritivos, isto é, conter o detalhamento de todas as ações que o usuário pode realizar no sistema, especificadas individualmente. Este processo de especificar as ações individualmente não é uma boa prática, porque prejudica a manutenção dos cenários.

Portanto, identifica-se a necessidade de criar uma DSL que atenda a condição de permitir a geração de modelos para aplicação de MBT, sem dificultar a manutenção dos cenários. Nesse contexto, propõe-se, neste trabalho, a DSL Aquila.

A principal etapa da construção de uma DSL é a definição do domínio para o qual ela está sendo criada. O domínio para o qual se aplica a DSL Aquila é testes funcionais de software em equipes ágeis. A partir da definição do domínio, foram criados requisitos que determinam as principais características que a DSL precisa conter para contemplar as necessidades do domínio específico e decisões de projeto que definem como estes requisitos serão atendidos. Tais requisitos são mencionados a seguir:

R1: A DSL deve ser textual.
R2: A DSL deve utilizar linguagem de fácil acesso e comum às pessoas que fazem parte do domínio específico de testes de software;
R3: A DSL deve utilizar linguagem seminatural evitando ambiguidades e vícios de linguagens proporcionadas pela utilização de linguagem natural.
R4: A DSL deve possuir termos que representam comportamentos genéricos e comportamentos específicos do usuário no sistema.
R5: A DSL deve possuir termos que representem o resultado esperado do sistema para cada possível ação do usuário.
R6: A DSL precisa permitir o mapeamento do cenário para modelo e para *scripts* de teste.
R7: Por meio da DSL deve ser possível informar os dados que serão utilizados como entrada para campos específicos do sistema, permitindo inclusive a especificação de diversos valores para um mesmo campo.
R8: Por meio da DSL deve ser possível combinar as múltiplas entradas especificadas para os campos da funcionalidade, visando gerar *scripts* que garantam que diferentes valores de entrada não causem comportamentos diferentes no sistema.
R9: Por meio da DSL deve ser possível a geração de *scripts* de teste que garantam a cobertura de todo o sistema documentado nos requisitos.
R10: A DSL projetada deve permitir a geração de artefatos de testes para aplicações multiplataforma e independentes.

Para atender a estes requisitos, cada um deles foi associado a uma ou mais decisões de projeto. Estas decisões concedem um embasamento para o projeto e o desenvolvimento da DSL.

D1: A DSL Aquila estende a DSL Gherkin, adicionando palavras-chave que representam comportamentos do usuário no sistema e resultados esperados para estes comportamentos.
D2: Para representação dos cenários em modelos deve ser utilizado grafo acíclico direcionado (*directed acyclic graph* - DAG).
D3: A partir do modelo gerado deve ser utilizado um algoritmo de caminhamento em grafo por busca em profundidade (*depth-first search* - DFS)[50] para garantir que todo o grafo seja percorrido gerando artefatos de testes.
D4: Cada palavra chave Aquila irá representar uma nova vértice no DAG.
D5: Quando for utilizada mais de uma entrada para a mesma palavra chave, será gerado uma vértice no DAG para cada uma delas, por exemplo: para uma palavra chave que representa inserção de valores, e para qual foram especificados os dados de entrada: a, b e c, serão geradas três vértices.
D6: As entradas serão combinadas por meio do algoritmo DFS garantindo que todas as combinações de entradas sejam testadas.

D7: Embora a DSL Aquila utilize como base a DSL Gherkin, ela não será projetada para realização de BDD *Behavior Driven Development* isto é, não irá gerar métodos separados que es-

truturam o desenvolvimento de software a partir das palavras chaves Given, When e Then, como acontece quando se utiliza Gherkin na ferramenta Cucumber. O Objetivo da DSL Aquila será exclusivamente gerar *scripts* de testes completos e reaproveitáveis. Considerando que os *scripts* serão gerados baseados nos cenários Gherkin convertidos em modelos, a rastreabilidade entre requisito e teste estará explicitada no modelo, não sendo necessária a aplicação de BDD para este fim.

A decisão de estender Gherkin e não criar uma DSL externa, foi devido ao fato de que Gherkin é consolidada entre as equipes de desenvolvimento ágil de software. Por isso, a curva de aprendizagem e o tempo de adaptação dos profissionais para utilização de uma DSL baseada em Gherkin tende a ser inferior do que se fosse criada uma DSL externa com uma sintaxe completamente diferente da habitual.

Na DSL Gherkin, utilizam-se as cláusulas GIVEN, WHEN e THEN para criação de uma documentação viva, que é utilizada como entrada para criação de métodos de teste. Contudo, BDD não permite a geração automatizada dos *scripts* de testes, e sim de uma estrutura de métodos que se baseiam nos cenários escritos em Gherkin para guiar e organizar a programação dos *scripts*. Nesse contexto, aplica-se a DSL Aquila como uma extensão da linguagem Gherkin, visando permitir, por meio da geração automatizada de modelos comportamentais, a aplicação da técnica de MBT para geração de *scripts* de testes.

A partir dos cenários escritos no padrão Aquila, são gerados DAGs. A decisão de utilizar DAG se deu pelos seguintes motivos:

1. É necessário que o modelo seja direcionado para identificar com maior facilidade o fluxo do sistema.

2. Os modelos gerados serão acíclicos, isto é, não retornam do ponto final para o ponto inicial, uma vez que esse retorno daria início a um novo cenário de teste, representado em um novo modelo;

3. O DAG atende de forma simples todas as necessidades da DSL Aquila, permitindo a representação das TAGS em vértices e a aplicação do algoritmo de DFS.

A decisão de utilizar o algoritmo de DFS, para geração das sequências de teste foi tomada porque este algoritmo é tradicionalmente utilizado para localização de valores em grafos por meio de busca em profundidade. Por percorrer o grafo buscando valores, este algoritmo tem a capacidade de mapear todos os possíveis caminhos que podem ser percorridos do ponto inicial ao ponto final do grafo. Para a geração de artefatos de testes, estes caminhos são chamados de sequências de testes, porque dado que o modelo representa uma funcionalidade, cada caminho representa um conjunto de ações que o usuário deve seguir ao utilizar essa funcionalidade, e o conjunto de todos os caminhos garante a geração de combinações que cobrem todas as ações.

Para que seja possível gerar, pelos cenários, os modelos que são a base para aplicação do MBT, dos quais são extraídas as sequências de testes, é necessário que os cenários sejam detalhados com o uso de novas palavras-chave. Estas palavras-chaves representam as interações

que o usuário pode fazer com o sistema, por exemplo: inserção de valores em campos e clique em botões.

A seleção destas palavras chaves foi baseada na interação do usuário com os principais tipos de *inputs* de campos de formulários HTML definidos pela W3C [1]. Além disso, também foram consideradas as ações que podem ser feitas utilizando a ferramenta Selenium Webdriver. Embora a Aquila seja projetada para permitir sua utilização em diversas linguagens e ferramentas, ela foi criada tendo o Selenium Webdriver como base. Essa decisão foi tomada porque ele é uma ferramenta amplamente difundida e utilizada como base para outras ferramentas, como o Appium, por exemplo, que tem sua sintaxe muito parecida com a do Selenium.

Cada palavra chave Aquila é usada associada a uma palavra chave Gherkin, sendo que, palavras-chave que representam o status do sistema (por exemplo a página inicial do teste) são utilizadas junto com a palavra chave Gherkin GIVEN; palavras-chave que representam comportamentos (por seleção de valores em combobox) são utilizadas junto com a palavra chave Gherkin WHEN e palavras-chave que representam o resultados (por exemplo um determinado campo deve ficar inativo) são utilizadas junto com a palavra chave Them. Sendo que, junto com WHEM e THEM pode ser utilizado AND para inserção de novas informações.

Este cenário fica mais evidente nos casos em que a equipe utiliza cenários Gherkin altamente descritivos, isso é, cenários onde todo detalhamento de ações de usuário é centralizado nos cenários. Esta hipótese é levantada porque, embora sejam adicionadas informações funcionais detalhadas nos cenários descritivos, os *scripts* de testes continuam não sendo gerados automaticamente, demandando com isso, esforço de desenvolvimento.

Além disso, quando se utiliza BDD com cenários altamente descritivos um grande número de métodos serão gerado para que neles sejam criados os *scripts*, ao passo que, utilizando as palavras chaves Aquila os *scripts* são gerado automaticamente e a manutenção deve ser realizada apenas nos cenários.

A Tabela 4.1 apresenta as palavras-chave definidas para a DSL Aquila (Coluna 1), bem como qual palavra chave Gherkin é associada a ela (coluna 3) e a descrição do comportamento representado por cada uma delas (coluna 2).

Alguns cenários podem requerer a seleção de valores, é o caso dos cenários que contêm as palavras-chave: put, use-valid-data, checked, choose, e select-data. Nesse caso é criada uma tabela onde são especificados os nomes dos campos e os seus respectivos valores. Na Figura 4.2 é possível visualizar um cenário que necessita valores de entrada para os campos “pais” e “estado”, e a tabela com estes valores.

No caso de cenários que contém a palavra chave use-valid-data é necessário especificar na Tabela todos os campos que serão preenchidos e os valores para estes campos como é exemplificado na Figura 4.3:

[1] principal organização de padronização da World Wide Web https://www.w3schools.com/html/html$_{f}orms.asp$

Tabela 4.1 – Palavras-Chave Aquila

Palavra chave Aquila	**Definição**	**Palavra chave Gherkin**
< >	Utilizado para informar uma URL ou caminho de um aplicativo	GIVEN
{ }	Usado para referenciar um cenário dentro de outro cenário	GIVEN
click[field]	Representa um clique em um campo	WHEN
click-link[field]	representa um clique em um link	WHEN
put[field]	Representa a inserção de valores em um campo	WHEN
use-valid-data	Representa que todos os campos presentes na tela e especificados na tabela de entradas receberão valores válidos	WHEN
dont-fill-out[field]	Representa que um determinado campo será deixado em branco	WHEN
checked[field]	Representa a seleção de um valor em um checkbox	WHEN
choose[field]	Representa a seleção de um valor em um radio button	WHEN
select-data[field]	Representa a seleção de um valor em um list	WHEN
mouse-over[field]	Representa a movimentação do mouse para cima de um elemento	WHEN
enable[field]	Usado para verificar se um campo está ativo	THEN
disable[field]	Usado para verificar se um campo está inativo	THEN
showed[field]	Verifica a presença de uma palavra ou frase no sistema em teste	THEN
opened[url]	Representa a abertura de uma determinada página	THEN
showed-title[word]	Representa a verificação do título de uma página	THEN

When I **select-data[**Pais**]**
| **Pais** |
| USA|
And I **select-data[**Estado**]**
|**Estado**|
| Nevada|
| Texas|

Figura 4.2 – Exemplo de Campos que Requerem Entrada de Dados

When I **use-valid-data:**
|**Pais**| **Estado**| **E-mail**| **Nome**| |França|Loire|alice@test.com| Alice|

Figura 4.3 – Exemplo de use-valid-data

4.1.1 Geração de Modelos e *Scripts* a partir dos cenários Aquila

Nesta seção, demonstra-se como acontece o processo de geração de modelos e *scripts* de testes a partir de cenários. O fluxo completo é formado por seis etapas (veja na Figuras 4.4). Destas seis etapas duas requerem interação humana (escrita dos cenários Aquila e Melhorias nos *scripts* gerados); três etapas são realizadas de forma automática pela ferramenta que suporta a DSL Aquila, chamada *Aquila tool* (geração dos modelos, geração das sequências de testes e geração dos *scripts* de testes); e uma deve ser executada com uma ferramenta de automação escolhida pelo testador (execução dos *scripts* de testes).

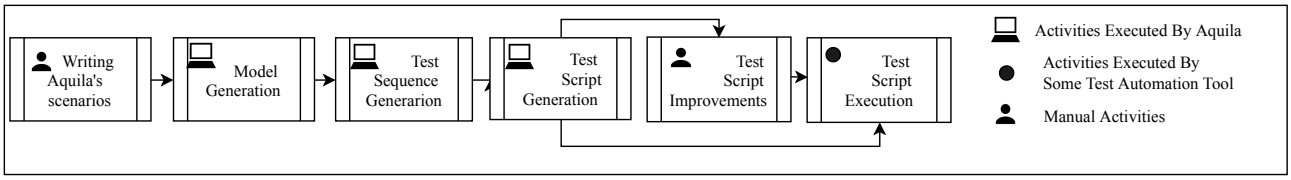

Figura 4.4 – Fluxo de Geração de Modelos a Partir de Cenários

1) Escrita dos cenários Aquila: A escrita dos cenários Aquila pode acontecer da seguinte forma:

1) dentro da ferramenta Aquila *Tool* utilizando seu editor de texto próprio (Figura 4.5);

2) fazendo uso de um *plugin* desenvolvido para o programa notepad++ (Figura 4.6) ou então

3) utilizando bibliotecas e ferramentas que suportam a escrita de Gherkin, por exemplo Cucumber.

No caso de optar pela terceira opção o usuário não terá informações visuais da escrita correta das palavras-chave Áquila, uma vez que, estas bibliotecas reconhecem apenas Gherkin, na primeira e segunda opção, o usuário terá estes *feedback* do uso correto das palavras-chave em tempo real.

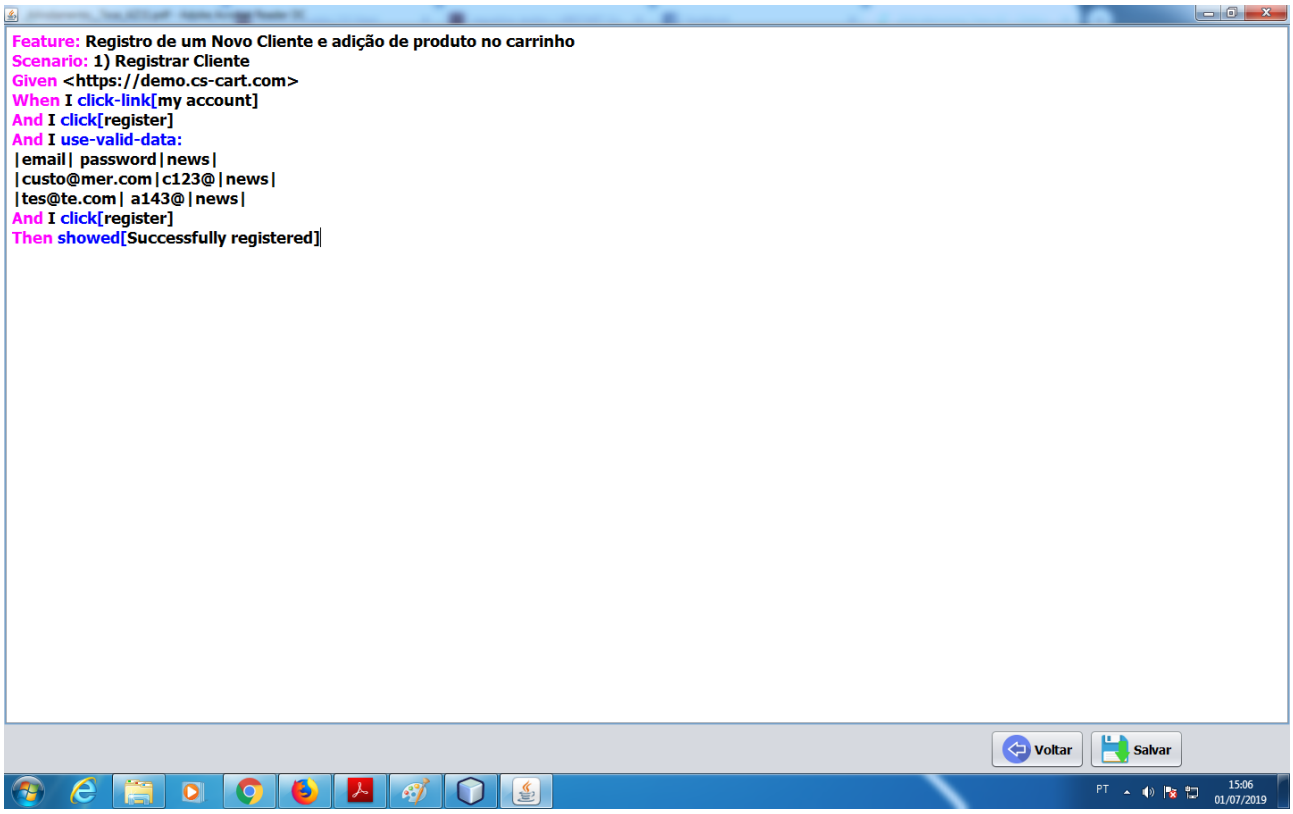

Figura 4.5 – Editor De Texto Aquila *Tool*

Independente da ferramenta escolhida, a escrita dos cenários precisa respeitar a regra de que cenários formados pelo mesmo modelo devem ser escritos no mesmo arquivo de *feature*. Isso acontece porque entende-se que um modelo representa uma funcionalidade do sistema, seu fluxo principal e alternativo, outras funcionalidades precisam ser apresentados e outros modelos. A seguir, apresenta-se as regras para a criação de cenários:

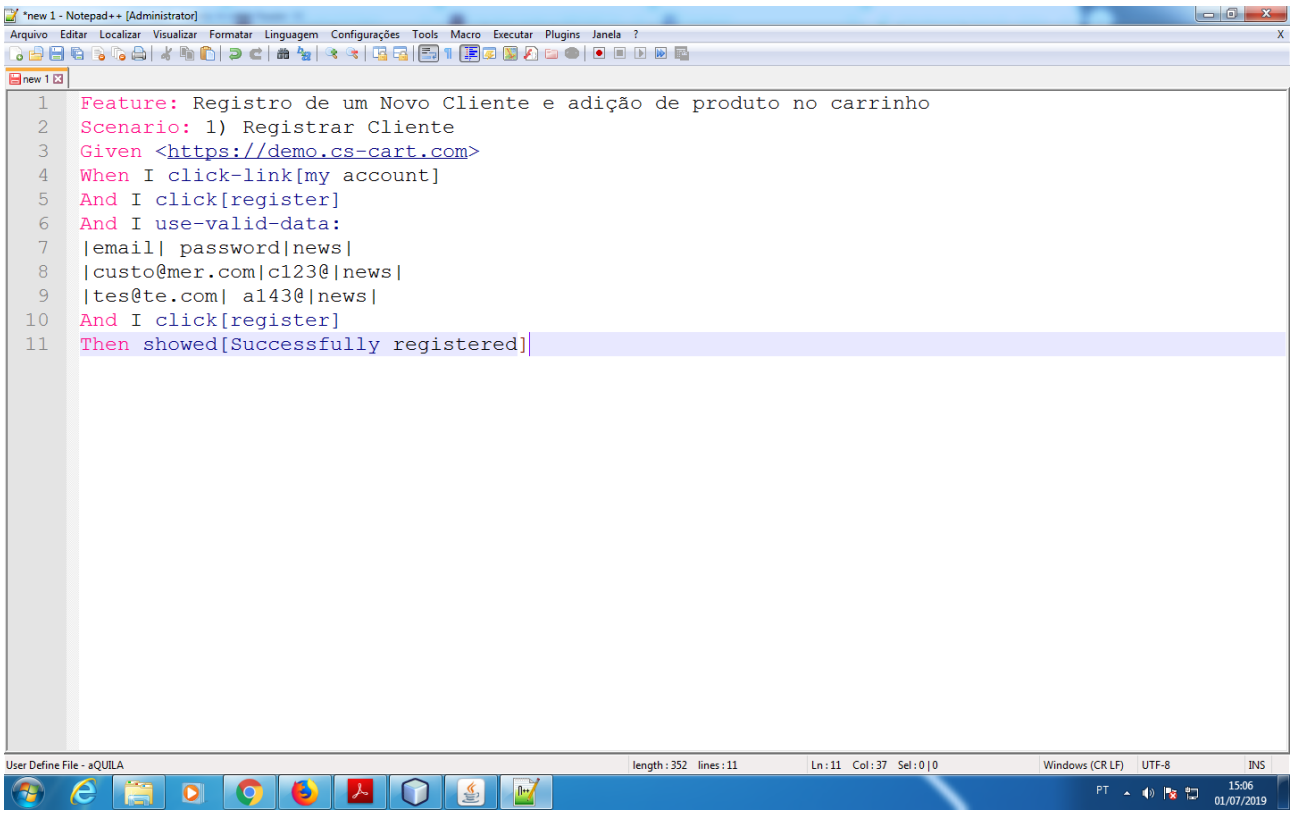

Figura 4.6 – *Plugin* Aquila Para Notepad++

1. Todo arquivo inicia com a palavra Feature: e o nome da feature que está sendo representada. A partir disso pode-se colocar uma descrição um não.

2. Cada cenário inicia com a palavra Scenário: e o nome do cenário. Um cenário pode conter as palavras-chave GIVEN, WHEN, THEN e AND ou conter apenas algumas delas, sendo que, a palavra chave GIVEN é obrigatória.

3. A palavra chave aquila deve ser a última palavra de cada linha

4. As linhas que contém tabela de entradas devem começar e terminar com pipe (|)

A seguir, são apresentados dois exemplos de cenários que utilizam a DSL Aquila: Registrar Cliente e Adicionar Produto ao carrinho. Estes são modelados com base no site https://demo.cs-cart.com

Feature: Registro de um Novo Cliente e adição de produto no carrinho

1. **Scenario**: 1) Registrar Cliente
2. **Given** <https://demo.cs-cart.com>
3. **When** I **click-link[**my account**]**
4. **And** I**click[**register**]**
5. **And** I**use-valid-data:**
6. | **email**| **password**| **news**|
7. | **custo@mer.com** | **c123@** | **news**|
8. | **tes@te.com** | **a143@** | **news**|
9. **And** I **click[**register**]**
10. **Then showed[**Successfully registered**]**

11.**Scenario**: 2) Adicionar Produto no Carrinho
12. **Given** { Register Custumer }
13.**Given I click[**search-performed**]**
14. **And Iput[**search-performed**]**
15.| **search-performed**|
16.| **bike**|
17. | **game**|
18. **And Iclick[**search**]**
19. **And Iclick-link[**Catania**]**
20. **And Iclick[**addToCard**]**
21. **Then showed[** The product was added to your cart**]**

22.**Scenario**: 3) Falha ao Registrar Cliente
23.**Given** <https://demo.cs-cart.com >
24.**When** I **click-link[**my account**]**
25. **And Iclick[**register**]**
26.**And Iput[**email**]**
27.| **email**|
28.| **a@b.com**|
29. | **c@d.com**|
30. **And Iput[**Password**]**
31.| **Password**|
32.| **abcd124**|
33. | **1234abcd**|
34. **And** I **dont-fill-out[**Confirm password**]**
35. **And Iput[**Birthday**]**
36.| **Birthday**|
37.| **10/10/1965**|
38. **Then showed[**The Confirm password field is mandatory**]**

2) Geração de Modelos. A etapa de geração de modelos é a etapa mais importante do processo, uma vez que os modelos são a base para geração dos *scripts* de testes por meio da combinação dos fluxos comportamentais por ele representados. É nesta etapa que está o grande diferencial deste trabalho. Produzir o modelo completo, detalhado suficientemente para aplicação de MBT é algo inovador proposto pela nossa abordagem. Um cenário pode produzir o modelo partindo de três contextos distintos conforme oque estiver especificado na linha da palavra chave Given:

1. O cenário iniciar um novo modelo: este caso acontece quando os valores informados junto com a palavra chave Given ainda não estão em nenhum modelo;
2. O cenário continuar um modelo existente a partir do seu final: acontece quando um cenário possui outro como pré requisito. Este caso acontece quando o nome de outro cenário é especificado entre chaves na linha da palavra chave Given;
3. Um cenário continuar um modelo já existente a partir do Given do modelo: isto acontece quando um cenário tem o mesmo início que outro cenário, contudo representa um fluxo diferente. Nesse caso o modelo existente é complementado com as informações de um novo cenário.

Em relação às demais palavras-chaves:

1. As palavras-chave “click”, “click-link”, “mouse-over”,“enable”,“disable”,“showed”,“dont-fill-out”, “opened” e “showed-title” são convertidas cada uma em uma vértice do modelo.

2. As palavras-chave “use-valid-data”, “put”, “dont-fill-out”, “checked”, “choose” e “ select-data” podem ser convertidas em uma ou mais vértices, isto porque, estas palavras-chaves requerem dados de entrada, e nesse caso, para cada entrada informada será criada uma vértice.

3. Embora a palavra “use-valid-data” seja utilizada para informar dados válidos para vários campos, ela segue a mesma lógica das demais palavras-chaves, contudo nesse caso uma entrada de dados (representada em uma vértice) é formada por um dado válido para cada campo especificado na tabela de entradas de “use-valid-data”.

Quando acontece a geração de mais de uma vértice para a mesma palavra chave automaticamente ocorre uma bifurcação no modelo. Além disso, bifurcação também pode acontecer, quando um cenário dá continuidade a um modelo já existente (continuando a partir do final ou a partir do Given, conforme descrito acima).Para os cenários representados na seção anterior o modelo gerado é representado na Figura 4.7.

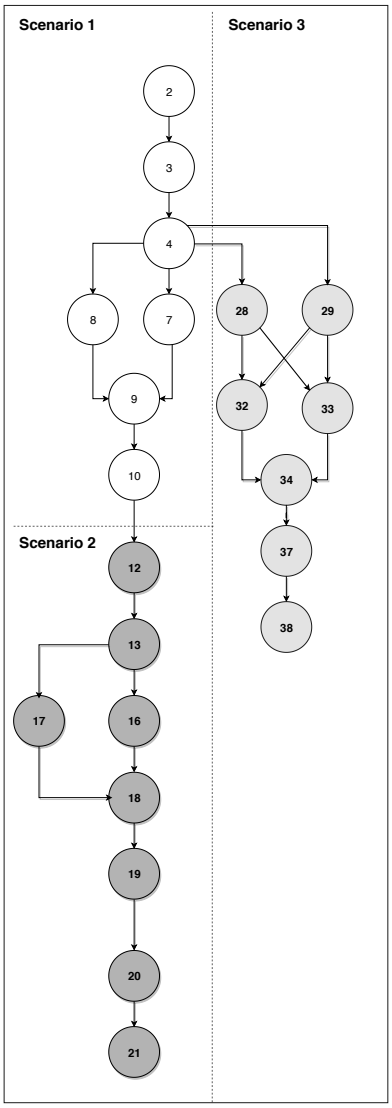

Figura 4.7 – Registro de um Novo Cliente (com e sem sucesso) e Adição de Produto no carrinho

Nesse modelo, é possível perceber a existência das bifurcações (fluxos alternativos) que representam a segunda opção de entrada para as palavras-chave use-valid-data e put especificadas na tabela de entrada((linhas 7,8,16, 17, 28 e 29) dos Cenários 1 e 2 na Figura 4.7). Além disso, nesse modelo é possível perceber que os três cenários especificados formaram um único modelo, isto porque o cenário 3 tem o mesmo Given que o cenário 1, e o Cenário 2, tem o cenário 1 como pre requisito. É importante destacar que o número das vértices do DAG representa o número da linha do cenário a qual vértice representa.

No momento da geração dos modelos, caso a Aquila *Tool* não entenda alguma das informações contidas nos cenários, ela irá gerar normalmente uma aresta no modelo para a linha que não foi compreendida. Contudo, o conteúdo desta linha será armazenado em uma lista, para que depois seja gerado um arquivo contendo métodos em branco para que o usuário possa inserir comandos personalizados para estes casos. Optamos por realizar a geração deste arquivo externo para que o usuário possa personalizar seu código para situações não cobertas pela DSL Aquila, e este código não ser perdido nas alterações do modelo, uma vez que, está armazenado em um local separado e apenas referenciado no modelo.

Para o cenário descrito a seguir, por exemplo, é gerado o arquivo auxiliar que consta na Figura 4.8

Feature: Arquivo Adicional

1. **Scenario**: Gerar Arquivo Auxiliar

2.**Given** <www.geraarquivo.com>

3. **When** I efetuar uma ação desconhecida pela Aquila

10.**Then:**Será gerado um arquivo adicional para que o usuário possa completar.

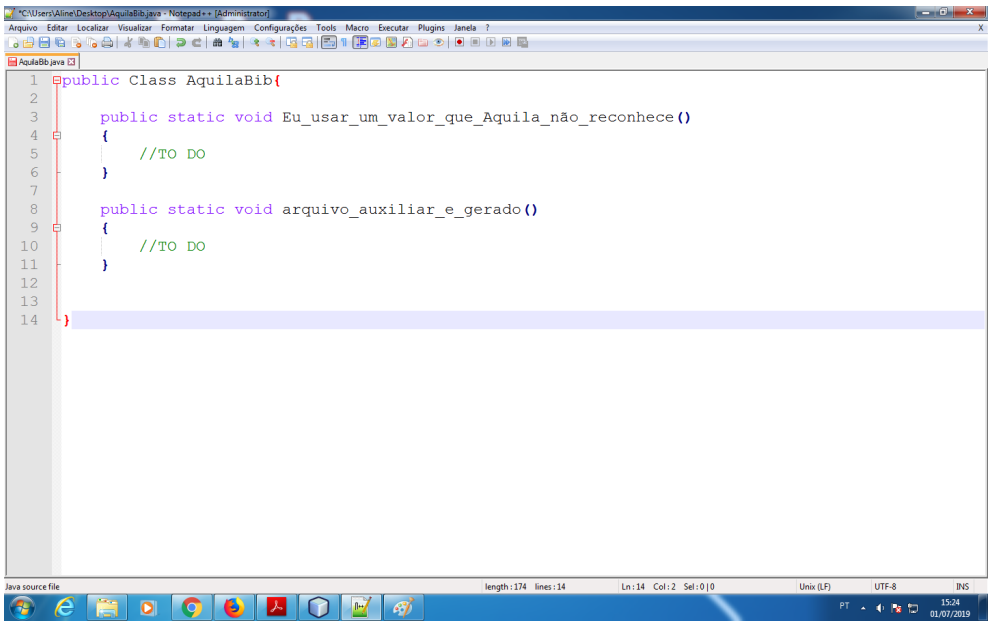

Figura 4.8 – Arquivo Auxiliar Gerado

A página inicial da ferramenta Aquila *Tool* é exibida na Figura 4.9 e a operacionalização para geração dos modelos, das sequências e dos *scripts* é a seguinte:

- No campo 1 selecione o arquivo que contém os cenários;
- No campo 2 informe onde serão salvos os *scripts* gerados e o nome do arquivo que conterá os *scripts*;
- No campo 3, selecione onde será salvo o arquivo adicional caso ele seja gerado.;
- Clique em “Generate”;

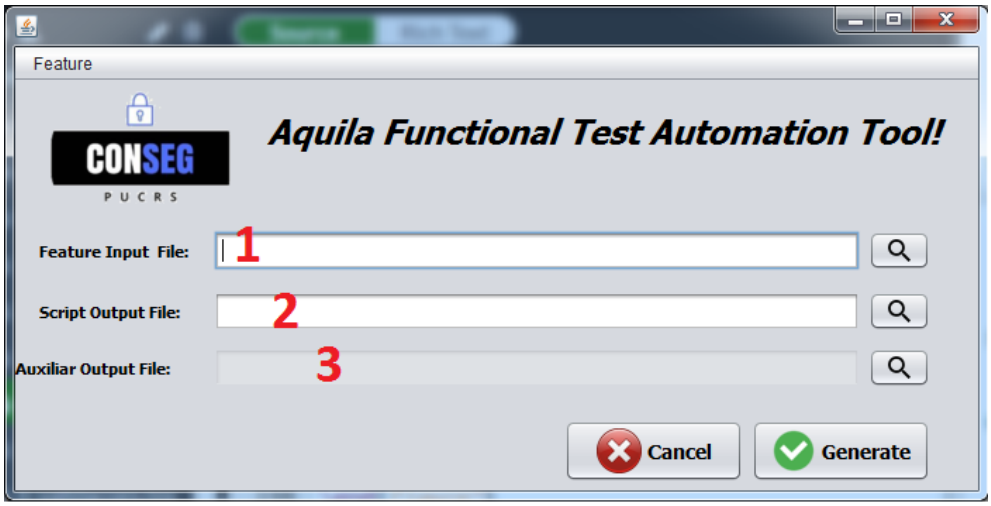

Figura 4.9 – Página Inicial Aquila

3) Geração de sequências de testes: Uma sequência de teste é gerada antes da geração de um *script* de teste. Ela determina quais funcionalidades que serão testadas em cada *script*.

Para a geração das sequências, utiliza-se o algoritmo de busca em profundidade *Depth-First Search algorithm* (DFS) [50]. Este algoritmo percorre um grafo, do ponto inicial ao ponto final, passando por todas as vértices e arestas. Com isso, o algoritmo gera os diversos caminhos que podem ser usados para percorrer esta rota. Cada um destes caminhos forma uma sequência de testes.

Sempre que existirem bifurcações no modelo, além de gerar uma sequência para cada bifurcação o algoritmo irá combinar os fluxos alternativos criados pelas bifurcações. Por exemplo: dado um campo B, com entradas 1 e 2 e um campo C com entradas 3 e 4, ambos originados de um campo A e com destino um campo D seriam geradas as seguintes sequências:

A - B1 - C3 - D
A - B2 - C3 - D
A - B1 - C4 - D
A - B2 - C4 - D

Para facilitar a representação das sequências, elas sequências são formadas pelos números que representam as linhas de um cenário e seus respectivos nós no DAG.

O Modelo representado na Figura 4.7 representa três cenários distintos, considerando que, estes cenários podem ser escritos separados, se gerarmos o modelo apenas do cenário 1, e efetuarmos a geração das sequências obteríamos as seguintes sequências:

1) 2-3-4-7-9-10
2) 2-3-4-8-9-10

Ao completar o modelo do cenário 1, com as informações do cenário 2, se geramos novamente as sequências vamos as seguintes:

1) 2-3-4-7-9-10-12-13-16-18-19-20-21
2) 2-3-4-7-9-10-12-13-17 -18-19-20-21
3) 2-3-4-8-9-10-12-13-16-18-19-20-21
4) 2-3-4-8-9 -10-12-13-17-18-19-20-21

Ao adicionar completar o modelo com as informações do cenário 3, se geramos novamente as sequências vamos as seguintes:

1) 2-3-4-7-9-10-12-13-16-18-19-20-21
2) 2-3-4-7-9-10-12-13-17 -18-19-20-21
3) 2-3-4-8-9-10-12-13-16-18-19-20-21
4) 2-3-4-8-9 -10-12-13-17-18-19-20-21
5) 2-3-4-28-33-34-37-38
6) 2-3-4-28-32-34-37-38
7) 2-3-4-29- 33-34-37-38
8) 2-3-4-29-32-34-37-38

Caso existissem novos cenários o processo de repetiria gerando novas sequências conforme novos fluxos são adicionados.

4) Geração de *scripts* de teste. O *script* de teste é baseado nos modelos gerados anteriormente e nas sequências de teste. Para cada modelo, um ou mais *scripts* de teste podem ser gerados de acordo com o número de sequências de teste geradas.

Para cada sequência (que conforme supracitado representa um fluxo que pode ser percorrido no modelo) é gerado um novo *script*. Cada palavra chave Aquila, precisa ser relacionada a um comando reconhecido pela ferramenta de automação e linguagem de programação escolhida. Desta forma, ao traduzir as sequencias de testes para *script* cada etapa da sequência, que faz relação a uma palavra chave irá gerar um trecho de código.

A Figura 5.14 mostra um trecho de código gerado para a ferramenta Selenium Webdriver, uma ferramenta de automação de testes para aplicações Web. Além disso a Figura 5.5 apresenta

o código criado baseado no mesmo modelo porém para a ferramenta Appium, uma ferramenta amplamente utilizada para automação de testes para dispositivos móveis.

Escolhemos estas duas ferramentas para exemplificar a aplicação da DSL Aquila, contudo, a DSL Aquila pode ser utilizada para geração de *scripts* para qualquer ferramenta ou linguagem, sendo necessário para isso realizar ajustes para nova sintaxe na ferramenta Aquila *Tool*.

```
1     @Test
2     public void addProductToCart() {
3       Webdriver driver = new FirefoxDriver();
4       driver.get("https://demo.cs-cart.com");
5       WebElement linkText = driver.findElement(By.linkText("my account"))
          ;
6       linkText.click();
7       WebElement register = driver.findElement(By.name("register"));
8       register.click();
9       WebElement email = driver.findElement(By.name("email"));
10      email.sendKeys("cuso@mer.com");
11      WebElement password = driver.findElement(By.name("password"));
12      password.sendKeys("c123@.com");
13      WebElement register2 = driver.findElement(By.name("register"));
14      register2.click();
15     (...)
16    }
```

Figura 4.10 – *Script* Gerado Para o Cenário “Add Product To Card” - Seleniun Webdriver

5) Melhorias nos *scripts* de testes gerados. Essas melhorias podem ser necessárias para corrigir algum erro de especificação dos cenários ou para efetuar a automação de testes de uma ação que não é totalmente coberta pela Aquila. Nesses casos, um *script* parcial é gerado e o profissional de testes precisará concluir o *script* gerado. O *script* parcial é gerado em uma biblioteca e pode ser editado a qualquer momento. Isso permite que o engenheiro de teste personalize seus testes para situações peculiares não cobertas pelas palavras-chave da Aquila.

6) Execução dos *scripts* de testes. Para executar os *scripts* de teste, é necessário usar uma ferramenta de automação de testes tradicional, por exemplo: Seleniun Webdriver (https://www.seleniumhq.org/projects/webdriver/) ou Appium (http://appium.io/) e o ambiente de desenvolvimento configurado para tal. A execução deve ser feita com a mesma ferramenta para qual foram gerados os *scripts*, então, se gerados *scripts* para Selenium os testes deverão ser executados com Selenium.

```
1       @Test
2       public void addProductToCart() {
3       AndroidDriver driver;
4       DesiredCapabilities desiredCapabilities = new DesiredCapabilities();
5       desiredCapabilities.setCapability("platformName", "Android");
6       desiredCapabilities.setCapability("deviceName", "Google Pixel 2");
7       desiredCapabilities.setCapability("automationName", "uiautomator2");
8       desiredCapabilities.setCapability("app", "app-release.apk");
9       URL remoteUrl = new URL("http://localhost:4723/wd/hub");
10      driver = new AndroidDriver(remoteUrl, desiredCapabilities);
11      MobileElement linkText = driver.findElement(By.linkText("my account")
           );
12        linkText.click();
13        MobileElement register = driver.findElementByName("register"));
14        register.click();
15        MobileElement email = driver.findElementByName("email"));
16        email.sendKeys("cuso@mer.com");
17        MobileElement password = driver.ffindElementByName("password"));
18        password.sendKeys("c123@.com");
19        MobileElement register2 = driver.findElementsByPartialLinkText("
             register"));
20        register2.click();
21       (...)
22      }
```

Figura 4.11 – Script gerado para o cenário “Add Product to card”- Appium

4.2 Sobre o Desenvolvimento Da Ferramenta Aquila *Tool*

A Aquila *Tool*[2] foi desenvolvida fazendo uso da linguagem de programação Java. Optou-se pela utilização de Java por diversos motivos, entre eles: esta linguagem de programação é a que a pesquisadora responsável por este trabalho possui maior conhecimento técnico e desta forma, poderia dar um suporte maior para o desenvolvimento. Além disso, como a Aquila utiliza a DSL Gherkin como base, utilizou-se como base o código da ferramenta Cucumber[3] e esta ferramenta está implementada em Java.

Outro fator considerado foi a facilidade de manutenção em código java devido a utilização de orientação a objetos, oque permitiu deixar a ferramenta facilmente adaptável para geração de *scripts* de testes em outras linguagens de programação ou para outras ferramentas de testes diferentes da utilizada neste trabalho (Seleniun WebDriver). Tecnicamente, a alteração da ferramenta

[2]A ferramenta Aquila *tool* foi desenvolvida dentro do grupo de pesquisas no qual o autor deste trabalho é parte. O desenvolvimento da ferramenta contou com o apoio do programador Henry C. Nune, um programador que atuava como estagiário do grupo de pesquisa.

[3]esse código está disponível publicamente no github e seu contrato de uso permite copia, edição e distribuição inclusive para fins comerciais(https://github.com/cucumber/cucumber/blob/master/LICENSE)

Aquila *Tool* para geração de *scripts* para outra ferramenta de testes ou linguagem de programação exige unicamente a criação de uma nova classe.

A ferramenta atualmente concentra a lógica da conversão de comandos Aquila em modelos dentro de um pacote Java denominado comandos. Neste pacote, existem classes separadas para cada comando da DSL Aquila. Estas classes fazem a conversão de cada palavra chave Aquila para modelo. Além disso, existe no mesmo pacote uma classe denominada Selenium.java, que possui a sintaxe Selenium e Java que é utilizada nos *scripts* gerados, sendo que, para outras linguagens ou ferramentas deve ser alteradas apenas esta classe.

Durante o desenvolvimento de Aquila *Tool* optou-se por não utilizar ferramenta prontas para de construção de DSL. Isto porque, o Cucumber já provia suporte para a identificação das palavras chaves básicas da linguagem Gherkin e através disso foi facilitado o desenvolvimento do suporte as palavras chave Aquila. Além disso, optou-se pela criação de um software independente para a Aquila *Tool*, que dispensasse a utilização conjunta de uma IDE de desenvolvimento, por exemplo Eclipse, uma vez que, em java é possível construir a interface gráfica e o editor de texto para escrita dos cenários de forma fácil e adaptável para o contexto desejado.

4.3 Considerações sobre o capítulo

Nesse capítulo apresentou-se a abordagem para aplicação de MBT em AT. Esta abordagem foi testada por meio de alguns exemplos didáticos, os quais forneceram embasamento para a sequência desta pesquisa. Os primeiros testes efetuados demonstraram a possibilidade de geração de sequências de testes a partir de cenários.

5. EXEMPLO DE USO

Para a exemplificação da utilização da DSL Aquila utilizamos um sistema de rastreamento de encomendas, em suas versões online e mobile. O sistema escolhido possui, em sua versão mobile, mais de 500 mil downloads na loja de aplicativos da Google. esse sistema permite, através de código de rastreio, monitorar o status do frete de uma encomenda bem como, emite notificações de qualquer alteração neste status. A aplicação web pode ser acessada através da URL https://www.muambator.com.br/pacotes/pendentes/ e os downloads para dispositivos móveis podem ser feitos em https://www.muambator.com.br/apps. Nas próximas seções, explica-se como foi utilizada a Aquila para gerar os *scripts* de testes para testar as principais funcionalidades do sistema Muambeitor.

5.1 Escrita dos Cenários

No processo de escrita de cenários foram identificados os cenários que cobrem as principais funcionalidades desta aplicação. Cada uma delas foi escrita em um arquivo .feature separado por mapear fluxos diferentes do sistema.

As funcionalidades mapeadas são descritas a seguir e a Tabela 5.1 apresenta o número de cenários identificados para cada uma dessas features:

- Cadastro: compreende os principais cenários que podem ser percorridos pelo usuário ao realizar o cadastro de um novo usuário no sistema.
- Login: compreende os cenários de autenticação de um usuário já cadastrado.
- Pacotes: Os cenários de inserção e busca de pacotes mapeiam os fluxos de cadastro de um novo pacote para ser monitorado pelo Muambeitor, que faz a gestão e o monitoramento dos pacotes já cadastrados.
- Assinatura: compreende o mapeamento do fluxo seguido pelo usuário que deseja comprar uma versão paga do Muambeitor para usufruir de funcionalidades adicionais.
- Oráculo: a funcionalidade de oráculo valida o oráculo de CEPS e entregas. Esse oráculo apresenta dados sobre o tempo médio de entrega de encomendas de um CEP de origem para um CEP de destino, dentro e fora do Brasil.

Tabela 5.1 – Mapeamento dos Cenários Por *Feature*

Feature	**Número de Cenários**	**Nome dos Cenários**	**Feature**	**Número de Cenários**	**Nome dos Cenários**
Assinatura	3	Realizar Assinatura	Login	4	Efetuar Login
		Botão Finalizar inativo			Efetuar Login - Senha Invalida
Pacotes	15	Buscar Pacote - Usuário Logado			Efetuar Login - ausente senha
		Novo Pacote - Usuário Logado			Efetuar Login - ausente usuário
		Novo Pacote - Sem Código	Cadastro	7	Criar nova conta
		Novo Pacote - Código Inexistente			Criar conta sem informar nome
		Novo Pacote - Código Inválido			Criar conta com e-mail invalido
		Marcar pacote como entregue			Criar conta sem informar e-mail
		Confirmar marcar como entregue			Criar conta sem informar senha
		Arquivar pacote entregue			Criar conta com senha e confirmação diferentes
		Confirmar arquivar pacote entregue			Link Termo de Uso
		Excluir pacote entregue	Oráculos	4	Acesso oráculo Gringo
		Confirmar exclusão de pacote			Busca oráculo Gringo
		Desarquivar pacote			
		Confirmar desarquivar pacote			Acesso oráculo tupiniquim
		Excluir pacote arquivado			
		Confirmar exclusão de pacote			Busca em Oráculo tupiniquim

Nessa seção, por ter objetivo de exemplificar a utilização da DSL Aquila, selecionamos alguns destes cenários para demonstrar a especificação deles utilizando a DSL e também para demonstrar a geração dos modelos e *scripts*. Os seguintes cenários foram selecionados: Cenário: Criar nova conta; Cenário: Link Termo de Uso; Cenário: Acesso oráculo Gringo; Cenário: Busca em oráculo Gringo; Cenário: Acesso oráculo Tupiniquin; Cenário: Busca em oráculo Tupiniquin; Cenário: Realizar Assinatura Cenário: Botão Finalizar Inativo;

Esses cenários foram escolhidos porque por meio do uso deles é possível visualizar a utilização de todas as TAGS da DSL Aquila, o que os torna suficientemente representativos. Os demais cenários descritos na Tabela 5.1 podem ser vistos no Apêndice F.

Feature: 1) Cadastro

Scenario: Criar nova conta

Given<https://www.muambator.com.br/>

When click[crie sua conta]

And use-valid-data

Usuario	**email**	**nome**	**senha**	**confirme**
abcd	**testemu@gmail.com**	**teste**	**123456**	**123456**
fged	**abcde@gmail.com**	**teste**	**789456**	**789456**

And click[criarConta]

Then opened[https://www.muambator.com.br/perfil/registre-se/confirmacao/]

And showed[cadastro completo]

Scenario: Link Termo de Uso

Given<https://www.muambator.com.br/>

When click[crie sua conta]

And click-link[Termos de Uso]

Then opened[https://www.muambator.com.br/termos-de-uso/]

And showed-title[Termos de Uso e Privacidade - Muambator]

Neste cenário, mesmo na versão mobile a palavra-chave *opened* é utilizada junto de uma URL, porque é um caso onde o aplicativo direciona para uma página web.

Feature: 2) Oráculos

Scenario: Acesso oráculo Gringo

Given{ efetuarlogin}

When I**mouse-over**[oraculo]

And click[oraculoGringo]

Them showed-title[Oráculo Gringo]

And opened[https://www.muambator.com.br/oraculo/gringo]

Scenario: Busca em oráculo Gringo

Given {Acesso oráculo Gringo}

When I**select-data**[pais]

| **pais**|
| **italia**|

And put[cepDestino**]**

| **cepDestino**|

| **90050230**|

Then showed[Media**]**

And showed[34,50**]**

Scenario: Acesso oráculo TUpiniquin

Given{ efetuarlogin}

When I **mouse-over[**oraculo**]**

And click[oraculo tupiniquim**]**

Them showed-title[Oráculo tupiniquim**]**

And opened[https://www.muambator.com.br/oraculo/tupiniquim/**]**

Scenario: Busca em oráculo Tupiniquin

Given {Acesso oráculo Tupiniquin**}**

When I**put[**cepOrigen**]**

| **cepOrigen**|

| **90050230**|

| **90619900**|

And put[cepDestino**]**

| **cepDestino**|

| **99150000**|

Then showed[Marau/BR**]**

And showed[Porto Alegre/BR**]**

Esta *feature* de oráculos não é utilizada na versão mobile do muambator.

Feature: 3) Assinatura

Scenario: Realizar Assinatura

Given<https://www.muambator.com.br/assinar/**>**

When I**choose[**plano**]**

| **plano**|

| **Muambeitor Pro Mensal**|

And I**use-valid-data**

| cep| numero| rua| bairro| cidade| estado

| 90050230| 310| xyzk| centro| Curitiba| Parana|

Then enable[finalizar]

And showed[assinado com sucesso]

Scenario: Botão Finalizar Inativo

Given<https://www.muambator.com.br/assinar/>

When Iuse-valid-data

| nome| dataNascimento| cpf| cep| numero| rua| bairro| cidade| estado|

| 90050230| 310| xyzk| centro| Curitiba| Parana|

Then disabled[finalizar]

5.2 Geração dos Modelos

Nesta seção, apresenta-se os modelos criados para os cenários especificados na Seção 5.1. Os modelos criados para os demais cenários descritos na Tabela 5.1 e no Apêndice F podem ser vistos no Apêndice G

- *Feature* Cadastro (Figura 5.1): neste modelo é possível perceber a presença de dois cenários, o cenário “Criar nova conta” e o cenário “Link Termos de Uso”. Neste modelo existe uma bifurcação entre os dois nodos que contém use-valid-data e o primeiro nodo do cenário “Link Termos de Uso”. Existem dois novos diferentes com use-valid-data porque neste cenário foram especificadas duas entradas para cada campo especificado em use-valid-data e sendo assim, são gerados dois fluxos no modelo. O cenário “Link Termos de Uso” inicia a partir do nodo “When click[crie sua conta]” porque esse nodo e o nodo “Given<https://www.muambator.com.br/> são comuns para os dois cenários”. Neste modelo é possível visualizar as seguintes palavras chave Aquila: use-valid-data, click, opened, showed e showed-title.

- *Feature* Assinatura (Figura 5.2): Neste modelo são apresentados os fluxos dos cenários “Realizar Assinatura” e “Botão Finalizar inativo”. Neste modelo o único fluxo alternativo está entre na divisão dos dois cenários. O cenário “Botão Finalizar inativo” inicia logo após o nodo que contém a palavra-chave Given, dado que esse é o nodo que é comum entre os dois cenários. Neste modelo, pode-se demonstrar a utilização das TAGS Aquila: choose, use-valid-data, enable, showed, disable.

- *Feature* Oráculos (Figura 5.2): Neste Modelo são apresentados os fluxos dos cenários: “Acesso oráculo gringo”, “Busca em oráculo gringo”,“Acesso oráculo Tupiniquim” e “Busca em oráculo Tupiniquim”. Além disso, neste modelo também é possível visualizar o modelo gerado para

o cenário “efetuar login”. esse cenário, embora não seja parte da *feature* oráculos é um pré-requisito para alguns cenários desta *feature* e por isso o modelo inicia trazendo os nodos referentes ao cenário efetuar “efetuar login”. Neste modelo é possível visualizar uma bifurcação nos nodos que contém “When put[cep_origem]” isto porque foram especificadas neste cenários duas possibilidades de entradas para esse campo. Neste modelo é possível visualizar as palavras chave Aquila: use-valid-data, click, opened, showed-title, put e mouse-over.

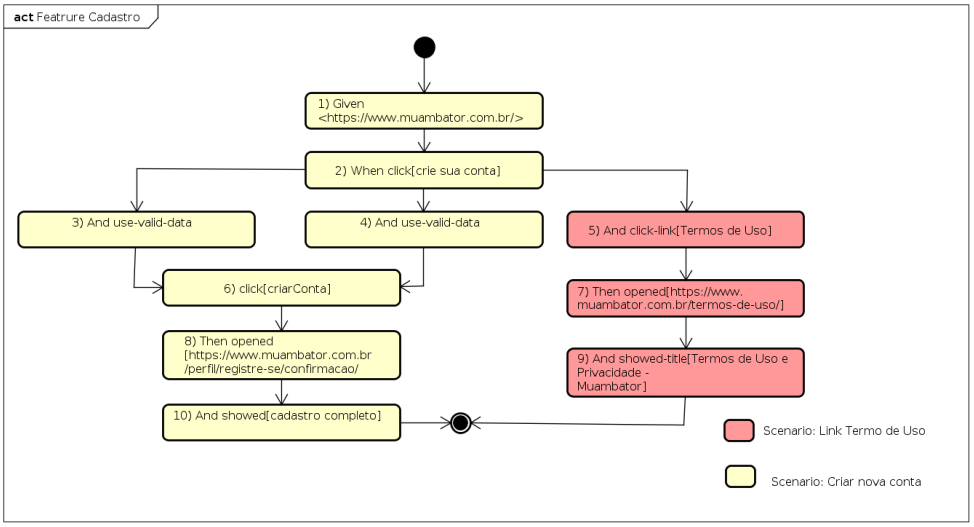

Figura 5.1 – *Modelo Feature* Cadastro

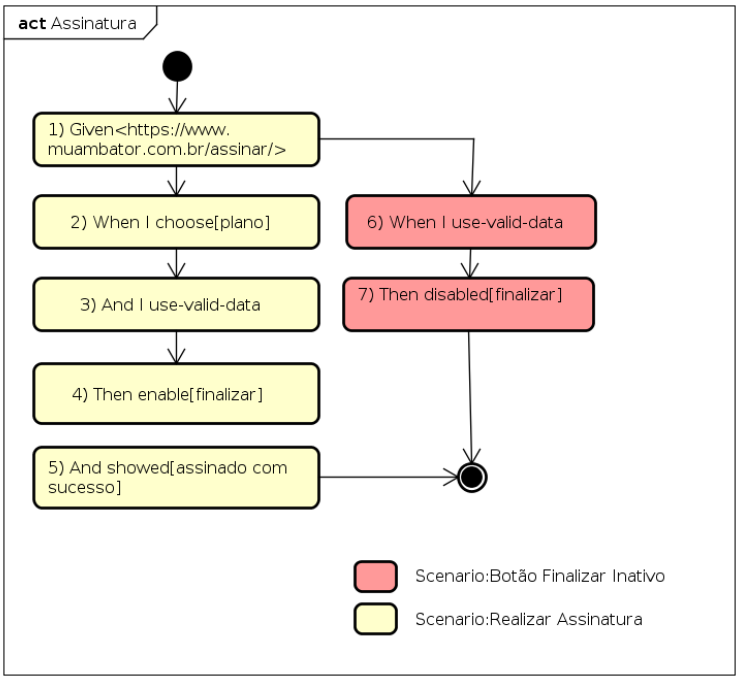

Figura 5.2 – *Modelo Feature* Assinatura

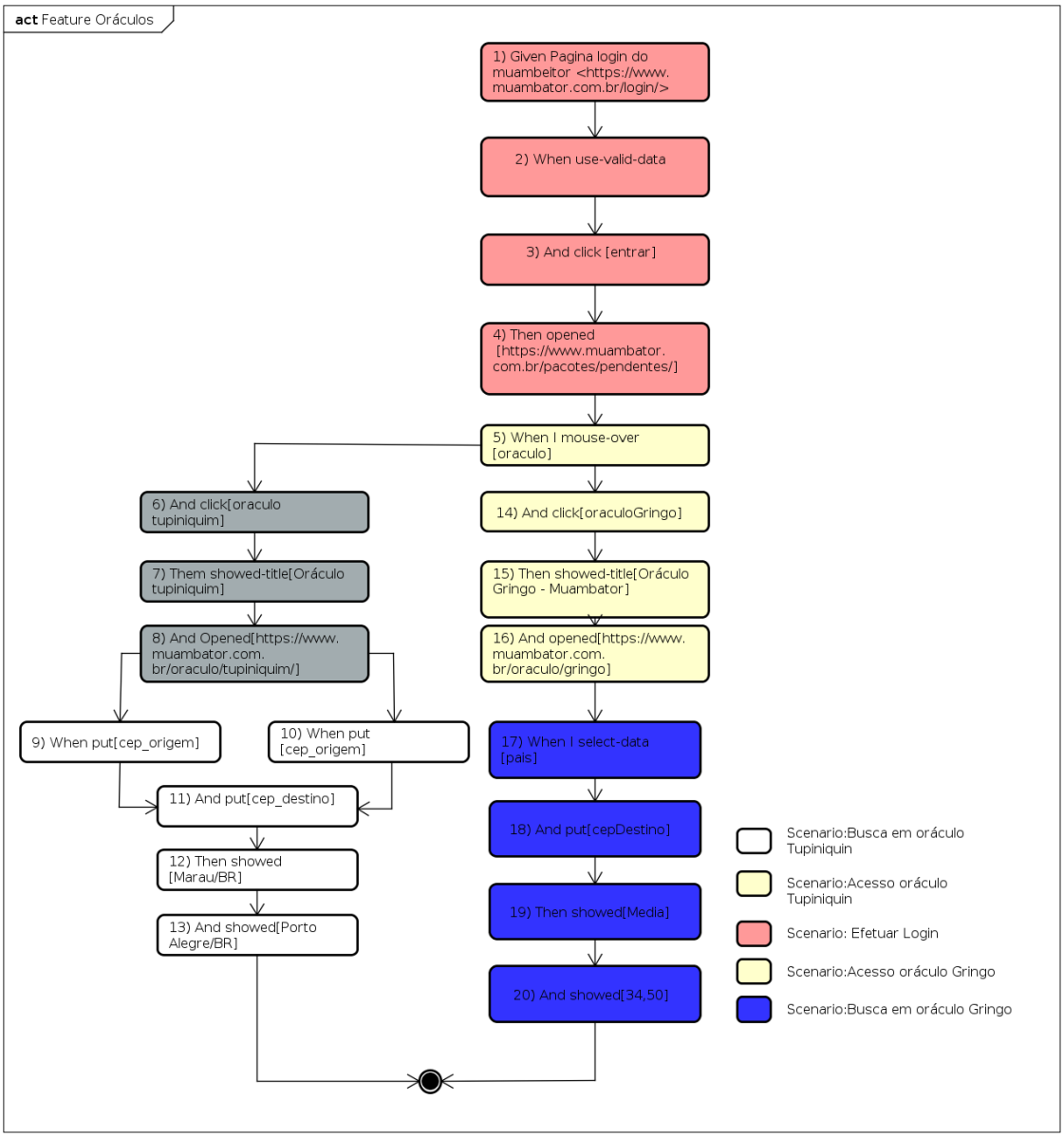

Figura 5.3 – *Modelo Feature* Oráculos

5.3 Geração das Sequências de Teste

Conforme supracitado, uma sequência de testes descreve um fluxo que pode ser seguido em um modelo. As sequências aqui utilizadas são formadas por números que representam os números que constam nos nodos dos modelos. As sequências identificadas para os modelos apresentados na seção 5.2 são as seguintes:

1. *Feature* Cadastro
   (a) 1-2-3-6-8-10
   (b) 1-2-4-6-8-10
   (c) 1-2-5-7-9

2. *Feature* Oráculos:

   (a) 1-2-3-4-5-6-7-8-9-11-12-13

   (b) 1-2-3-4-5-6-7-8-10-11-12-13

   (c) 1-2-3-4-5-14-15-16-17-18-19-20

3. *Feature* Assinatura:

   (a) 1-2-3-4-5

   (b) 1-6-7

5.4 Geração dos *Scripts* de Teste - Webdriver

Nesta seção, são apresentados os *scripts* gerados para as funcionalidades modeladas na Seção 5.2. Esses *scripts* utilizam linguagem java e a sintaxe da ferramenta Seleniun Webdriver, cada script gerado representa uma sequência de testes especificada na Seção 5.3.

```
1   @Test
2   public void featureCadastro_a() {
3     Webdriver wd = new FirefoxDriver();
4     wd.get("https://www.muambator.com.br/");
5     WebElement crieSuaConta = wd.findElement(By.name("crieSuaConta"));
6     crieSuaConta.click();
7     WebElement usuario = wd.findElement(By.name("usuario"));
8     usuario.sendKeys("abcd");
9     WebElement email = wd.findElement(By.name("email"));
10    email.sendKeys("testemu@gmail.com");
11    WebElement nome = wd.findElement(By.name("nome"));
12    nome.sendKeys("teste");
13    WebElement senha = wd.findElement(By.name("senha"));
14    senha.sendKeys("123456");
15    WebElement confirme = wd.findElement(By.name("confirme"));
16    confirme.sendKeys("123456");
17    WebElement criarConta = wd.findElement(By.name("criarConta"));
18    criarConta.click();
19    assertEquals("https://www.muambator.com.br/perfil/registre-se/confirmacao/
          ", wd.getCurrentUrl());
20    assertEquals(true, wd.getPageSource().contains(``Cadastro Completo''));
21  }
```

Figura 5.4 – *Script Feature* Cadastro (a) - Webdriver

A Figura 5.4 apresenta o código gerado para a primeira sequência de testes da feature cadastro. Neste modelo pode ser visualizado o código gerado para as tag: Given, click, use-valid-data, opened and showed.

```
1   @Test
2   public void featureCadastro_b() {
3     wd driver = new FirefoxDriver();
4     driver.get("https://www.muambator.com.br/");
5     WebElement crieSuaConta = driver.findElement(By.name("crieSuaConta"));
6     crieSuaConta.click();
7     WebElement usuario = wd.findElement(By.name("usuario"));
8     usuario.sendKeys("fged");
9     WebElement email = wd.findElement(By.name("email"));
10    email.sendKeys("abcde@gmail.com");
11    WebElement nome = wd.findElement(By.name("nome"));
12    nome.sendKeys("teste");
13    WebElement senha = wd.findElement(By.name("senha"));
14    senha.sendKeys("789456");
15    WebElement confirme = wd.findElement(By.name("confirme"));
16    confirme.sendKeys("789456");
17    WebElement criarConta = wd.findElement(By.name("criarConta"));
18    criarConta.click();
19    assertEquals("https://www.muambator.com.br/perfil/registre-se/confirmacao/
          ", wd.getCurrentUrl());
20    assertEquals(true, Driver.getPageSource().contains(``Cadastro Completo''))
          ;
21  }
```

Figura 5.5 – *Script Feature* Cadastro (b) - webdriver

A Figura 5.5 apresenta o código gerado para a segunda sequência de testes da *feature* cadastro. Esse script é muito similar ao anterior, sendo um fluxo alternativo do modelo cadastro. A Figura 5.6 apresenta o código gerado para a segunda sequência de testes da *feature* cadastro. Esse *script* tem por característica adicional, em relação aos anteriores, o fato de ter sido gerado para um fluxo do modelo que contempla a palavra-chave click-link

```
@Test
public void featureCadastro_c() {
  Webdriver wd = new FirefoxDriver();
  wd.get("https://www.muambator.com.br/");
  WebElement crieSuaConta = wd.findElement(By.name("crieSuaConta"));
  crieSuaConta.click();
  WebElement click_link = wd.findElement(By.partialLinkText("Termos"));
  click_link.click();
  assertEquals("https://www.muambator.com.br/termos-de-uso/", wd.
      getCurrentUrl());
  String title= wd.gettitle();
  assertEquals(title, ''Termos de Uso e Privacidade - Muambator'');
}
```

Figura 5.6 – *Script Feature* Cadastro (c) - Webdriver

```
@Test
public void featureOraculos_a() {
  Webdriver driver = new FirefoxDriver();
  driver.get("https://www.muambator.com.br/login/");
  WebElement usuario = driver.findElement(By.name("usuario"));
  usuario.sendKeys("aa@aa.com");
  WebElement senha = driver.findElement(By.name("senha"));
  usuario.sendKeys("12345abc");
  WebElement entrar = driver.findElement(By.name("entrar"));
  entrar.click();
  assertEquals("https://www.muambator.com.br/pacotes/pendentes/", driver.
      getCurrentUrl());
  field = driver.findElement(By.linkText("oraculo"));
  action = new Actions(driver);
  action.moveToElement(element).build().perform();
  WebElement oraculoTupiniquin = driver.findElement(By.partialLinkText("
      oraculoTupiniquin"));
  oraculoTupiniquin.click();
  String title= driver.gettitle();
  assertEquals(title, ''Oraculo Tupiniquim'');
  assertEquals(''https://www.muambator.com.br/oraculo/tupiniquim/'',,driver.
      getCurrentUrl());
  WebElement cepOrigem = driver.findElement(By.name("cepOrigem"));
  senha.sendKeys("90050230");
  WebElement cepDestino = driver.findElement(By.name("cepDestino"));
  senha.sendKeys("99150000");
  assertEquals(true, Driver.getPageSource().contains(''Marau/BR''));
 assertEquals(true, Driver.getPageSource().contains(''Porto Alegre/BR''));
}
```

Figura 5.7 – *Script Feature* Oráculos (a) - Webdriver

A Figura 5.7 apresenta o código gerado para a primeira sequência de testes da *feature* Oráculos. Esse *script* contempla as tags Given, use-valid-data, click, opened, mouse-over, showed-title.

```
1   @Test
2   public void featureOraculos_b() {
3     Webdriver driver = new FirefoxDriver();
4     driver.get("https://www.muambator.com.br/login/");
5     WebElement usuario = driver.findElement(By.name("usuario"));
6     usuario.sendKeys("aa@aa.com");
7     WebElement senha = driver.findElement(By.name("senha"));
8     usuario.sendKeys("12345abc");
9     WebElement entrar = driver.findElement(By.name("entrar"));
10    entrar.click();
11    assertEquals("https://www.muambator.com.br/pacotes/pendentes/", driver.
          getCurrentUrl());
12    field = driver.findElement(By.linkText("oraculo"));
13    action = new Actions(driver);
14    action.moveToElement(element).build().perform();
15    WebElement oraculoTupiniquin = driver.findElement(By.partialLinkText("
          oraculoTupiniquin"));
16    oraculoTupiniquin.click();
17    String title= driver.gettitle();
18    assertEquals(title, ``Oraculo Tupiniquim'');
19    assertEquals(``https://www.muambator.com.br/oraculo/tupiniquim/'',,driver.
          getCurrentUrl());
20    WebElement cepOrigem = driver.findElement(By.name("cepOrigem"));
21    senha.sendKeys("90050230");
22    WebElement cepDestino = driver.findElement(By.name("cepDestino"));
23    senha.sendKeys("99150000");
24    assertEquals(true, Driver.getPageSource().contains(``Marau/BR''));
25   assertEquals(true, Driver.getPageSource().contains(``Porto Alegre/BR''));
26  }
```

Figura 5.8 – *Script Feature* Oráculos (b) - Webdriver

A Figura 5.8 apresenta o código gerado para a segunda sequência de testes da *feature* Oráculos. Esse *script* contempla as tags Given, use-valid-data, click, opened, mouse-over, showed-title, opened e put.

```
1       @Test
2       public void featureOraculos_c() {
3         Webdriver driver = new FirefoxDriver();
4         driver.get("https://www.muambator.com.br/login/");
5         WebElement usuario = driver.findElement(By.name("usuario"));
6         usuario.sendKeys("aa@aa.com");
7         WebElement senha = driver.findElement(By.name("senha"));
8         usuario.sendKeys("12345abc");
9         WebElement entrar = driver.findElement(By.name("entrar"));
10        entrar.click();
11        assertEquals("https://www.muambator.com.br/pacotes/pendentes/", driver.
              getCurrentUrl());
12        field = driver.findElement(By.linkText("oraculo"));
13        action = new Actions(driver);
14        action.moveToElement(element).build().perform();
15        WebElement oraculoGringo = driver.findElement(By.partialLinkText("
              oraculoGringo"));
16        oraculoTupiniquin.click();
17        String title= driver.gettitle();
18        assertEquals(title, ''Oraculo Gringo - Muambator'');
19        assertEquals(''https://www.muambator.com.br/oraculo/gringo/'',,driver.
              getCurrentUrl());
20        Select dropdown = new Select(driver.findElement(By.name("pais")));
21        dropdown.selectByVisibleText("Italia");
22        WebElement cepDestino = driver.findElement(By.name("cepDestino"));
23        senha.sendKeys("90050230");
24        assertEquals(true, Driver.getPageSource().contains(''Media''));
25        assertEquals(true, Driver.getPageSource().contains(''34,50''));
26      }
```

Figura 5.9 – *Script Feature* Oráculos (c) - Webdriver

A Figura 5.9 apresenta o código gerado para a terceira sequência de testes da *feature* Oráculos. Esse *script* contempla as tags Given, use-valid-data, click, opened, mouse-over, showed-title, opened, select-data, put e showed.

```
1   @Test
2   public void featureAssinatura_a() {
3     Webdriver w = new FirefoxDriver();
4     wd.get("https://www.muambator.com.br/planos/");
5     WebElement plano = wd.findElement(By.id("MuambeitorProMensal"));
6     plano.click();
7     WebElement cep = w.findElement(By.name("cep")).sendKeys("90050230");
8     WebElement numero = w.findElement(By.name("numero")).sendKeys("310");
9     WebElement rua = w.findElement(By.name("rua")).sendKeys("xyzk");
10    WebElement bairro = w.findElement(By.name("bairro")).sendKeys("centro");
11    WebElement cidade = w.findElement(By.name("cidade")).sendKeys("Curitiba");
12    WebElement estado = w.findElement(By.name("estado")).sendKeys("Parana");
13    WebElement finalizar = w.findElement(By.name("finalizar"));
14    boolean enable = finalizar.isEnabled();
15    assertEquals(true, enable);
16    assertEquals(true, w.getPageSource().contains(``assinado com sucesso''));
17  }
```

Figura 5.10 – *Script Feature* Assinatura (a) - Webdriver

A Figura 5.10 apresenta o código gerado para a primeira sequência de testes da *feature* Assinatura. Esse *script* contempla as tags: Given, choose, use-valid-data e enable.

```
1   @Test
2   public void featureAssinatura_b() {
3     Webdriver w = new FirefoxDriver();
4     w.get("https://www.muambator.com.br/planos/");
5     WebElement cep = wd.findElement(By.name("cep")).sendKeys("90050230");
6     WebElement numero = w.findElement(By.name("numero")).sendKeys("310");
7     WebElement rua = w.findElement(By.name("rua")).sendKeys("xyzk");
8     WebElement bairro = w.findElement(By.name("bairro")).sendKeys("centro");
9     WbElement cidade = w.findElement(By.name("cidade")).sendKeys("Curitiba");
10    WebElement estado = w.findElement(By.name("estado")).sendKeys("Parana");
11    WebElement finalizar = w.findElement(By.name("firstname"));
12    boolean enable = finalizar.isEnabled();
13    assertEquals(false, enable);
14  }
```

Figura 5.11 – *Script Feature* Assinatura (b) - Webdriver

A Figura 5.11 apresenta o código gerado para a primeira sequência de testes da *feature* Assinatura. Esse *script* contempla as tags: Given, use-valid-data e disable.

5.5 Geração dos *Scripts* de Teste - Appium

As *features* “Oráculo” e “Assinar” não estão disponíveis na versão mobile do aplicativo, dessa forma, são apresentados aqui os *scripts* utilizados para a *feature* cadastro. É importante enfatizar que os cenários poderiam ser representados pela DSL Aquila da mesma forma que na versão web, contudo o aplicativo mobile tem suas funcionalidades reduzidas e, por isso, não apresenta oráculo e assine.

```
1   @Test
2   public void featureCadastro_a_appium() {
3       AndroidDriver ad;
4       DesiredCapabilities desiredCapabilities = new DesiredCapabilities();
5       desiredCapabilities.setCapability("platformName", "Android");
6       desiredCapabilities.setCapability("deviceName", "Google Pixel 2");
7       desiredCapabilities.setCapability("automationName", "uiautomator2");
8       desiredCapabilities.setCapability("app", "muambator.apk");
9       URL remoteUrl = new URL("http://localhost:4723/wd/hub");
10      ad = new AndroidDriver(remoteUrl, desiredCapabilities);
11      WebElement crieSuaConta = ad.findElementByName("crieSuaConta")).click();
12      WebElement usuario = ad.findElementByName("usuario")).sendKeys("abcd");
13      WebElement email = ad.findElementByName("email")).sendKeys("
            testemu@gmail.com");
14      WebElement senha = ad.findElementByName("senha")).sendKeys("123456");
15      WebElement criarConta = ad.findElementByName("criarConta")).click();
16      assertEquals("ListPackageActivity", ad.getCurrentUrl());
17      assertEquals(true, Driver.getPageSource().contains("Nenhum Pacote Por
            Aqui ate o momento"));
18  }
```

Figura 5.12 – *Script Feature* Cadastro (a) - Appium

```
1   @Test
2   public void featureCadastro_b_appium() {
3    AndroidDriver ad;
4     DesiredCapabilities desiredCapabilities = new  DesiredCapabilities();
5     desiredCapabilities.setCapability("platformName", "Android");
6     desiredCapabilities.setCapability("deviceName", "Google Pixel 2");
7     desiredCapabilities.setCapability("automationName", "uiautomator2");
8     desiredCapabilities.setCapability("app", "muambator.apk");
9     URL remoteUrl = new URL("http://localhost:4723/wd/hub");
10    ad = new AndroidDriver(remoteUrl,desiredCapabilities);
11    WebElement crieSuaConta = ad.findElementByName("crieSuaConta"));
12    crieSuaConta.click();
13    WebElement usuario = ad.findElementByName("usuario")).sendKeys("fged");
14    WebElement email = ad.findElementByName("email")).sendKeys("abcde@gmail.
          com");
15    WebElement senha = ad.findElementByName("senha")).sendKeys("789456");
16    WebElement criarConta = ad.findElementByName("criarConta")).click();
17    assertEquals("ListPackageActivity", ad.getCurrentUrl());
18    assertEquals(true, Driver.getPageSource().contains(''Nenhum Pacote Por
          Aqui ate o momento''));
19  }
```

Figura 5.13 – *Script Feature* Cadastro (b) - Appium

```
1   @Test
2   public void featureCadastro_c_appium() {
3     Webdriver driver = new FirefoxDriver();
4     driver.get("https://www.muambator.com.br/");
5     WebElement crieSuaConta = driver.findElementByName("crieSuaConta"));
6     crieSuaConta.click();
7     WebElement click_link = driver.findElementByName("Termos de Uso"));
8     click_link.click();
9     assertEquals("https://www.muambator.com.br/termos-de-uso/", driver.
          getCurrentUrl());
10    String title= driver.gettitle();
11    assertEquals(title, ''Termos de Uso e Privacidade - Muambator'');
12  }
```

Figura 5.14 – *Script Feature* Cadastro (c) - Appium

6. VALIDAÇÕES DA ABORDAGEM PARA APLICAÇÃO DE MBT EM AT

Nesta Seção são apresentados os trabalhos de validação realizados para averiguar a adequação da abordagem proposta com o cotidiano de equipes ágeis. Esta validação se deu em dois momentos: primeiramente realizou-se um estudo de grupo focal apresentando a DSL Aquila e a abordagem para aplicação de MBT em AT para um grupo de profissionais e coletando as suas perspectivas. Depois, oportunizamos a alguns profissionais que utilizassem a DSL Aquila em um ambiente controlado e através de questionário coletamos as suas perspectivas após a utilização.

Nas próximas seções descrevemos as metodologias utilizadas, os resultados obtidos e também as ameaças a validade de cada um dos estudos.

6.1 Estudo 1: Grupo Focal (*Focus Group*)

Para avaliar a aplicabilidade de Aquila, utilizamos um método de pesquisa que traria a percepção de especialistas na área. Um Grupo Focal é organizado de forma sistemática e dividido em etapas: planejamento, execução e análise de resultados.

6.1.1 Planejamento do Estudo de Grupo Focal

O primeiro passo para garantir que as sessões do Grupo Focal, possam ser conduzidas de forma adequada, é definir os objetivos do estudo. No caso dessa tese, o objetivo geral, como mencionado anteriormente, é entender a percepção dos especialistas sobre a aplicabilidade de da abordagem para aplicação de MBT em AT proposta nesse estudo e da DSL Aquila.

Para isso, primeiro buscamos entender a percepção dos especialistas sobre a produtividade que a utilização de MBT em AT, baseado na DSL Aquila, pode prover. Além disso, nos propusemos a entender qual a avaliação dos especialistas quando a produtividade proporcionada pela DSL Aquila é comparada com a produtividade de um processo de testes que considera a programação manual dos *scripts* para automação de testes.

Posteriormente questionamos os especialistas a respeito da suas percepções deles sobre a curva de aprendizado do uso da Aquila em comparação com a curso de aprendizagem para a criação manual de *scripts* de teste. Assim, estabelecemos duas questões de pesquisa (RQ) que queríamos responder até o final do estudo de Grupo Focal:

RQ1: Qual é a percepção dos especialistas sobre a produtividade que pode ser obtida ao usar a DSL Aquila em comparação com a criação manual de *scripts* de teste para automação de testes?

RQ2: Quais são as percepções dos especialistas em relação à curva de aprendizado sobre o uso

de Aquila quando comparadas à curva de aprendizado para a criação manual de *scripts* de teste para automação de testes?

O segundo passo na fase de planejamento foi determinar o perfil e a quantidade de especialistas que participariam da execução do Grupo Focal. Nesse estudo, foram convidados profissionais de TI que trabalham em equipes ágeis de desenvolvimento de software, cujas principais responsabilidades eram análise de negócios, programação ou análise de qualidade de software. Foi realizada uma chamada pública descrevendo o estudo e o perfil requerido para participação, convidando profissionais para colaborarem com o estudo. Esse convite foi feito via redes sociais e os profissionais que demonstraram interesse foram convidados para participar.

Uma vez determinados o perfil e o número de sujeitos, definiu-se a data e o local em que a sessão de grupo focal seria executada. Foi escolhido um local próximo ao trabalho dos sujeitos e também estabelecida uma janela de tempo para que os participantes escolhessem a data mais adequada.Considerando que a maioria dos sujeitos que aceitou participar do estudo trabalha no parque tecnológico da PUC (Tecnopuc), optou-se por realizar o encontro dentro da Universidade.

Para garantir o sucesso da sessão, preparou-se uma sala para que nenhum fator externo interferisse, de modo que fosse possível coletar todos os dados dos participantes. Por exemplo, escolheu-se uma sala de reunião que não tivesse interferência de ruído, com equipamentos de gravação de áudio e vídeo previamente testados. Ademais, todos os documentos que seriam usados durante as sessões foram revisados e contabilizados.

Além disso, para verificar se o protocolo do Grupo Focal estava correto, realizamos um estudo piloto com três profissionais de TI com experiência em programação e teste de software. Os resultados desse estudo piloto não foram incluídos na análise final. Basicamente, esse piloto foi utilizado para verificar se a gravação de áudio e vídeo estava capturando de maneira correta e precisa a voz e as imagens dos participantes; se os questionários estavam coletando as informações necessárias para o estudo; se os documentos que seriam fornecidos aos sujeitos continham informações suficientes sobre Aquila, para que eles pudessem fornecer uma boa avaliação; e se a duração estimada da sessão (tempo) seria suficiente. O estudo piloto nos ajudou a melhorar os seguintes aspectos: os documentos que descrevem a Aquila e seu funcionamento; as questões utilizadas nos questionários; a posição do equipamento de gravação.

Para conduzir a sessão de grupo focal, tivemos a participação um moderador e dois assistentes para garantir que os dados importantes obtidos durante as sessões não seriam perdidos. A prática de utilizar moderador e assistente está de acordo com o processo descrito por [33].

Para execução do estudo de grupo focal, a sessão foi dividida em duas fases. Na fase 1, a Aquila foi apresentada e os participantes receberam um questionário para responder questões relacionadas às questões de pesquisa desse estudo, de acordo com suas percepções individuais. Na fase 2 foi realizada uma discussão coletiva onde todos os participantes apresentaram suas opiniões. Durante esse período, o moderador não interviu na discussão exceto em situações onde se fez necessário assegurar que a discussão permanecesse dentro do escopo estabelecido. Os assistentes

durante toda sessão anotaram pontos importantes das discussões, controlaram o tempo, monitorar equipamento de gravação e distribuíram os documentos aos sujeitos sempre que necessário.

Os seguintes documentos foram fornecidos a todos os participantes: Questionário de Identificação do Perfil do Participante, Formulário de Consentimento da Participação na Pesquisa (Apêndice B), Roteiro do Grupo Focal (Apêndice A) e Questionário sobre a DSL Aquila (Apêndice C);

As seguintes perguntas foram feitas aos sujeitos participantes para compreender os seus perfis:

1) Qual é a sua área de atuação (teste ou qualidade de software; análise de negócios; gerenciamento de projetos; desenvolvedor; outros)?;

2) Você tem conhecimento sobre automação de testes? Avalie seu conhecimento;

3) Qual é o seu nível de experiência como um programador de equipe ágil?;

4) Qual o seu nível de experiência em relação à área de negócio (requisitos, cenários, regras de negócio ...)?;

5) Qual o seu nível de experiência em relação ao gerenciamento de equipes ágeis?

6) Qual é o seu nível de conhecimento sobre o BDD ou a criação de cenários usando o Gherkin?

Para todas as questões, as alternativas foram as seguintes: Iniciante (já teve contato, era capaz de trabalhar com isso, mas não o fez); Médio (menos de dois anos trabalhando nisso); Pleno (mais de 2 e menos de 5 anos trabalhando nisso); Sênior (mais de 5 anos trabalhando nisso); ou não tem conhecimento disso.

Além disso, foram feitas as seguintes perguntas sobre a DSL Aquila:

Q1) Você acredita que o uso do Aquila pode melhorar a produtividade da equipe? Se sim, como?

Q2) Comparado com a criação manual de *scripts* para automação de testes, você consideraria que o Aquila é mais produtivo, menos produtivo ou indiferente? Explique.

Q3) Qual a sua percepção sobre a curva de aprendizado do uso de Aquila em relação à curva de aprendizado para a criação manual de *scripts* para automação de testes ?

Q4) Você aplicaria Aquila no projeto de teste que está trabalhando com desenvolvimento de software ágil? Por quê?

Q5) Caso você aplique Aquila a um projeto, quem você acha que deveria ser responsável por escrever os cenários?

6.1.2 Execução do Estudo de Grupo Focal

Nesta seção, apresenta-se os detalhes sobre a execução do Grupo Focal. Essa execução foi baseada no planejamento supradescrito. Como mencionado anteriormente, 8 sujeitos participaram do Grupo Focal. O *background* destes sujeitos é: profissionais que trabalham em equipes ágeis de desenvolvimento de software e realizam atividades com ênfase em programação, negócios, gerenciamento ou testes. O perfil individual de cada assunto pode ser visto na Tabela 6.1

Tabela 6.1 – Perfil Individual dos Sujeitos Participantes do Estudo de Grupo Focal

S1	Profissional com mais de 2 anos de experiência em programação, conhecimento inicial em automação de testes, negócios e Gherkin e sem conhecimento de gestão.
S2	Profissional com até 2 anos de experiência em automação de testes e Gherkin e mais de 2 anos de experiência em programação, negócios e gerenciamento.
S3	Profissional que com até 2 anos de experiência em Gherkin, conhecimento inicial em automação de testes e negócios e sem conhecimento de programação e gerenciamento ágil de projetos.
S4	Profissional com até 2 anos de experiência em automação de testes, negócios, gerenciamento e Gherkin e sem conhecimento de programação.
S5	Profissional com mais de 5 anos de experiência em automação de testes e negócios; até 2 anos de experiência com Gherkin e conhecimento inicial em programação e gerenciamento ágil de projetos.
S6	Profissional com até 2 anos de experiência em automação de testes e programação, com mais de 2 anos de experiência em gestão e negócios e mais de 5 anos de experiência com Gherkin.
S7	Profissional com mais de 5 anos de experiência em programação, até 2 anos de experiência em automação de testes e conhecimento inicial em negócios, gestão e Gherkin.
S8	Profissional com até 2 anos de experiência com automação de testes Gherkin e mais de 5 anos de experiência em programação, negócios e gerenciamento.

Como mencionado, a Sessão foi dividida em duas fases. Na primeira, além da apresentação da DSL Aquila, do preenchimento do questionário de perfil e do termo de consentimento, os participantes responderam a perguntas individuais sobre a Aquila. Esta fase durou 40 minutos. Na segunda fase, a discussão das questões foi realizada coletivamente, por aproximadamente 30 minutos.

6.1.3 Análise de Resultados: Fase 1

Na primeira fase desse estudo, os resultados são baseados nas percepções individuais de cada sujeito. A seguir, são apresentadas as conclusões de cada pergunta feita (ver Seção 6.1.1).

Q1: Você acredita que o uso do Aquila pode melhorar a produtividade da equipe? Se sim, como? A maioria dos participantes respondeu que em sua opinião a Aquila melhora a produtividade de seus projetos. Entretanto, um dos participantes considerou que a Aquila poderia, em um momento inicial, reduzir a produtividade da equipe devido à curva de aprendizado para utilização das palavras chaves Aquila. O mesmo participante embasa esse comentário no fato de que se a equipe não estiver acostumada a usar o BDD ou o Gherkin, a produtividade da equipe tende a cair porque a equipe precisa mudar a estrutura de trabalho para usar o Aquila. Por outro lado, os outros 6 participantes consideraram que isso poderia ser o caso inicialmente, mas que haverá um ganho de produtividade mais tarde devido à redução do trabalho pela geração de *scripts* de teste automaticamente.

Com base nas respostas obtidas, e considerando que a Aquila foi pensada para um contexto ágil, onde os profissionais devem ser capazes de experimentar novas tecnologias que agreguem vantagens à equipe, bem como considerando que Aquila foi pensado com foco em equipes que já use Cucumber[10] ou BDD[47], podemos dizer que a DSL influenciará positivamente na produtividade da equipe.

Q2: Comparado com a criação manual de *scripts* para automação de testes, você consideraria que o Aquila é mais produtivo, menos produtivo ou indiferente? Explique. Nesta questão, todos os entrevistados consideram que utilizar a DSL Aquila pode ser mais produtivo do que criar *scripts* manualmente. Nesse momento, os participantes enfatizaram a importância da padronização da nomenclatura de campos nos sistemas para facilitar o uso do Aquila e, consequentemente, colaborar com o aumento da produtividade. Além disso, os participantes lembraram da importância de que a ferramenta que for criada para suportar a DSL Aquila busque minimizar a quantidade de alterações que o testador precisa fazer nos *scripts* gerados, bem como a qualidade do código gerado para facilitar manutenções futuras. Isso foi considerado pelos participantes como um fator determinante para a produtividade e para utilização da ferramenta.

Q3: Qual a sua percepção sobre a curva de aprendizado do uso de Aquila em relação à curva de aprendizado para a criação manual de *scripts* para automação de testes ?A maioria dos participantes participantes (6) consideraram que a curva de aprendizado para utilização da DSL Aquila é menor do que a curva de aprendizado para criar manualmente o *script* para automação de testes. No entanto, um dos sujeitos participantes acredita que não há diferença na curva de aprendizado e outro sujeito acredita que a curva de aprendizado é maior, pois a aprendizagem da Aquila não dispensa o conhecimento necessário de alguma ferramenta de automação de testes para entender os *scripts* gerados.

Q4: Você aplicaria Aquila no projeto de teste que está trabalhando com desenvolvimento de software ágil? Por quê? Para esta questão, todos os participantes responderam que usariam a DSL Aquila. No entanto, algumas condições foram apresentadas para seu uso: dois participantes responderam que usariam Aquila desde que a equipe já estivesse habituada a escrever cenários Gherkin; um participante apontou que em seu projeto Aquila seria usado primeiramente, em caráter experimental, como uma prova de conceito (POC) para depois ser usado em projetos reais. Não obstante, o sujeito acredita que a experiência como POC teria sucesso e o uso seria institucionalizado. Finalmente, um participante considerou que Aquila poderia ser utilizado, em seu projeto quando acontece a integração de novos membros na equipe. Um fator limitante para o uso de Aquila foi em relação ao número de palavras chave. O participante considerou que as palavras chaves propostas podem não ser suficientes para cobrir todos os requisitos de teste do sistema e, portanto, ainda precisaria fazer alguma programação de teste manual.

Com base nas respostas obtidas, entendemos que a Aquila pode ser aplicada em equipes ágeis com sucesso. Em relação ao número de palavras chave, acredita-se que o número a estrutura das palavras chaves abranjam a maioria das ações que podem ser realizadas pelo usuário em um sistema. Naturalmente, dependendo dos domínios em que se aplicarem os software para o qual

Aquila será utilizado para testar, diferentes palavras chaves podem ser necessárias. Nesse sentido é importante enfatizar que a Aquila considerará um processo para que o testador possa personalizar os *scripts* para estas situações em que a DSL não possui palavra chave específica. Além disso, uma vez provado que é possível gerar *scripts* utilizando MBT a partir de cenários, e considerando que a DSL será disponibilizada de forma *open source*[1] publicamente, os projetos possam adaptar a DSL para as suas necessidades.

Q5: Caso você aplique Aquila a um projeto, quem você acha que deveria ser responsável por escrever os cenários? Esta questão foi apresentada aos sujeitos em um formato de múltipla escolha. Desta forma eles podiam escolher entre as seguintes opções:

1. Área de Negócios/Representante do Cliente escreve o cenário no formato Gherkin e Testador adapta colocando as palavras chaves da Aquila.
2. Área de Negócios/Representante do Cliente escreve o cenário no formato Gherkin e Programador adapta colocando as palavras chaves da Aquila.
3. Área de Negócios/Representante do Cliente escreve os requisitos em outro formato e Programador/Testador escrevem os cenários colocando as palavras chaves da Aquila.
4. Área de Negócios/Representante do Cliente escreve os requisitos em outro formato, programador coloca os requisitos no padrão Gherkin e testador coloca no padrão Aquila.

A opção que obteve mais respostas foi “Aŕea de Negócios ou Clientes escreve o cenário no formato Gherkin e a Equipe de Desenvolvimento adapta cenários adicionando as palavras-chave de Aquila” (resposta obtida 4 sujeitos); seguido por: “Área de Negócios ou Representante do Cliente escreve os requisitos em outro formato (não Gherkin ou Aquila) em conjunto” e “equipe de desenvolvimento e testes escrevem os cenários adicionando as palavras-chave da Aquila (resposta obtida de 3 dos sujeitos)”.

Estas respostas nos levaram a acreditar que os sujeitos participantes consideram que a participação de profissionais de negócios ou representante do cliente no processo de escrever cenários é muito importante. No entanto, as opiniões dos participantes divergem em relação ao profissional de negócios juntamente com o cliente escrever os requisitos no formato Gherkin ou escrever requisitos em outro formato. Acreditamos que em ambos os cenários é possível utilizar o Aquila, e é importante adaptá-lo ao contexto do projeto para não causar mudanças abruptas na rotina dos profissionais.

6.1.4 Análise dos Resultados: Fase 2

A segunda fase desse estudo apresenta os resultados da discussão que aconteceu coletivamente com todos os participantes. As questões abordadas nesta parte do estudo são as mesmas

[1] Código aberto que pode ser alterado e utilizado

utilizadas na fase anterior. Isso facilitou a interação dos participantes na discussão, uma vez que, eles já os haviam analisado as questões durante a primeira fase formando a sua opinião individual para poder debater no grupo.

Similarmente à análise realizada na primeira fase, descrevemos a análise da discussão organizada por tópicos conforme as questões discutidas. Basicamente, analisamos os pontos em que os participantes convergiram ou divergiram em cada um dos pontos. Os resultados foram organizados em mapas mentais, que serão mostrados nas Figuras 6.2, 6.3, 6.4 e 6.5.

- Influência da Aquila na produtividade das equipes

Para este tópico tivemos cinco sujeitos que participaram ativamente da discussão, todos concordaram que Aquila influencia diretamente na produtividade da equipe. No entanto, houve alguma divergência entre influência positiva e negativa na produtividade.

Os participantes acreditam que a Aquila pode influenciar positivamente na produtividade das equipes, podendo ser utilizada como suporte para inserção de novos profissionais nos times e como ferramenta de aprendizagem de automação de testes. Entretanto, é comum entre os participantes a percepção de que, para que esta vantagem seja evidenciada, os profissionais, mesmo que iniciantes, precisam já possuir algum conhecimento sobre automação de testes ou BDD.

Isto porque os participantes consideraram que, em situações onde o profissional não tem conhecimento sobre automação de testes, haverá uma influência negativa na produtividade devido à necessidade do profissional aprender a usar o Aquila (empregar corretamente as palavras chaves e escrever cenários) e, adquirir um conhecimento inicial em alguma ferramenta de automação de testes por (exemplo: Seleniun Webdriver) para que possa compreender o código gerado pela DSL Aquila e dar manutenção neste.

Além disso, os participantes consideram que a Aquila pode interferir positivamente na produtividade da equipe quando os profissionais já estão acostumados a usar algum padrão de design. Nesse contexto, um participante (S3) sugeriu que, se o projeto não possui diretrizes de teste, o uso de Aquila pode influenciar positivamente a organização do projeto, o que também é um fator positivo.

Outro fator que foi considerado pelo grupo como um influenciador positivo para a produtividade, é o uso de palavras chaves que se assemelham a linguagem natural. Isto porque Aquila permite que pessoas que não possuem um grande conhecimento de linguagens de programação, por exemplo, analistas de negócios ou mesmo clientes, tenham facilidade de entender e, portanto, possibilidade de revisar os cenários para garantir que estes atendam aos requisitos, e inclusive possam editar as palavras chaves corrigindo erros de especificação de requisitos.

Em resumo, pode-se dizer que o grupo considera que a Aquila irá influenciar positivamente na produtividade da equipe, desde que algumas condições sejam respeitadas: o testador deve ter o mínimo de conhecimento em automação de testes; a pessoa de negócios precisa estar envolvida com a criação dos cenários e o projeto deve ser organizado para essa estrutura de utilização de

cenários e automação de testes. Estas conclusões estão resumidas na Figura 6.2 e reforçadas pelas seguintes afirmações:

“ Em termos de produtividade, que eu acho que é o foco da questão, eu acho que sim. Quem já tem alguma experiência costuma ter suas notas e "copiar e colar"para os novos cenários e com a Aquila em vez de editar o "copiar e colar", um testador pode fazer isso diretamente na especificação e gerar o código. Isso é melhor [S5]”.

“Tenho a sensação de que, se o testador é iniciante em automação de testes, é muito mais fácil [S4]”.

“Se levarmos isso para a área de negócios, para eles escreverem os cenários de teste e outra pessoa apenas os complementar, é muito mais viável para eles digitar nomes de palavras chaves do que escrever código Java, por exemplo [S2]”.

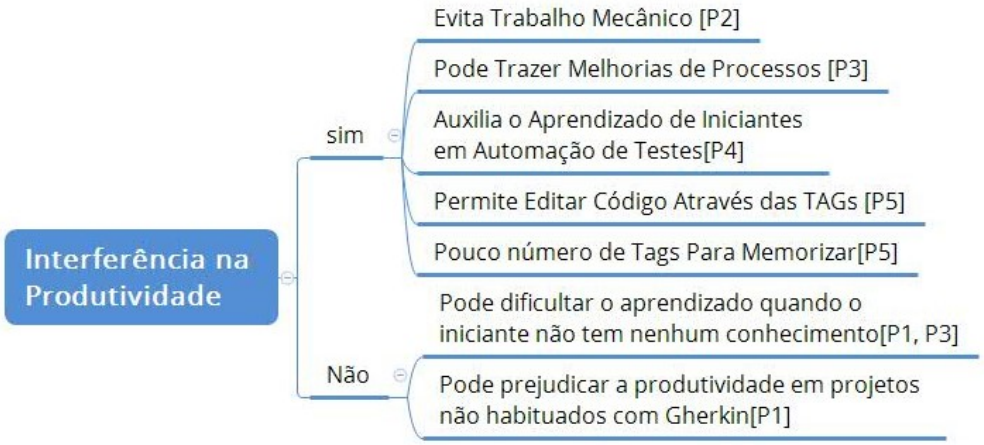

Figura 6.1 – Interferência da DSL Aquila Produtividade

- Produtividade da DSL Aquila Comparada à Criação Manual de *scripts* de Testes.

Novamente, nesse tópico, cinco participantes estiveram ativamente envolvidos na discussão, e houve um consenso de que a Aquila poderia prover uma melhor produtividade quando comparada com a criação manual de *scripts* para automação de testes. No entanto, alguns limites foram apontados pelos participantes que devem ser o foco dos pesquisadores no desenvolvimento de trabalhos futuros relacionados à Aquila, sendo eles: garantir que o código dos *scripts* gerado seja de fácil compreensão e manutenção; garantir que não seja necessário ao testador efetuar codificação manual ou que seja necessário apenas pequenas adaptações; buscar uma alternativa para Aquila localizar automaticamente o nome dos campos que estão presentes no sistema que está sendo testado para completar os *scripts* auxiliando o testador. Estas conclusões são reforçadas pelas seguintes afirmações:

“Comparado com a criação manual de *scripts* de teste, acredito que aumentaria a produtividade [S4]”.

“Poderia ser mais produtivo até certo ponto, dependendo do que teria que ser alterado manualmente [S7]”.

“Em trabalhos futuros poderia ser desenvolvido um plugin onde eu, tendo a Aquila e eu preciso testar um site específico, eu apenas rodo a Aquila, informo a URL e ele completa tudo pra mim [S8]”.

OBS: Nesta parte do estudo houve três intervenções do moderador: repetir a pergunta, responder se o participante poderia fazer uma pergunta e responder à pergunta do participante.

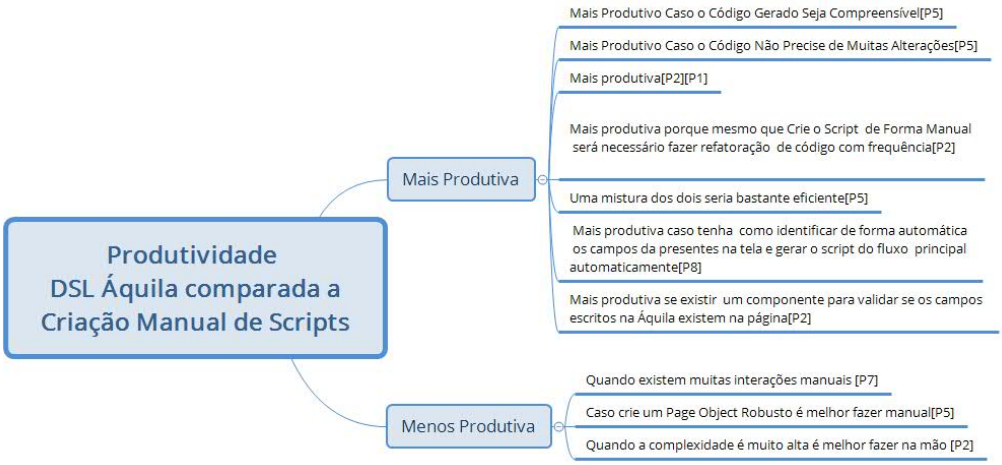

Figura 6.2 – Produtividade da DSL Aquila Comparada, Criação Manual de *Scripts*

- Curva de Aprendizagem da DSL Aquila em Relação à Curva de Aprendizagem Para Criação Manual de *Scripts* de Testes.

Em relação a curva de aprendizagem os participantes são unânimes em afirmar que a curva de aprendizagem da DSL Aquila é inferior a curva de aprendizagem para escrita manual de *scripts* de testes para automação. É importante enfatizar que, alguns participantes consideram que a curva de aprendizado da DSL Aquila só é pequena quando o profissional já tem conhecimento prévio de automação e, outros consideram que mesmo que o profissional não conheça automação, a curva de aprendizado da DSL Aquila segue sendo menor, e que a DSL Aquila pode, inclusive, auxiliar no aprendizado de outra ferramenta de automação. Foi citado também que o fato de utilizar linguagem seminatural é um fato positivo para a redução de curva de aprendizagem.

Em síntese, analisando as respostas supracitadas, conclui-se que a DSL Aquila pode facilmente ser aprendida e utilizada pelos profissionais. Assim como toda ferramenta inovadora a DSL Aquila exige uma curva de aprendizagem e um período de adaptação, mas as discussões indicam que a curva é menor do que a curva que habitualmente existe para o aprendizado por exemplo de

uma linguagem de programação e que a DSL Aquila pode ser facilmente utilizada em conjunto com outra ferramenta de automação e com isso auxiliar o aprendizado desta, uma vez que a DSL Aquila gera o código que representa uma ação, de modo que o programador iniciante consegue ver esse código, podendo analisá-lo, entendê-lo e memorizá-lo.

> “Facilita o aprendizado do Selenium inclusive [S2]”.

> “Na minha opinião o Selenium é mais complexo, essa ali acho eu que não tem muitos comandos como o Selenium mais fácil [S6]”.

> “Eu acho que é mais simples porque tá mais próximo de uma linguagem natural assim[S8]”.

> “Eu acho que a curva do Aquila é muito menor, mas claro a pessoa precisa saber escrever teste manualmente para poder editar [S5]”.

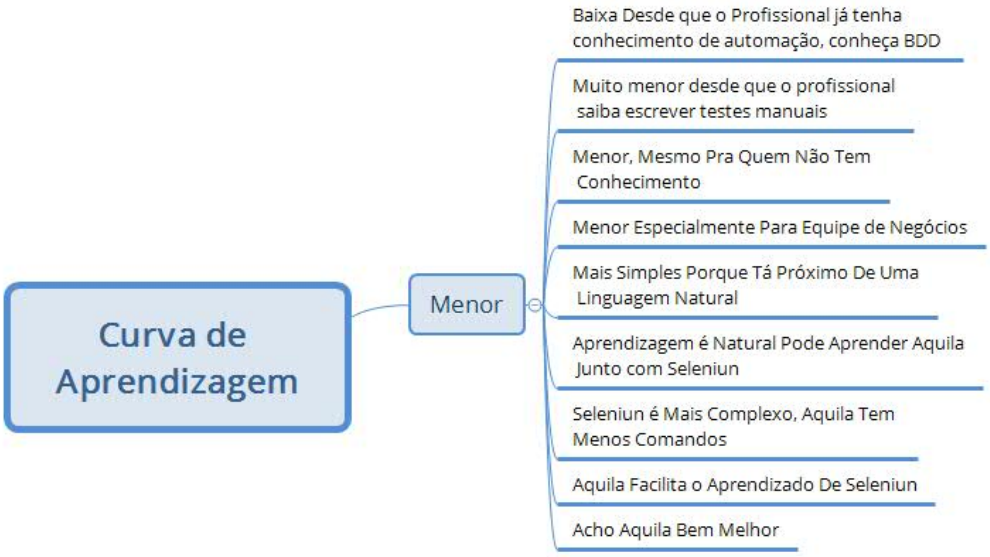

Figura 6.3 – Curva de Aprendizagem da DSL Aquila em Relação à Curva de Aprendizagem Para Criação Manual de *Scripts* de Testes

- Aplicabilidade da DSL Aquila

Nesta questão, diversas opiniões foram expressadas pelos participantes do estudo. Nenhum participante declarou que não utilizaria a DSL Aquila, contudo, foram expostos diversos pontos que seriam considerados pelos profissionais no momento de decidir entre utilizar ou não a DSL Aquila. O principal ponto, o qual foi citado por diversos participantes, é que aplicação da DSL Aquila é mais recomendada em projetos os quais já estão habituados a utilizar a técnica de BDD ou a escrita de requisitos do formato Gherkin. Outro fator considerado determinante na escolha de utilizar ou não a DSL Aquila é a quantidade de alterações que precisam ser feitas nos *scripts* gerados

pela Aquila, sendo que, esta quantidade de alterações necessária está altamente relacionada à complexidade do sistema e ao tipo de funcionalidades que o sistema possui.

Analisando as respostas supracitadas, pode-se dizer que a DSL Aquila é aplicável para o contexto a qual se propõe. Ela foi projetada pensando em equipes ágeis que utilizam Gherkin para escrita de requisitos, ou então equipes ágeis que estão buscando utilizar esta metodologia. Quanto a quantidade de alterações necessárias nos *scripts* a DSL Aquila está preparada para automatizar a maioria das ações que podem ser feitas em um sistema, então, considera-se que para a maioria dos casos ela poderá ser aplicada.

Além disso outro fator citado como um ponto de decisão para utilização da DSL Aquila é a disponibilidade da equipe. Entende-se que a flexibilidade a mudanças é uma característica desejada a profissionais que atuam em times ágeis, isto vem especialmente de encontro com o manifesto ágil o qual deve ser o norte das equipes que utilizam estas metodologias [5].

Figura 6.4 – Aplicabilidade da DSL Aquila

- Responsável Pela Escrita Dos Cenários

Em sua maioria os profissionais presentes na discussão e que participaram ativamente desta, consideram que deve existir um trabalho colaborativo no time todo para a escrita dos cenários no formato Aquila. É imune a percepção de que o PO (*product Owner*) ou o responsável pelo negócio deve pelo menos iniciar a escrita dos cenários, sendo que, um dos participantes acredita que o PO irá apenas escrever de forma informal e os demais que ele já deve entregar para equipe os requisitos no formato Gherkin, sendo um participante favorável a o PO já inserir as palavras chaves Aquila ao escrever os cenários.

Todos os participantes concordam que refinamentos devem ser feitos pela equipe de desenvolvimento, sendo que esta inclui desenvolvedores, testadores e demais membros do time. Estes refinamentos podem ser melhorias nos cenários e a inserção das palavras chaves Aquila. Uma outra situação possível é o trabalho colaborativo para escrita dos cenários, inserção de palavras chaves Aquila as atividades subsequentes do processo de desenvolvimento.

Analisando a percepção dos participantes, entende-se que deve existir uma colaboração entre todos os membros da equipe para o sucesso dos times que utilizam a DSL Aquila, sendo que, é altamente recomendável que os profissionais de negócio realizam a escrita inicial dos cenários e o restante do time efetue o detalhamento e melhorias.

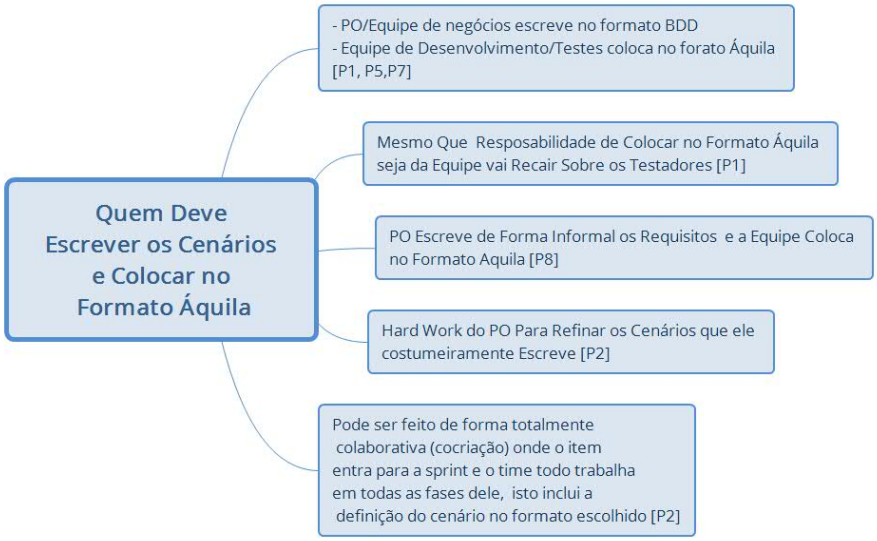

Figura 6.5 – Persona Responsável Pela Escrita Dos Cenários

6.1.5 Resultados

Analisando as respostas obtidas nos questionários individuais e na sessão de *Focus Group* pode-se chegar a algumas considerações sobre a DSL Aquila. Inicialmente, comparando as respostas pode ser observado que as percepções foram bastante similares quando os participantes responderam individualmente quando responderam na discussão do grupo. Contudo, na discussão do grupo foi dado maior ênfase em algumas questões que nos questionários individuais haviam sido superficialmente citadas. Por exemplo: na sessão de *Focus Group* ganhou bastante ênfase o fato de em alguns casos, os *scripts* gerados pela DSL Aquila precisaram ser adaptados para cobrir totalmente as funcionalidades em testes. Esse fato é considerado pelos participantes da sessão com um grande limitador e um ponto de atenção que precisa ser considerado.

Outro ponto levantado na sessão de *Focus Group* que não havia sido levantado nos questionários individuais é a sugestão de que a DSL Aquila localize os campos da tela de forma automatizada e gere sem intervenção humana o *script* do caminho principal.

Estas questões nos levam a crer que durante a discussão coletiva os participantes comparando os seus projetos com os dos demais conseguem ter uma visão mais ampla do cenário Ágil como um todo. Tanto nos questionários individuais quanto no *Focus Group* obteve-se respostas positivas para a DSL Aquila nos contextos as quais avaliamos.

Desta forma respondendo a primeira questão de pesquisa (RQ1) os especialistas consideram que a DSL Aquila pode aumentar a produtividade das equipes, desde que estas equipes estejam habituadas ou dispostas a criar o hábito de usar Gherkin ou BDD e que os *scripts* gerados exijam o mínimo possível de manutenção.

Já a segunda questão de pesquisa (RQ2) que diz respeito a curva de aprendizagem da utilização da DSL Aquila em relação a curva de aprendizagem para a criação manual de *scripts* para automação de testes os especialistas consideram a curva de aprendizagem para utilização da DSL Aquila menor, contudo, acreditam que profissionais que possuem algum conhecimento de automação de testes terão maior facilidade do que profissionais que estão iniciando na área de testes e não tem conhecimento nenhum.

6.1.6 Ameaças a Validade do Estudo

Analisando o protocolo de estudo de grupo focal definido para este estudo e descrito na Seção 6.1.1 pode ser identificado algumas ameaças a viabilidade desse estudo as quais são descritas nesta Seção

1. Número de Participantes: o número de participantes desse estudo é um número reduzido e pode não representar a totalidade dos profissionais que atuam em equipes ágeis.

2. Perfil dos Participantes: o perfil dos participantes desse estudo é bastante eclético (negócio, desenvolvimento, testes), se por um lado isto dá uma visão maior da totalidade do projeto pode deixar faltando representatividade em áreas específicas;

3. Contato dos Participantes com a DSL Aquila: o grupo focal foi conduzido sem que os participantes tivesse oportunidade de experimentar a utilização da DSL, sendo por esse motivo as opiniões dos profissionais baseadas em suas percepções e não em uma vivencia experimental;

4. Escolha dos participantes: a escolha dos participantes foi feita por conveniência, tendo sido divulgado via redes sociais o convite e realizado o estudo com os profissionais que se voluntariaram

Nesse sentido, considera-se que a amostra utilizada embora seja um número reduzido proporcionou uma discussão rica em informações e atendeu ao que a literatura sugere como número ideal de participantes para um estudo de grupo focal. Alternativamente, poderia ter sido realizado outras sessões para confirmar os dados, contudo considera-se que por terem participado da sessão profissionais com diversos perfis podem ser consideradas representativas as suas opiniões.

Embora a amostra tenha sido escolhida por conveniência considera-se que a variedade de perfis de participantes envolvidos nos deu uma visão ampla do processo ágil como um todo. Isto

nos garante que os resultados obtidos não possuem um viés tendencioso do olhar de uma área apenas, por exemplo: profissionais de testes que para desonerar seu trabalho sugerem a criação de requisitos pela equipe de negócios, contudo esta não possui tempo hábil. A representatividade de múltiplos perfis nos garantiu que a discussão teria visão de diversas áreas.

6.2 Estudo 2: *Survey*

Para esse segundo estudo de validação da aplicação de MBT em equipes ágeis, reuniu-se um grupo de profissionais que atuam com testes de software em equipes ágeis e lhes foi dada a oportunidade de interagir com a DSL Aquila para posteriormente avaliar a mesma.

A interação dos profissionais com a DSL se deu por meio de um cenário controlado onde foi apresentado a eles um protótipo de interface de um sistema e eles foram desafiados a construir os cenários e gerar os *scripts* para realização de testes na funcionalidade prototipada. Anteriormente, visando dar embasamento para que eles pudessem cumprir o desafio, a DSL Aquila e o funcionamento da ferramenta que suporta a DSL foram explicados aos participantes por meio de apresentação de slides. Nesse momento os participantes tiveram oportunidade de fazer questionamentos sobre a DSL e sobre o funcionamento da ferramenta.

O estudo contou com a participação de treze profissionais que trabalham com testes em equipes ágeis, cujo perfil é detalhado na Tabela 6.2. Os participantes classificaram seu conhecimento em iniciante, intermediário, avançado e não tenho conhecimento. Auto avaliação dos participantes seguiu o seguinte critério:

- Iniciante: Tenho conhecimento, posso realizar tarefas básicas;
- Intermediário: Tenho conhecimentos e posso realizar tarefas complexas;
- Avançado: Tenho conhecimento e posso realizar todo tipo de tarefa inclusive gestão;
- Não tenho conhecimento: profissional sem conhecimento do assunto.

A amostra foi escolhida por conveniência, sendo que, de participantes foi realizado um convite público via redes sociais e através da rede de contatos oferecida pela comunidade do Grupo de Usuários de Testes de Software do Rio Grande do Sul, sendo que, os profissionais que voluntariam através desse convite foram os que participaram da pesquisa. Os participantes concordaram com a participação através um termo de consenso livre e esclarecido que pode ser visualizado no Apêndice D.

Tabela 6.2 – Perfil dos Sujeitos Participantes da *Survey* (Validação)

Área de Conhecimento	**Nível de Conhecimento**			
	Iniciante*	**Intermediário****	**Avançado*****	**Não tenho conhecimento**
Automação de testes	9	4	0	0
Gherkin	6	4	0	3
Métodos ágeis	4	7	2	0
Tempo de Trabalho (em anos)	**até um**	**até dois**	**dois a cinco**	**mais de cinco**
Testes de software	3	4	0	3

Durante o período em que os participantes responderam ao desafio o moderador não interviu, sendo que, os participantes precisaram realizar individualmente o desafio. As únicas dúvidas tiradas pelo moderador durante o desafio diziam respeito a funcionalidades da ferramenta e não a Aquila em si.

O período em que os participantes estiveram escrevendo os cenários e depois gerando os *scripts* teve duração de uma hora, sendo que, todos os participantes concluíram o desafio. Após realizar o desafio os participantes responderam às seguintes perguntas:

1. Sobre a dificuldade de aprendizado da DSL Aquila, você considera?
   (a) Fácil de aprender
   (b) Difícil de aprender
   (c) Médio aprendizado
   (d) Não consegui aprender
   (e) Não sei opinar
2. Comparando a facilidade de aprendizado da Aquila com outras ferramentas de testes que você tenha tido contato, você considera:
   (a) Mais fácil de aprender
   (b) Mais difícil de aprender
   (c) De igual nível de aprendizado
   (d) Não consegui aprender utilizar Aquila
3. Você acredita que em uma sprint de um projeto ágil seja possível escrever os cenários Aquila para geração dos *scripts* de teste?
   (a) Sim
   (b) Não

4. Descreva com suas palavras quais as principais vantagens e as principais desvantagens da utilização da DSL Aquila, em sua opinião, (questão dissertativa).

5. Quais melhorias você sugeriria para aprimorar a ferramenta que suporta a DSL Aquila? Considere sugestões de usabilidade, funcionalidade e correção de problemas (questão dissertativa).

6.2.1 Análise dos resultados

A primeira pergunta direcionada aos profissionais participantes, diz respeito às dificuldade encontrada para aprender a utilizar DSL Aquila. Nesse sentido, 10 participantes responderam que consideram fácil de aprender; Dois participantes consideraram que o nível de dificuldade para aprender Aquila é médio e um participante considerou difícil. Pode-se perceber com isso que o objetivo da DSL de ser de fácil compreensão e aprendizado foi atingido

Na segunda pergunta, é feita uma comparação entre a facilidade de aprender Aquila com a facilidade de aprender outras ferramentas de testes. Para esse cenário, 7 participantes consideram mais fácil de aprender; 4 de igual nível de aprendizado e 2 consideram mais difícil de aprender. Analisando estas estas respostas pode ser notado que a DSL Aquila foi considerada pela maior parte dos participantes mais fácil de aprender. Os dois participantes que consideraram mais difícil, não tem conhecimento de Gherkin e por isso sentiram dificuldade na utilização das palavras chave nos cenários.

A terceira pergunta teve por objetivo entender se seria possível aplicar a DSL Aquila em sprints de projetos ágeis e por isso versava sobre a possibilidade de escrever os cenários Aquila dentro de um Sprint. Nesta pergunta 12 participantes responderam que sim seria possível e apenas um que não. O grande número de respostas positivas a esta questão nos deu um maior embasamento para considerar a DSL aplicável para o contexto ágil de desenvolvimento de software

Em relação às perguntas dissertativas, a primeira pediu para que os participantes apresentassem sua opinião em relação a vantagens e desvantagens da DSL Aquila. A Figura 6.6 apresenta um mapa mental sumarizando as respostas obtidas.

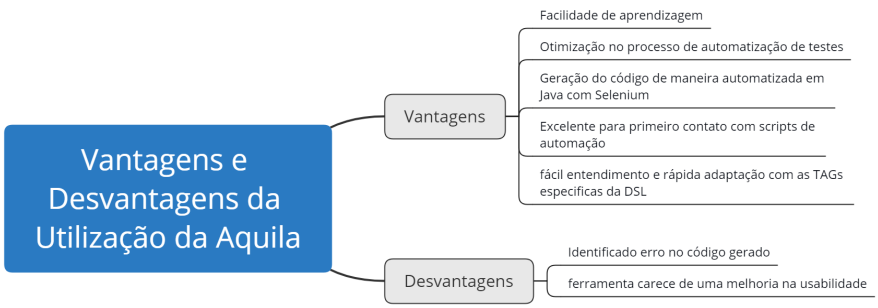

Figura 6.6 – Vantagens e Desvantagens da DSL Aquila

Analisando as respostas pode ser visto diversas referências para a facilidade de utilização da DSL, para o fato da ferramenta gerar scripts automatizados e ser de simples utilização. Como desvantagens destaca-se uma crítica referente ao código gerado pela ferramenta conter erros, esse código já foi revisado e os erros corrigidos, bem como foi revisada a usabilidade da ferramenta que suporta a DSL para atender a crítica de que a usabilidade da ferramenta estava insatisfatória.

A última pergunta discursiva diz respeito a sugestões de melhoria que podem ser feitas na ferramenta que suporta a DSL Aquila, os seguintes pontos foram apontados pelos participantes:

Deixar claro que a ferramenta gera *scripts* com *annotations* JUnit; Salvar por padrão o *script* gerado em um local e com um nome igual ao arquivo utilizado como entrada (input), acrescentando a extensão conforme a linguagem de programação utilizada; Mostrar mensagem de erro caso não consiga gerar *script* de testes, e informar porque não foi possível gerar; Além de ler o arquivo e gerar o *script* também ler o arquivo e se houver algo de errado no BDD ele apontar o que deve ser corrigido; Abranger mais TAGs com especificando melhor os cenários.

As sugestões apontadas estão todas relacionadas diretamente com a ferramenta e não com a solução para aplicação de MBT em AT. Estas sugestões foram todas registradas e estarão disponíveis em uma versão futura da ferramenta.

6.2.2 Discussão dos Resultados

Analisando os resultados apresentados na *survey* pode-se constatar que a abordagem de aplicação de MBT em AT, através da DSL Aquila pode prover benefícios às empresas que a utilizarem, especialmente no que diz respeito a curva de aprendizagem e a produtividade.

Contudo pode-se identificar também que pessoas não familiarizadas com Gherkin tendem a achar a DSL de difícil compreensão, nesse contexto avaliou-se a possibilidade de tornar a ferramenta que suporta a DSL Aquila mais intuitiva, pois entende-se que os problemas descritos em relação à dificuldade foram pelo participante ter tido problema para configuração e execução dos seus testes.

É importante enfatizar também que o fato de quando questionados sobre a viabilidade de execução do fluxo todo da Aquila em um sprint todos os participantes responderam que consideram possível. Isto nos dá um maior embasamento em relação a adequação da DSL em equipes ágeis de desenvolvimento de sistemas.

6.2.3 Ameaças a Validade do Estudo

Para este estudo identificou-se alguns pontos de atenção que podem ser considerados pontos de ameaças a viabilidade do estudo, sendo eles:

1. Baixo Número de Participantes
2. Perfil dos Participantes Iniciante
3. Pouca interação com a ferramenta
4. Utilização da ferramenta em ambiente controlado e não em cenário real.

Em relação ao número de participantes considera-se que embora seja um número reduzido de participantes, eles possuem perfis diversos que torna a amostra representativa suficiente. Em relação a experiência dos participantes considera-se que o fato de ter profissionais iniciantes em automação nos permitiu ter uma visão melhor sobre a curva de aprendizagem da utilização da DSL Aquila, sobre um ponto de vista de pessoas que estão iniciando na automação.

Em relação a pouca interação dos profissionais com a ferramenta e da utilização de um ambiente controlado e não de um projeto real, justifica-se pelo fato de ser a alternativa encontrada para que profissionais que atuam em diversas empresas diferentes pudessem participar do estudo. Não teve-se abertura de uma empresa específica para executar o estudo dentro da empresa, sendo necessário simular a utilização utilizando ambiente controlado.

7. CONSIDERAÇÕES FINAIS

A realização de testes de software é uma das estratégias que pode ser utilizada para melhorar a qualidade do produto entregue ao cliente. Neste trabalho abordamos a técnica de testes baseado em modelos aplicada num contexto de equipes ágeis de desenvolvimento de sistemas. Este trabalho embasou-se no fato de que a técnica de MBT é amplamente discutida em diversos trabalhos acadêmicos que exploram a sua utilização no contexto de ciclos de vida tradicionais de desenvolvimento de software.

Contudo, por não ser explorada em equipes ágeis, estas equipes não podem se beneficiar das vantagens oferecidas por MBT.Neste sentido buscamos neste trabalho identificar os motivos que fazem com que MBT não seja explorado em equipes ágeis e, alternativas para viabilizar a reversão deste contexto. A partir disso trabalhamos com a hipótese de que gerar modelos a partir de cenários poderia ser uma alternativa viável para aproximar MBT de times ágeis.

Através da geração de modelos a partir de cenários, possibilitamos que, com apenas um cenário, fossem mapeados diversos fluxos alternativos e gerados testes para garantir a cobertura de todos estes. Isto reduz o trabalho do testador de identificar os fluxos e evita erros humanos neste mapeamento de testes.

Contudo, a estratégia mais comum de criação de cenários em equipes ágeis, que é fazendo uso da DSL Gherkin não possui em sua essência a característica de detalhar o comportamento funcional do sistema, sendo usada em um nível maior de abstração. Neste sentido, propomos gerar modelos a partir de cenários escritos em uma nova DSL, chamada DSL Aquila, a qual apresentamos neste trabalho.

Esta DSL adiciona informações comportamentais nos cenários, utilizando palavras chaves genéricas que representam comportamentos comuns e não deixam o cenário estático a ponto de precisar ser refeito quando mínimas alterações ocorrem no sistema.

Para chegar a conclusão de que propor uma nova DSL era fazia necessário realizamos dois estudos preliminares, que nos permitiram entender as principais dificuldades no processo de implementação de MBT em times ágeis. Estes estudos foram uma revisão de literatura e um *survey* em formato de entrevistas com profissionais da área.

Identificamos que dois pontos que dificultam a utilização de MBT em métodos ágeis sendo eles: a falta de tempo e a falta de informações para criação dos modelos.

Visando entender como solucionar esses desafios criamos uma proposta de boas práticas para implementação de MBT em equipes ágeis, que contemplou com diversos pontos, entre eles: a necessidade de treinamento e conscientização da equipe e a necessidade de que o modelo no qual será aplicado MBT seja criado no início do ciclo de desenvolvimento, ainda na conversa com o cliente.

Essa proposta de boas práticas foi validada, por meio de questionário, por 30 profissionais de testes de software. Contudo, embora a proposta de boas práticas tenha tido uma boa aceitação,

identificou-se nas entrevistas e na literatura que, embora construir o modelo no início do ciclo de desenvolvimento do software facilite o processo, ainda, em uma *sprint* ágil, não existe tempo e informações para a criação ou conclusão da criação e detalhamento do modelo. Pensando nisso, propôs-se a geração automatizada desses modelos por meio de cenários, por ser uma estratégia de escrita de requisitos já familiar a equipes ágeis.

Para viabilizar tal procedimento, criou-se a DSL Aquila, definiu-se as duas palavras-chave que formam a arquitetura básica da linguagem, e propôs-se que, para cada palavra-chave escrita no cenário, fosse gerada uma nova vértice em um DAG, exceto em casos em que existisse mais de uma entrada para ser validada, quando são geradas mais de uma vértice para caracterizar o fluxo alternativo.

Para testar essa abordagem, foi criada uma ferramenta que suporta a DSL Aquila, chamada de Aquila Tool. Essa ferramenta é utilizada para efetuar a geração dos modelos a partir dos cenários Áquila e também de *scripts* a partir desses modelos. Depois de construído os modelos, para efetuar a geração de *script* é utilizado o algoritmo DFS.

A DSL Aquila e a proposta de gerar modelos a partir de cenários, baseando-se em palavras-chave, foi validada por meio de dois estudos: um estudo de grupo focal e um *survey* em formato de questionário. No estudo de grupo focal, foi apresentada a DSL e os participantes puderam avaliar a sua aplicabilidade. Já na *survey*, os participantes interagiram com a DSL Aquila, utilizando-a para modelar os cenários de uma aplicação e depois respondendo perguntas sobre essa experiência.

Analisando os estudos realizados, foi possível perceber que gerar modelos a partir de cenários, desonera as atividades de criação e manutenção de modelos e pode viabilizar a implementação de MBT em AT. Com isso, as equipes podem ter um ganho de produtividade e tempo. Essa é uma característica de MBT e também um ponto que foi apontado nos questionários quando os profissionais avaliaram a produtividade da DSL Aquila.

De forma geral, acredita-se que houve evolução do estado da arte em relação à integração dessas duas áreas: teste baseado em modelos e métodos ágeis. A ferramenta Aquila Tool está disponível para download e utilização na plataforma github, no endereço:
https://github.com/alinnezanin/Aquila.

Com a disponibilização da ferramenta acreditamos que beneficiam tanto o segmento empresarial como acadêmico, empresarial porque a ferramenta pode ser utilizada para realização dos testes nas empresas, bem como pode ser adaptada para o contexto da empresa. Já para a academia, disponibilizar a ferramenta ajuda os próximos pesquisadores a terem um ponto de partida para geração de modelos a partir de cenários.

Os estudos iniciais realizados para concepção da DSL foram apresentados no artigo “Uma Proposta de Boas Práticas para Aplicação de Teste Baseado em Modelos em Métodos Ágeis”, que foi publicado no ano de 2017 no evento: Simpósio Brasileiro e Qualidade de Software (SBQS). A DSL Aquila e a abordagem de geração de modelos a partir de cenários foram apresentados em dois eventos, um na Escola Regional de Engenharia de Software, no ano de 2018, e no CBSOFT - Brazilian Conference of Software no Workshop de Teses e Dissertações.

A DSL Aquila foi apresentada também, em um evento do Grupo de Usuários de Testes de Software, o qual teve uma boa aceitação da DSL, sendo que, existe uma fila de espera de participantes para uma próxima edição que ainda precisa ser agendadas.

Além disso, está sendo conduzida uma iniciativa de análise da utilização da DSL Aquila no ensino de testes de software. Até o momento, a DSL foi utilizada apenas com uma turma de alunos de cursos técnico, e ainda não foram publicados resultados desse estudo.

7.1 Limitações

A ferramenta Aquila Tool precisa passar por melhorias no código gerado, de forma a respeitar alguns padrões de projeto que facilitem a sua utilização em um contexto empresarial. A DSL Aquila em si pode ser complementada adicionando novas palavras-chave, porque atualmente existem ações, especialmente para o contexto mobile (por exemplo: abertura de camera, gps, bluetooth) que não são cobertas pela DSL Aquila. No entanto, caso alguma empresa ou pesquisador queira estender esse trabalho, a lógica da geração dos modelos segue sendo a mesma, sendo necessário apenas aumentar a lista de palavras-chave.

Além disso, não foi possível realizar a implantação da DSL Aquila em um time ágil para verificar os problemas e as vantagens. Sendo que os estudos realizados no âmbito deste trabalho, foram baseados na opinião de profissionais e na utilização da DSL em um ambiente controlado.

7.2 Trabalhos Futuros

Como trabalhos futuros, podem ser elencados os seguintes:

- Efetuar a publicação dos resultados deste trabalho em um periódico;
- Efetuar melhorias na ferramenta Aquila Tool, no sentido de corrigir falhas, adequar a padrões e criar parser para outras ferramentas de automação, prioritariamente para Appium e a migração da ferramenta Aquila Tool para plataforma web;
- Efetuar o registro da ferramenta Aquila Tool;
- Realizar um estudo de caso utilizando a DSL Aquila dentro de um time de automação de software;
- Estender a DSL Aquila para outros tipos de testes.
- Efetuar um estudo de performance da Aquila Tools, visando identificar o impacto do tamanhos dos cenários na performance da ferramenta

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Aichernig, B. K.; Lorber, F.; Tiran, S. “Formal test-driven development with verified test cases”. In: Proceedings of the International Conference on Model-Driven Engineering and Software Development (MODELSWARD), 2014, pp. 626–635.

[2] Albiero, F. W. “Uma abordagem de teste para aplicativos android utilizando os cenários do behavior driven development”, Dissertação de mestrado, Universidade Federal do Rio Grande do Sul.

[3] Anand, T.; Mani, V. “Practices to make agile test teams effective: Challenges and solutions”. In: Proceedings of the International Conference on Software Engineering International Conference on Global Software Engineering Workshops (ICGSE), 2015, pp. 7–11.

[4] Avizienis, A.; Laprie, J.-C.; Randell, B.; Landwehr, C. “Basic concepts and taxonomy of dependable and secure computing”, *IEEE Transactions on Dependable and Secure computing*, vol. 1–1, Jan 2004, pp. 11–33.

[5] Beck, K.; Beedle, M.; Van Bennekum, A.; Cockburn, A.; Cunningham, W.; Fowler, M.; Grenning, J.; Highsmith, J.; Hunt, A.; Jeffries, R.; et al.. “Manifesto for agile software development”. Recuperado de: http://www.agilemanifesto.org, junho 2019.

[6] Bernardino, M.; Zorzo, A. F.; Rodrigues, E. M. “Canopus: A domain-specific language for modeling performance testing”. In: Proceedings of the IEEE International Conference on Software Testing, Verification and Validation (ICST), 2016, pp. 157–167.

[7] Bierhoff, K.; Liongosari, E. S.; Swaminathan, K. S. “Incremental development of a domain-specific language that supports multiple application atyles”. In: Proceedings of the Workshop on Domain Specific Modeling (DSM), 2006, pp. 67–78.

[8] Chow, T. S. “Testing software design modeled by finite-state machines”, *IEEE Transactions on Software Engineering*, vol. 41–3, May 1978, pp. 178–187.

[9] Crispin, L.; Gregory, J. “Agile testing: A practical guide for testers and agile teams”. Pearson Education, 2009, 533p.

[10] Cucumber. “Cucumber”. Recuperado de: https://cucumber.io/, junho 2019.

[11] Cucumber. “Gherkin”. Recuperado de: https://cucumber.io/docs/gherkin/reference, junho de 2019.

[12] da Silveira, M. B. “Conjunto de características para teste de desempenho: uma visão a partir de modelos”, Dissertação de mestrado, Pontifícia Universidade Católica do Rio Grande do Sul.

[13] Dalal, S. R.; Jain, A.; Karunanithi, N.; Leaton, J.; Lott, C. M.; Patton, G. C.; Horowitz, B. M. “Model-based testing in practice”. In: Proceedings of the International Conference on Software Engineering (ICSE), 1999, pp. 285–294.

[14] Dwarakanath, A.; Era, D.; Priyadarshi, A.; Dubash, N.; Podder, S. “Accelerating test automation through a domain specific language”. In: Proceedings of the IEEE International Conference on Software Testing, Verification and Validation (ICST), 2017, pp. 460–467.

[15] Dwarakanath, A.; Era, D.; Priyadarshi, A.; Dubash, N.; Podder, S. “Accelerating test automation through a domain specific language”. In: Proceedings of the International Conference on Software Testing, Verification and Validation (ICST), 2017, pp. 460–467.

[16] El-Far, I. K.; Whittaker, J. A. “Encyclopedia of Software Engineering”. American Cancer Society, 2001, cap. 2, pp. 825–837.

[17] Elallaoui, M.; Nafil, K.; Touahni, R. “Automatic generation of uml sequence diagrams from user stories in scrum process”. In: Proceedings of the International Conference on Intelligent Systems: Theories and Applications (SITA), 2015, pp. 1–6.

[18] Elallaoui, M.; Nafil, K.; Touahni, R.; Messoussi, R. “Automated model driven testing using andromda and uml2 testing profile in scrum process”, *Procedia Computer Science*, vol. 100–83, Dec 2016, pp. 221–228.

[19] Entin, V.; Winder, M.; Zhang, B.; Christmann, S. “Combining model-based and capture-replay testing techniques of graphical user interfaces: An industrial approach”. In: Proceedings of the International Conference on Software Testing, Verification and Validation Workshops (ICST), 2011, pp. 572–577.

[20] Entin, V.; Winder, M.; Zhang, B.; Christmann, S. “Combining model-based and capture-replay testing techniques of graphical user interfaces: An industrial approach”. In: Proceedings of the International Conference on Software Testing, Verification and Validation Workshops (ICSTW), 2011, pp. 572–577.

[21] Entin, V.; Winder, M.; Zhang, B.; Christmann, S. “Introducing model-based testing in an industrial scrum project”. In: Proceedings of the International Workshop on Automation of Software Test (AST), 2012, pp. 43–49.

[22] Entin, V.; Winder, M.; Zhang, B.; Claus, A. “A process to iicrease the model quality in the context of model-based testing”. In: Proceedings of the IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW), 2015, pp. 1–7.

[23] Entin, V.; Winder, M.; Zhang, B.; Claus, A. “A process to increase the model quality in the context of model-based testing”. In: Proceedings of the International Conference on Software Testing, Verification and Validation Workshops (ICSTW), 2015, pp. 1–7.

[24] Fujiwara, S.; Khendek, F.; Amalou, M.; Ghedamsi, A.; et al.. “Test selection based on finite state models”, *IEEE Transactions on Software Engineering*, vol. 17–6, Jun 1991, pp. 591–603.

[25] Gomes, A. F. “Agile: Desenvolvimento de software com entregas frequentes e foco no valor de negócio”. Editora Casa do Código, 2014, 208p.

[26] International Software Qualification Board, I. “Certified tester foundation level syllabus”. Recuperado de: http://www.istqb.org/downloads/syllabi/foundation-level-syllabus.html, junho 2019.

[27] International Software Qualification Board, I. “Glossário padrão de termos utilizados em testes de software”. Recuperado de: https://www.bstqb.org.br/uploads/glossario/glossario_ctfl_3.2br.pdf, junho 2019.

[28] International Software Qualification Board, I. “Syllabus foundation level - model based tes”. Recuperado de: https://www.bstqb.org.br/uploads/syllabus/syllabus_ctfl_mbt_2015br.pdf, junho 2019.

[29] Jalalinasab, D.; Ramsin, R. “Towards model-based testing patterns for enhancing agile methodologies.” In: Proceedings of the International Conference on Intelligent Software Methodologies, tools, and Techniques (SOMET), 2012, pp. 57–72.

[30] Katara, M.; Kervinen, A. “Making model-based testing more agile: A use case driven approach”. In: Proceedings of the Haifa Verification Conference (HVC), 2006, pp. 219–234.

[31] King, T. M.; Nunez, G.; Santiago, D.; Cando, A.; Mack, C. “Legend: an agile dsl toolset for web acceptance testing”. In: Proceedings of the International Symposium on Software Testing and Analysis (ISSTA), 2014, pp. 409–412.

[32] Kitchenham, B.; Brereton, O. P.; Budgen, D.; Turner, M.; Bailey, J.; Linkman, S. “Systematic literature reviews in software engineering–a systematic literature review”, *Information and Software Technology*, vol. 51–1, Jan 2009, pp. 7–15.

[33] Krueger, R. A.; Casey, M. A. “Designing and conducting focus group interviews”. Recuperado de: https://www.eiu.edu/ihec/Krueger-FocusGroupInterviews.pdf, junho de 2019.

[34] Li, N.; Escalona, A.; Kamal, T. “Skyfire: Model-based testing with cucumber”. In: Proceedings of the IEEE International Conference on Software Testing, Verification and Validation (ICST), 2016, pp. 393–400.

[35] Li, N.; Offutt, J. “A test automation language framework for behavioral models”. In: Proceedings of the International Conference on Software Testing, Verification and Validation Workshops (ICSTW), 2015, pp. 1–10.

[36] Löffler, R.; Güldali, B.; Geisen, S. “Towards model-based acceptance testing for scrum”. Recuperado de:https://pdfs.semanticscholar.org/d21a/de37eceb90f550aca97507cbffaa01c2ab4e.pdf junho 2019.

[37] Luo, G.; Petrenko, A.; v. Bochmann, G. “Selecting Test Sequences for Partially-Specified Nondeterministic Finite State Machines”. Springer US, 1995, cap. 6, pp. 95–110.

[38] Mernik, M.; Heering, J.; Sloane, A. M. “When and how to develop domain-specific languages”, *ACM Computing Surveys*, vol. 37–4, Dec 2005, pp. 316–344.

[39] Myers, G. J.; Sandler, C.; Badgett, T. “The art of software testing”. John Wiley & Sons, 2011, 240p.

[40] (ONG)), O. M. G. “Introduction to omg’s unified modeling language (uml)”. Recuperado de: http://www.uml.org/what-is-uml.htm, junho 2018.

[41] Petrenko, A.; Simao, A.; Maldonado, J. “Model-based testing of software and systems: Recent advances and challenges”, *Procedia Computer Science*, vol. 4–14, Dec 2016, pp. 383–386.

[42] Pretschner, A.; Lotzbeyer, H.; Philipps, J. “Model based testing in evolutionary software development”. In: Proceedings of the International Workshop on Rapid System Prototyping (RSP), 2001, pp. 155–160.

[43] Pretschner, A.; Lötzbeyer, H.; Philipps, J. “Model based testing in incremental system development”, *Journal of Systems and Software*, vol. 70–3, Mar 2004, pp. 315–329.

[44] Sabbagh, R. “Scrum: Gestão ágil para projetos de sucesso”. Casa do Código, 2014, 319p.

[45] Sanz, C.; Salas, A.; De Miguel, M.; Alonso, A.; De La Puente, J.; Benac, C. “Automated model-based testing based on an agnostic-platform modeling language”. In: Proceedings of the International Conference on Model-Driven Engineering and Software Development, Proceedings (MODELSWARD), 2015, pp. 239–246.

[46] Sivanandan, S.; B, Y. C. “Agile development cycle: Approach to design an effective model based testing with behaviour driven automation framework”. In: Proceedings of the International Conference on Advanced Computing and Communications (ADCOM), 2014, pp. 22–25.

[47] Smart, J. “BDD in Action: Behavior-Driven Development for the Whole Software Lifecycle”. Manning Publications Company, 2014, 353p.

[48] Sommerville, I. “Engenharia de software”. Pearson Brasil, 2011, 544p.

[49] Sommerville, I.; Sawyer, P. “Requirements engineering: a good practice guide”. John Wiley & Sons, Inc., 1997, 404p.

[50] Tarjan, R. “Depth-first search and linear graph algorithms”, *SIAM Journal on Computing*, vol. 1–2, Oct 1972, pp. 146–160.

[51] Teles, V. M. “Extreme Programming: Aprenda como encantar seus usuários desenvolvendo software com agilidade e alta qualidade”. Novatec Editora, 2017, 328p.

[52] Thummalapenta, S.; Sinha, S.; Singhania, N.; Chandra, S. “Automating test automation”. In: Proceedings of the International Conference on Software Engineering (ICSE), 2012, pp. 881–891.

[53] Törsel, A. “A testing tool for web applications using a domain-specific modelling language and the nusmv model checker”. In: Proceedings of the International Conference on Software Testing, Verification and Validation (ICST), 2013, pp. 383–390.

[54] Utting, M.; Pretschner, A.; Legeard, B. “A taxonomy of model-based testing approaches”, *Software Testing Verification and Reliability*, vol. 22–5, Aug 2012, pp. 297–312.

[55] van den Broek, R.; Bonsangue, M. M.; Chaudron, M.; van Merode, H. “Integrating testing into agile software development processes”. In: Proceedings of the International Conference on Model-Driven Engineering and Software Development (MODELSWARD), 2014, pp. 561–574.

[56] Wohlin, C. “Guidelines for snowballing in systematic literature studies and a replication in software engineering”. In: Proceedings of the International Conference on Evaluation and Assessment in Software Engineering (EASE), 2014, pp. 38:1–38:10.

[57] Wynne, M.; Hellesoy, A.; Tooke, S. “The Cucumber Book: Behaviour-Driven Development for Testers and Developers”. Pragmatic Programmers, 2017, 336p.

[58] Yue, T.; Ali, S.; Zhang, M. “Rtcm: A natural language based, automated, and practical test case generation framework”. In: Proceedings of the International Symposium on Software Testing and Analysis (SIGSOFT), 2015, pp. 397–408.

APÊNDICE A – ROTEIRO DO ESTUDO DE GRUPO FOCAL

GUIDE - Focus Group: AQUILA: Uma Linguagem de Domínio Específico para Testes Baseados em Modelos em Equipes Ágeis

- **Mensagem Aos Participantes:**

Obrigada pela presença de todos e reitero que sua opinião é muito importante para o trabalho.

- **Procedimentos Iniciais:**

Sua pasta possui:
- Um Formulário de Consentimento, para o qual peço que seja acompanhada a leitura e posteriormente assinado, no caso do acordo de participação. Novamente, colocado dentro da pasta;
- Um questionário para identificação de perfil do participante que deve ser preenchido antes do início da discussão;
- Um questionário referente a DSL Aquila que deverá ser preenchido de forma individual, após as explanações iniciais sobre a DSL;
- O material de apoio sobre o uso da DSL Aquila, que estará disponível para consulta durante toda a discussão;

- **Objetivo Geral do *Focus Group*:**

Identificar a opinião de especialistas da área a respeito da aplicabilidade da DSL Aquila no que diz respeito especialmente a produtividade e curva de aprendizagem

- **Estrutura do Focus Group**

Esta sessão de Focus Group será dividida em três momentos distintos, sendo eles:

- ➢ Leitura e assinatura do termo de consentimento e preenchimento do formulário de identificação de perfil do participante;
- ➢ Apresentação da DSL Aquila
- ➢ Resposta ao formulário de avaliação individual a DSL Aquila;
- ➢ Discussão coletiva sobre perguntas referentes a DSL Aquila;

- **Contextualização**

o **DSL Aquila**

A DSL Aquila é uma Linguagem de Domínio Específico que tem por objetivo permitir a geração automatizada de scripts de teste a partir de cenários Aquila. Os cenários Aquila são derivados de cenários Gherkin, utilizando a mesma estrutura e adicionando a eles novas palavras chaves que permitem interagir com o sistema gerando código de testes. Paralelo a isso a DSL Aquila implementa a técnica de testes *model based testing* (teste baseado em modelos) o que provê uma otimização dos scripts gerados, e a geração automatizada de modelos que representam o fluxo comportamental dos cenários que estão sendo testados.

o **Domain Specific Language (DSL)**

É um tipo de linguagem de programação ou especificação utilizado em desenvolvimento de software e engenharia de domínio. Essas linguagens tem por objetivo sanar um problema de um domínio específico, e por este motivo buscam utilizar em sua sintaxe, palavras que sejam familiares neste domínio. Um exemplo bastante difundido de DSL é o SQL, utilizado para manipulação de dados em banco de dados. Toda DSL tem sua própria sintaxe e não deve objetivar ser utilizada fora de seu domínio.

APÊNDICE B – TERMO DE CONSENTIMENTO - FOCUS GROUP

Termo de consentimento

Aquila: uma Linguagem de Domínio Específico para Teste Funcional Baseado em Modelos para Projetos Ágeis de Desenvolvimento de Software

Escola Politécnica / PUCRS
Avenida Ipiranga, 6681 - Prédio 32 - Sala 635
90619-900 - Porto Alegre – RS
Tel: (51) 3320-3558, ramal 8635

Participante:_____Data: _____

Você está sendo convidado a participar da pesquisa " **Aquila: uma Linguagem de Domínio Específico para Teste Funcional Baseado em Modelos para Projetos Ágeis de Desenvolvimento de Software" sob a responsabilidade da estudante de doutorado Aline Zanin**, sob a orientação do Professor Dr. Avelino Zorzo.

Você participará de um grupo focal que irá analisar e discutir **a aplicabilidade da DSL Aquila especialmente no que tange os reflexos da utilização desta DSL na produtividade e na curva de aprendizagem dos times que utilizam.**

Haverá **a aplicação** de **um** questionário, que registrará seu perfil e coletará informações sobre sua **percepção a respeito da DSL Aquila.** As discussões serão gravadas através de anotações, fotos, vídeo e gravação de áudio. As informações obtidas através desta pesquisa serão confidenciais e garantimos a confidencialidade de sua participação. Assim, os dados divulgados não permitirão qualquer identificação.

Sua participação é voluntária e se você decidir não participar ou desej**ar** cancelar sua participação a qualquer momento, você tem a liberdade absoluta de fazê-lo.

Mesmo sem ter benefícios diretos na participação, indiretamente você estará contribuindo para a compreensão do fenômeno estudado e para a produção de conhecimento científico.

Qualquer dúvida sobre a pesquisa pode ser feita através dos e-mails dos pesquisadores: aline.zanin@acad.pucrs.br e avelino.zorzo@pucrs.br e fone (51) 988886990

DECLARAÇÃO DE CONSENTIMENTO DO PARTICIPANTE DO ESTUDO

Eu concordo em participar deste estudo e declaro ter lido os detalhes descritos neste documento. Eu entendo que sou livre para aceitar ou recusar, e que posso interromper minha participação a qualquer momento sem dar um motivo. Eu concordo que os dados coletados serão usados para o propósito descrito acima. Compreendo as informações apresentadas nos TERMOS DE CONSENTIMENTO. Tive a oportunidade de fazer perguntas e todas as minhas perguntas foram respondidas. Recebi uma cópia assinada e datada deste documento de CONSENTIMENTO LIVRE E ESCLARECIDO.

[A ser preenchido pelos pesquisadores] DSL Aquila Condições Especiais (se não existem condições especiais, escrever "não): _____ _____ _____ _____	Com o conhecimento da informação exposta, expresso meu acordo de vontade espontânea de participar do estudo. _____ Assinatura do participante _____ Assinatura do Pesquisador: Aline Zanin _____ Assinatura do Pesquisador: Avelino Zorzo

Nos agradecemos a sua participação neste estudo.

APÊNDICE C – QUESTIONÁRIO FOCUS GROUP

Questionário Individual de Avaliação da DSL Aquila

1. Você acredita que a utilização da DSL Aquila pode interferir na produtividade da equipe? Se sim, de que forma?

2. Em comparação com a criação manual de scripts para automação de testes você consideraria a DSL Aquila mais produtiva, menos produtiva ou indiferente, justifique.

3. Qual a sua percepção em relação a curva de aprendizagem da utilização da DSL Aquila em relação a curva de aprendizagem para a criação manual de scripts para automação de testes

4. Você aplicaria a DSL Aquila em um projeto de testes considerando um time ágil? Justifique

5. No caso de aplicar a DSL Aquila em um projeto, quem você acha que deverá se responsabilizar pela escrita dos cenários?

   [ ] Área de Negócios/Representante do Cliente escreve o cenário no formato Gherkin e **Testador** adapta colocando as palavras chaves da Aquila
   [ ] Área de Negócios/Representante do Cliente escreve o cenário no formato Gherkin e **Programador** adapta colocando as palavras chaves da Aquila
   [ ] Área de Negócios/Representante do Cliente escreve o cenário no formato Gherkin e **Time de Desenvolvimento** adapta colocando as palavras chaves da Aquila
   [ ] Área de Negócios/Representante do Cliente escreve cenário no formato Aquila e **Time de Desenvolvimento** efetua adaptações
   [ ] Área de Negócios/Representante do Cliente escreve os requisitos em outro formato e **Programador/Testador** escrevem os cenários colocando as palavras chaves da Aquila
   [ ] Área de Negócios/Representante do Cliente escreve os requisitos em outro formato, **programador** coloca os requisitos no padrão Gherkin e **testador** coloca no padrão Aquila
   [ ] Área de Negócios/Representante do Cliente escreve o cenário em outro formato e **Time de Desenvolvimento** adapta criando os cenários no formato Aquila
   [ ] Área de Negócios/Representante do Cliente escreve o cenário em outro formato e **Time de Desenvolvimento** adapta criando os cenários no formato Gherkin e **Testador** coloca palavras chave **Aquila**

APÊNDICE D – TERMO DE CONSENTIMENTO - SURVEY

Termo de consentimento

Aquila: uma Linguagem de Domínio Específico para Teste Funcional Baseado em Modelos para Projetos Ágeis de Desenvolvimento de Software

Escola Politécnica / PUCRS
Avenida Ipiranga, 6681 - Prédio 32 - Sala 635
90619-900 - Porto Alegre – RS
Tel: (51) 3320-3558, ramal 8635

Participante:_____Data: _____

Você está sendo convidado a participar da pesquisa " **Aquila: uma Linguagem de Domínio Específico para Teste Funcional Baseado em Modelos para Projetos Ágeis de Desenvolvimento de Software" sob a responsabilidade da estudante de doutorado Aline Zanin**, sob a orientação do Professor Dr. Avelino Zorzo.

Você participará de uma survey, em formato de questionário que irá analisar e discutir **a aplicabilidade da DSL Aquila especialmente no que tange os reflexos da utilização desta DSL na produtividade e na curva de aprendizagem dos times que utilizam.**

Haverá **a aplicação** de **um** questionário, que registrará seu perfil e outro que coletará informações sobre sua percepção a respeito da DSL Aquila. Antes de responder o segundo questionário você irá assistir a uma demonstração do funcionamento da DSL Áquila e irá utilizar a ferramenta para realizar a automação de testes de alguns cenários especificados pela pesquisadora. Não haverá registro de áudio ou vídeo durante o estudo , apenas registro de imagem.

Sua participação é voluntária e se você decidir não participar ou desej**ar** cancelar sua participação a qualquer momento, você tem a liberdade absoluta de fazê-lo.

Mesmo sem ter benefícios diretos na participação, indiretamente você estará contribuindo para a compreensão do fenômeno estudado e para a produção de conhecimento científico.

Qualquer dúvida sobre a pesquisa pode ser feita através dos e-mails dos pesquisadores: aline.zanin@edu.pucrs.br e avelino.zorzo@pucrs.br.

DECLARAÇÃO DE CONSENTIMENTO DO PARTICIPANTE DO ESTUDO

Eu concordo em participar deste estudo e declaro ter lido os detalhes descritos neste documento. Eu entendo que sou livre para aceitar ou recusar, e que posso interromper minha participação a qualquer momento sem dar um motivo. Eu concordo que os dados coletados serão usados para o propósito descrito acima. Compreendo as informações apresentadas nos TERMOS DE CONSENTIMENTO. Tive a oportunidade de fazer perguntas e todas as minhas perguntas foram respondidas. Recebi uma cópia assinada e datada deste documento de CONSENTIMENTO LIVRE E ESCLARECIDO.

[A ser preenchido pelos pesquisadores] DSL Aquila Condições Especiais (se não existem condições especiais, escrever "não): _____ _____ _____ _____	Com o conhecimento da informação exposta, expresso meu acordo de vontade espontânea de participar do estudo. _____ Assinatura do participante _____ Assinatura do Pesquisador: Aline Zanin _____ Assinatura do Pesquisador: Avelino Zorzo

Nos agradecemos a sua participação neste estudo.

APÊNDICE E – PLANILHA CATEGORIZAÇÃO DA ANÁLISE DAS ENTREVISTAS

ID	Dificuldades	Melhorias	Processo
S1	Ausência de informações para criar os modelos em um momento inicial do projeto `` as user story como as feature elas são mais assim mais incrementais a gente vai incrementando trabalhando nas groumings"  No caso de funcionalidades que não são completamente definidas em uma sprint não existiria informação suficinete para criação dos modelos  ``Tem times que fazem as user stories tão quebradas que (...) acaba sendo produzida em duas, três e não teria como fazer durante a sprint teria que ser durante mais de uma" Resistência a mudanças por parte dos profissionais da equipe `` único problema de sempre adicionar alguma coisa nova é que sempre vai ter uma resistência por parte de qualquer pessoa no time de desenvolvimento e isso pode acarretar algum a demora na entrega"  Nível de dificuldade maior do que a criação manual de testes  `` Eu acho que mesmo que utilizando os modelos em relação aos testes manuais a automação sempre vai ser um pouco mais complicada" S1 considera mais demorado criar os modelos do que fazer o teste manual ``S1: Eu acho que seria mais demorado fazer a automação dos casos de testes. Entrevistador: Com modelos? S1: Com os modelos, isso!"  Quando já existe uma parte do modelo criado se torna fácil automatizar  `` regressão já tem o código já temos as funcionalidades já sabemos o que testar exatamente onde testar e, por exemplo, se for automatizada já sabemos os asserts, já temos tudo o que precisamos, agora se for criar uma coisa do zero eu acredito que vai demandar um pouco mais de tempo, vamos dizer que é uma funcionalidade para adição de novos clientes e que tenha trinta campos vai ter que ir lá pegar todos os campos e qual é e colocar no modelo e não sabe quais são os campos até saber fazer tudo acho que vai demorar um pouco mais de tempo. "	S1 não apresenta sugestões de melhoria apenas reforça que se a partir do modelo for gerado algum código para automação ou mesmo os steps para o teste manual já ajudaria a equipe porque nos primeiros dias da sprint se perde muito tempo em cerimônias e neste caso, se quando for iniciar os testes já existir algum artefato gerado ajudaria a ser um ponto de partida	PO E Tester são responsáveis pela criação do modelo e o dev ajuda pontualmente ``Acho que o Po e o Tester eles são os responsáveis e o dev seria mais accountable para alguma coisa que eles pudessem um input ou alguma coisa assim ele também é importante" Acredita que na maioria dos casos em uma sprint existe tempo e informações para criar o modelo, exceto em casos onde existe interdependência com stories que não são parte desta sprint ``Acredito que sim mais de noventa por cento dos casos sim, vão ter alguns casos que não serão possíveis por alguma interdependência"

ID	Dificuldades	Melhorias	Processo
S2	A primeira dificuldade relatada por S2 está na manutenção dos modelo ``Bom, primeiro de tudo é manter o modelo. Ter um modelo unificado e manter essa documentação, vamos dizer assim, atualizada, é algo que só faria sentido no contexto ágil"  Resistência da equipe, necessidade de provar os benefícios para que a equipe concorde em utilizar  ``Porque se não tem benefício em manter a documentação, não tem como convencer uma equipe ágil a fazer isso.'' As US, que são o principal documento com requisitos não tem informações suficientes ``User Story não tem a informação sobre interação com tela ou encaminhamento do usuário final no conjunto de tela. Então, user story já não é um bom input.''  Equipes ágeis tem muitos requisitos definidos com conversação, que não são incluídos nos documentos formais  ``O passo que está faltando aí é essa conversa entre quem está precisando dessa funcionalidade e a equipe ou o desenvolvedor ou quem é que vai fazer aquela coisa funcionar para que isso, essa troca de informações, aconteça. Então essas conversas que ocorrem para colocar informações que estão faltando nos documentos formais''	S2 sugere que o processo de MBT precisa gerar as verificações que o testador comumente faria manualmente, requisitando a menor manutenção possível nos scripts `` a partir desse modelo, um sistema que a gente criou faz as verificações adequadas. Eu não preciso programar as verificações, eu não preciso ter um profissional de teste fazendo aquela customização daquelas verificações. " S2 afirma que o processo de MBT precisa ser automatizado para fazer sentido: ``Se for manual, não faz o menor sentido. Aí, a gente não está nem falando de sprint, não está falando de ágil e a gente não está nem fazendo algo decente."	S2 afirma que receber um documento de requisitos completo não faz sentido em equipes ágeis, embora acredite que para criar os modelos seria adequado ``Bom, receber um documento de requisitos inteiro para fazer a tradução para um modelo adequado é uma hipótese, não faz muito sentido em ágil''  S2 acredita que os modelos devem ser construídos colaborativamente por todo time de desenvolvimento ``Deveria, na minha cabeça… No papel ideal seria a equipe de desenvolvimento, equipe técnica, seja ela composta de um desenvolvedor, um tester, um po e tem que ser algo da equipe, não pode ser responsabilidade de um papel,'' S2 acredita que para ter informações para construção do modelo é necessário incluir como parte dos requisitos conversas com o cliente ``Isso depende do que você quer dizer com requisitos. Se você quer dizer user Story, não, você não tem informação suficiente. Se você quer dizer cenários de BDD puro e simples, não, você não tem informação suficiente. Se você quiser dizer, com requisitos, tu ter a user story ou o cenário de BDD, ou um pedaço de papel e uma conversa de análise daquele requisito para que a equipe técnica e o cliente possam se conversar, numa reunião que é muitas vezes chamada de grooming, mas ela, também, pode ter o nome de… É o momento de conversar sobre os requisitos. O grooming é o jeito mais adequado, mas o pessoal de BDD tem uns nomes mais bonitinhos para derivar exemplos de user story, por exemplo. Mas se tu juntar o documento com essa conversa, sim, é a sprint adequada. Se tu dizer “não, só o documento de user story, ou só o documento de cenários de BDD”, não, não acho adequado."

Figura E.1 – Categorização da Análise das Entrevista- Sujeito S2

ID	Dificuldades	Melhorias	Processo
s2	Se o modelo não é atualizado corretamente ele perde sua função porque não representa mais a aplicação `` se não for atualizado correntemente, ele vai parar de refletir o estado da aplicação ''  S2 considera mais trabalhoso criar modelos do que executar testes manualmente  ``é mais trabalhoso fazer um modelo que vai gerar automaticamente os casos de teste.''	S2 afirma que para conseguir executar todo o processo de MBT, o modelo precisa ter sido construído continuamente e antes do momento dos testes finais, contudo faz ressalvas de que em alguns casos pode ter tempo para construir o modelo dentro da sprint: ``Por exemplo, se a sprint for fazer uma página de login, é adequado eu pensar que eu vou conseguir não gastar muito tempo em desenvolvimento de sistema, eu vou conseguir pegar aquela galera que está trabalhando muito em desenvolvimento de sistema para criar, também, um sistema de verificação adequado, que é o sistema baseado em modelo. ''	S2 descreve vantagens de MBT que vão além da geração de scripts automatizados ``se alguém precisar de documentação do que aqueles casos de testes fazem, ou do que a gente está pensando, ou gerar um recorte, isso tudo tende a ser mais facilitado porque o modelo está ali para isso e tirar o modelo, dar zoom ou qualquer coisa assim, é mais fácil porque tem o modelo, ponto.'' S2 considera que no caso de testes de regressão onde já existe modelo criado se torna muito mais fácil realizar o processo de MBT `` Na primeira, o meu esforço é de atualizar o modelo, é basicamente o meu esforço de aprender a ferramenta e modificar. Na segunda, eu tenho que fazer modelos novos, assim, atualizar o que eu tenho e tenho que juntar com o que eu já tinha. Então, claro. É muito mais fácil.''

Figura E.2 – Categorização da Análise das Entrevista- Sujeito S2 (Continuação)

ID	Dificuldades	Melhorias	Processo
S3	Algumas informações não são formalizadas, as histórias muitas vezes não são detalhes não tem informações ``eu noto que também informações ficam, digamos, não são registradas, não são formalizadas"  Trabalho duplicado, no momento em que se tem histórias escritas e depois é feita modelagem  ``a gente poderia ter um trabalho aí duplicado, porque a P.O. escreve as histórias e o intérprete faz a modelagem" Falta de informações para criar os modelos ``os critérios de aceitação que nós temos hoje nas histórias, eles são critérios de aceitação em ter negócio."  Não existe tempo hábil para execução de todo o processo dentro de uma sprint  ``Hoje, aqui, na realidade, no meu time, eu confesso que eu tenho dúvidas se a gente conseguiria adotar isso numa sprint, porque o nosso time tá fazendo sprints de uma semana. Então, é um curto espaço de tempo pra gente poder se envolver em modelar e testar." Poucas pessoas utilizam modelagem com UML em times ágeis ``A única questão que eu tenho em relação a isso, é da adoção que o mercado tem em relação com o uml, no sentido de que são poucas as pessoas que eu conheço, do meu círculo, que, de fato, usam, modelam, fazem modelagem com UML"	O processo precisa gerar exclusivamente scripts de teste, gerar testes manuais não agrega porque a equipe não faz testes manuais `` o pessoal aqui não faz, não escreve casos de teste. Faz faz automação ou faz teste manual exploratório. Então, aqui, o time não tem necessidade de criar casos de teste, "S3 Sugere que a UML precisa se modernizar para que possa ser utilizada	Idealmente S3 acredita que o PO deveria criar modelos em vez de apenas escrever as histórias mas também não vê problema em qualquer pessoa do time de desenvolvimento traduzir as histórias para modelos

Figura E.3 – Categorização da Análise das Entrevista- Sujeito S3

ID	Dificuldades	Melhorias	Processo
s3	Falta de conhecimento da equipe para realizar modelagem `` . Existe uma modelagem aí em caso de uso, ou algo parecido, teria que treinar as pessoas, treinar P.O., para poder utilizar os recursos da UML"  Criar modelo é mais trabalhoso do que automatizar manualmente  ``Orador B: Eu acho que, inicialmente, é mais trabalhoso. Orador A: Mais trabalhoso criar o modelo? Orador B: É, exato. Aqui a gente usa BDD pra fazer automação de testes, já escreve padrão BDD e a gente usa um framework produzido aqui internamente." O time não ver valor na modelagem ``E eu colocaria outro fator risco é de talvez, o time não enxergar o valor nisso. Um dos desafios, não só de treinar o time, mas de mostrar o valor e conseguir convencer os benefícios dessa modelagem."	`` UML deveria se modernizar. Eu acho que ela deveria acompanhar todas essas novas ferramentas que iam surgindo para apoiar o ágil. Talvez, tornar mais interativo essa construção. "	`` Eu acredito que qualquer um do time poderia fazer essas entradas. Se fosse considerar um modelo, assim, não vou dizer perfeito, mas, talvez, mais real, ao invés do P.O. registrar essas histórias na ferramenta, P.O. poderia já desenvolver a modelagem inicial, mas não vejo também problemas de qualquer pessoa do time, eventualmente, ter que fazer a tradução dessas histórias em modelagem. "
S4	Falta de informações em uma sprint para criação dos modelos. S4 afirma que terá informações para criar o modelo apenas em sprints maiores onde tem mais tempo. Em uma sprint não existe tempo hábil para executar todo processo de MBT ``quem trabalha com testes sabe que o nosso tempo de teste, de análise, enfim, todo processo, é o menor tempo. Geralmente. Então, eu acho que não, não é possível. Quer dizer, possível é, mas não é uma regra"  Esta técnica pode causar um consumo de tempo que as equipes não tem tem disponível  ``Eu acho que pode trazer desvantagens no sentido de tempo. Mas eu aí eu acho que vai muito de encontro de como a equipe em si trabalha. Eu acho que, assim, a inserção desses modelos, ela poderia trazer uma desvantagem em quesito que, sei lá, seja o PO, seja o Desenvolvedor enfim, a equipe toda do processo de desenvolvimento. Tem que realocar um tempo maior que está acostumado para começar a trabalhar com esse novo modelo. "	Se não for necessário criar o modelo manualmente, o profissional acredita que daí a técnica não terá nenhuma desvantagem, apenas benefícios ``Se não considerarmos o tempo de criação do modelo, apenas o tempo de aplicação da técnica, eu acho que eu... não vejo desvantagem. Eu acredito que daí a gente possa até encaixar a resposta da outra pergunta. Traria mais benefícios, na verdade."	O time todo constrói o modelo, tendo PO e QA apenas como pessoas chave `` Na minha visão, eu acredito que teria que ter o envolvimento do PO, do QA e em alguns prazos... na verdade, assim, eu acredito que deveria envolver o time todo. Eu não contaria como uma responsabilidade do próprio QA e como é utilizado esse modelo no processo todo, em teoria, quando eu recebo o caso de uso eu tenho já o cenário descrito."

Figura E.4 – Categorização da Análise das Entrevista- Sujeito S3 (Continuação) e Sujeito S4

ID	Dificuldades	Melhorias	Processo
S5	S5 acredita que embora consiga construir em uma sprint o modelo para gerar os testes, este modelo poderia na sprint imediatamente posterior precisar ser alterado significativamente por uma mudança ou adição de requisitos ``Eu acho que em uma sprint talvez eu até tenha informações suficientes para gerar o modelo mas provavelmente vai sofrer muitas alterações já na próxima sprint e eu vou ter que alterar o modelo." S5 acredita que em uma sprint geralmente não é possível efetuar todo o processo de MBT por falta de tempo, exceto em projetos menores com baixa demanda `` Execução eu acho muito difícil, bah mas também né, se bem que assim eu tenho um parâmetro muito complexo do meu projeto e se fosse um projeto extremamente simples onde eu já pudesse implementar uma estória de usuário completa na primeira sprint talvez seja possível executar o processo todo," O processo de MBT adiciona uma etapa a mais no processo de testes, a criação de modelos e isso pode não ser positivamente recebido pelas equipes no cotidiano de trabalho ``A desvantagem é que a criação do modelo é uma etapa a mais, um trabalho a mais que a equipe tem que fazer ao invés de ir na ferramenta e criar os casos, talvez demande um pouco mais de tempo."	S5 não sugere melhorias efetivas, apenas comenta que se o processo permitir que alterando o modelo sejam refletidas alterações para o script se forma que o testador não precise alterar o código, isso agregaria valor para o time `` Sim acredito que facilita o processo da criação dos casos de teste ou scripts e a manutenção deles. Porque como disse na primeira pergunta é mais fácil mexer no modelo do que mexer em todos os casos de teste que derivam ele." Principal melhoria seria os modelos serem gerados de forma automatizada baseada em alguma documentação ``Talvez se os modelos fossem gerados de forma automatizada talvez baseado nos protótipos de tela o processo seria mais rápido e demandaria menos tempo facilitando o trabalho."	S5 acredita que o analista de testes que vai ficar responsável pela criação do modelo, porque em seu projeto existe um tempo livre do analista de testes, enquanto os dev estão programando a funcionalidade, neste tempo ela acredita que o testador que faria o modelo, contudo enfatiza que no projeto os cenários BDD são os desenvolvedores que escrevem `` analista de testes que vai acabar fazendo, porque enquanto os dev estão lá programando o testador que vai fazer, se bem que no meu projeto quem faz os BDDs é o dev" S5 acredita que caso no momento da execução dos testes o modelo já exista, neste caso o processo de MBT irá apresentar apenas vantagens, esta resposta foi obtida como resposta à pergunta sobre: estes de regressão, em funcionalidades onde já exista um modelo criado, você acredita que a técnica de MBT pode trazer maiores vantagens quando comparada a execução dos testes em uma nova funcionalidade que precisa ser modelada ``Sim, porque neste caso eu tenho apenas as vantagens de gerar os casos de teste, com pequenas alterações no modelo, sem que eu precise ter a desvantagem de criar tudo.'"

Figura E.5 – Categorização da Análise das Entrevista- Sujeito S5

ID	Dificuldades	Melhorias	Processo
S6	As stories não tem informação suficiente para criação de modelos ``Não tem informações suficientes ainda, no período de baixa demanda as stories tem maiores informações e mesmo assim não são suficientes." S6 Acredita que no projeto que ele trabalha atualmente não existe tempo hábil em uma sprint para execução de todo o processo, de MBT, contudo acredita que em outros projetos pode ser possível `` Na minha realidade atual não mas acredito que no scrum bem robusto, bem feitinho é possível. Temos alguns times na empresa que são cases de sucesso onde acredito que o processo conseguiria ser executado por completo"'	Uma melhoria seria gerar testes automatizados, em vez de gerar excel, exportar para linguagem de programação por exemplo C#, integrar com um request automatizado, com um ambiente de automação. (Adept).	S6 Acredita que os modelos devem ser construídos pelo PO e mantidos por toda equipe `` Acredito que o BSA porque é ela que hoje faz as stories, mas no meio, nas reuniões com a equipe eles devem ser modificados, ou seja a responsabilidade de criar deve ser de todos os envolvidos no projeto, DEV, QA" S6 acredita que em testes de regressão é mais fácil aplicar MBT por já ter um modelo inicial criado ``Regressão será mais fácil por já ter o modelo criado."

Figura E.6 – Categorização da Análise das Entrevista- Sujeito S6

APÊNDICE F – DESCRIÇÃO COMPLETA DOS CENÁRIOS

Feature: Assinatura

Scenario: Realizar Assinatura
Given<https://www.muambator.com.br/assinar/>
When I**choose**[plano]
| **plano**|
| **muambator-pro-mensal**|
And I **use-valid-data**
| **cep**| **numero**| **rua**| **bairro**| **cidade**| **estado**|
| **90050230**| **310**| **xyzk**| **centro**| **porto alegre**| **rio grando do sul**|
Then enable[finalizar]
Then showed[assinado com sucesso]

Scenario: Botão Finalizar inativo
Given<https://www.muambator.com.br/assinar/>
When I**use-valid-data**
| **nome**| **dataNascimento**| **cpf**| **cep**| **numero**| **rua**| **bairro**| **cidade**| **estado**|
| **90050230**| **310**| **xyzk**| **centro**| **porto alegre**| **rio grando do sul**|
Then disabled[finalizar]

Feature: Pacotes

Scenario: Buscar Pacote - Usuário Logado
Given { Efetuar Login}
When click[buscar]
And put[buscar]
| **buscar**|
| **JU227715430BR**|
And click[pesquisar]
Then showed[Pesquisa de pacotes por JU227715430BR Foi encontrado 1 pacote]

Scenario: Novo Pacote - Usuário Logado

Given { Efetuar Login}

When use-valid-data

codigo| **nome**| **emailAdicional**|

| **JU227715431BR**| **pacote**| **alinnezanin@gmail.com**|

And click[cadastrar]

Then showed [pacote (JU227715430BR) Entrega Efetuada]

Scenario: Novo Pacote - Sem Código

Given { Efetuar Login}

When use-valid-data

| **nome**| **emailAdicional**|

| **pacote**| **alinnezanin@gmail.com**

And dont-fill-out[codigo]

And click[cadastrar]

Then showed [preencha este campo]

Scenario: Novo Pacote - Código Inexistente

Given { Efetuar Login}

When use-valid-data

| **nome**| **emailAdicional**|

| **pacote**| **alinnezanin@gmail.com**

And put[codigo]

| **codigo**|

| **AB997715430BR**|

And click[cadastrar]

Then showed [Ops! Atualmente aceitamos somente códigos das transportadoras Correios.]

Scenario: Novo Pacote - Código Inválido

Given { Efetuar Login}

When use-valid-data

| **nome**| **emailAdicional**|

| **pacote**| **alinnezanin@gmail.com**

And put[codigo]

| **codigo**|

| **1234**|

And click[cadastrar**]**

Then showed[Aumente este texto para 10 caracteres ou mais. No momento você está usando 5 caracteres**]**

Scenario: Novo Pacote - Mais opções

Given{ Efetuar Login}

When click[mais opções**]**

And use-valid-data

| **codigo**| **nome**| **emailAdicional**| **data_previsao_entrega**| **valor**| **cep_origem**| **cep_destino**| |
JU227715431BR| **pacote**| **alinnezanin@gmail.com**| **30/05/2019**| **50,00**| **90050230**| **99150000**|

And select-data[categoria**]**

| **categoria**|

| **livro**|

And click[cadastrar**]**

Then showed [pacote (JU227715430BR) Entrega Efetuada**]**

Scenario: marcar pacote como entregue

Given { Efetuar Login}

When click[marcar como entregue**]**

Then showed [Você tem certeza que deseja marcar o pacote AB227715430BR como entregue? A partir do momento que um pacote é marcado como entregue, ele não é mais atualizado no sistema. Também é possível desfazer este comando.**]**

Scenario: confirmar marcar como entregue

Given { marcar pacote como entregue}

When click[confirmar**]**

Then showed [marcado como entregue. Ele não será mais atualizado pelo sistema.**]**

Scenario: Arquivar pacote entregue

Given { Efetuar Login}

When click-link[entregues**]**

And click[arquivar pacote**]**

Then showed [Você tem certeza que deseja arquivar o pacote**]**

Scenario: Confirmar arquivar pacote entregue

Given { Arquivar pacote entregue}

When click[confirmar**]**

Then showed [arquivado pelo sistema.**]**

Scenario: Excluir pacote entregue

Given { Efetuar Login}

When click-link[entregues**]**

And click[excluir**]**

Then showed [Você tem certeza que deseja excluir o pacote**]**

Scenario: Confirmar exclusão de pacote

Given { Excluir pacote entregue}

When click[confirmar**]**

Then showed[excluído da sua conta.**]**

Scenario: Desarquivar pacote

Given { Efetuar Login}

When click-link[arquivados**]**

When click[desarquivar**]**

Then showed [Você tem certeza que deseja desarquivar o pacote**]**

Scenario: Confirmar desarquivar pacote

Given { Desarquivar pacote}

When click[confirmar**]**

Then showed[foi desarquivado pelo sistema.**]**

Scenario: Excluir pacote arquivado

Given { Efetuar Login}

When click-link[arquivados**]**

When click[excluir**]**

Then showed [Você tem certeza que deseja excluir o pacote**]**

Scenario: Confirmar exclusão de pacote arquivado

Given { Excluir pacote arquivado}

When click[confirmar]

Then showed [excluído da sua conta.]

Feature: Login

Scenario: Efetuar Login

Given Pagina login do muambeitor <https://www.muambator.com.br/login/>

When use-valid-data

| **username**| **senha**|

| **aa@aa.com**| **12345abc**|

And click [entrar]

Then opened[https://www.muambator.com.br/pacotes/pendentes/]

Scenario: Efetuar Login - Senha Inválida

Given Pagina login do muambeitor<https://www.muambator.com.br/login/>

When put[username]

| **username**|

| **teste@aline.com**|

| **user@invalido**|

And put[password]

| **password**|

| **senhaInvalida**|

| **senhaValida**|

And click [entrar]

Then showed [Usuário ou senha inválidos!]

Scenario: Efetuar Login - ausente senha

Given Pagina login do muambeitor<https://www.muambator.com.br/login/>

When put[username]

| **username**|

| **1234@xyzj@aline.com**|

And dont-fill-out[password]

And click [entrar]

Then showed [Usuário ou senha inválidos!]

Scenario: Efetuar Login - ausente usuário

Given Pagina login do muambeitor <https://www.muambator.com.br/login/>

When dont-fill-out[username**]**

And put[password**]**

| **password**|

| **testealine1234**|

And click [entrar**]**

Then showed[Usuário ou senha inválidos!**]**

Feature: 1) Cadastro

Scenario: Criar nova conta

Given<https://www.muambator.com.br/**>**

When click[crie sua conta**]**

And use-valid-data

| **Usuario**| **email**| **nome**| **senha**| **confirme**|

| **abcd**| **testemu@gmail.com**| **teste**| **123456**| **123456**|

| **fged** | **abcde@gmail.com**| **teste**| **789456**| **789456**|

click[criarConta**]**

Then opened[https://www.muambator.com.br/perfil/registre-se/confirmacao/**]**

And showed[cadastro completo**]**

Scenario: Criar conta sem informar nome

Given<https://www.muambator.com.br/**>**

When click[crie sua conta**]**

And use-valid-data

| **username**| **email**| **senha**| **confirmacao_senha**|

| **aabbcc**| **Aaa@aa.com**| **12345abc**| **12345abc**|

And dont-fill-out [nome**]**

And click[Criar Conta**]**

Then showed[preencha este campo**]**

Scenario: Criar conta com email inválido

Given<https://www.muambator.com.br/**>**

When click[crie sua conta**]**

And use-valid-data

| **username**| **first_name**| **senha**| **confirmacao_senha**|

| **aabbcc**| **aabbcc**| **12345abc**| **12345abc**|

| **aabbcc**| **aabbcc**| **12345abc**| **12345abc**|

| **aabbcc**| **aabbcc**| **12345abc**| **12345abc**|

put[email**]**

| **email**|

| **aa@aa**|

| **aaaa.com.br**|

| **@a.com.br**|

And click [Criar Conta**]**

Then showed [Ih, acho que esse e-mail não existe**]**

Scenario: Criar conta sem informar email

Given<https://www.muambator.com.br/**>**

When click[crie sua conta**]**

And use-valid-data

| **username**| **nome**| **senha**| **confirmacao_senha**|

| **aabbcc**| **aabbcc**| **12345abc**| **12345abc**|

And dont-fill-out [email**]**

And click [Criar Conta**]**

Then showed [preencha este campo**]**

Scenario: Criar conta sem informar senha

Given<https://www.muambator.com.br/**>**

When click[crie sua conta**]**

And use-valid-data

| **username**| **email**| **first_name**| **confirmacao_senha**|

| **aabbcc**| **aa@aa.com**| **aabbcc**| **12345abc**|

And dont-fill-out [senha**]**

And click [Criar Conta**]**

Then showed [preencha este campo**]**

Scenario: Criar conta com senha e confirmação diferentes

Given<https://www.muambator.com.br/**>**

When click[crie sua conta**]**

And use-valid-data

| **username**| **email**| **first_name**|

| **aabbcc**| **aa@aa.com**| **aabbcc**|

And put [senha**]**

| **senha**|

| **123456**|

And put [confirmacao_senha**]**

| **confirmacao_senha** |

| **789654**|

And click [Criar Conta**]**

Then showed [Sua confirmação de senha não está correta**]**

Scenario: Link Termo de Uso

Given<https://www.muambator.com.br/**>**

When click[crie sua conta**]**

And click-link[Termos de Uso**]**

opened[opened**]**

And showed-title[Termos de Uso e Privacidade - Muambator**]**

Feature: Oráculos

Scenario: Acesso oráculo Gringo

Given { efetuarlogin**)**

Whe i mouse-over[oraculo**]**

And click[oraculoGringo**]**

them showed-title[Oráculo Gringo**]**

And opened[https://www.muambator.com.br/oraculo/gringo/**]**

Scenario: Busca em oráculo Gringo

Given<https://www.muambator.com.br/oraculo/gringo/**>**

When I select-data[pais**]**

| **pais**|

| **italia**|

And put[cepDestino**]**

| **cepDestino**|

| **90050230**|

Then showed[Media**]**

And showed[34,50**]**

Scenario: Acesso oráculo tupiniquim

Given { efetuarlogin)

When I mouse-over[oraculo**]**

And click[oraculo tupiniquim**]**

Them showed-title[Oráculo tupiniquim**]**

And Opened[https://www.muambator.com.br/oraculo/tupiniquim/**]**

Scenario: Busca em Oráculo tupiniquim

Given { Acesso oráculo tupiniquim}

When put[cep_origem**]**

| **cep_origem**|

| **90050230**|

| **90619900**|

And put[cep_destino**]**

| **cep_destino**|

| **99150000**|

Then showed[Marau/BR**]**

And showed[Porto Alegre/BR**]**

APÊNDICE G – MODELOS

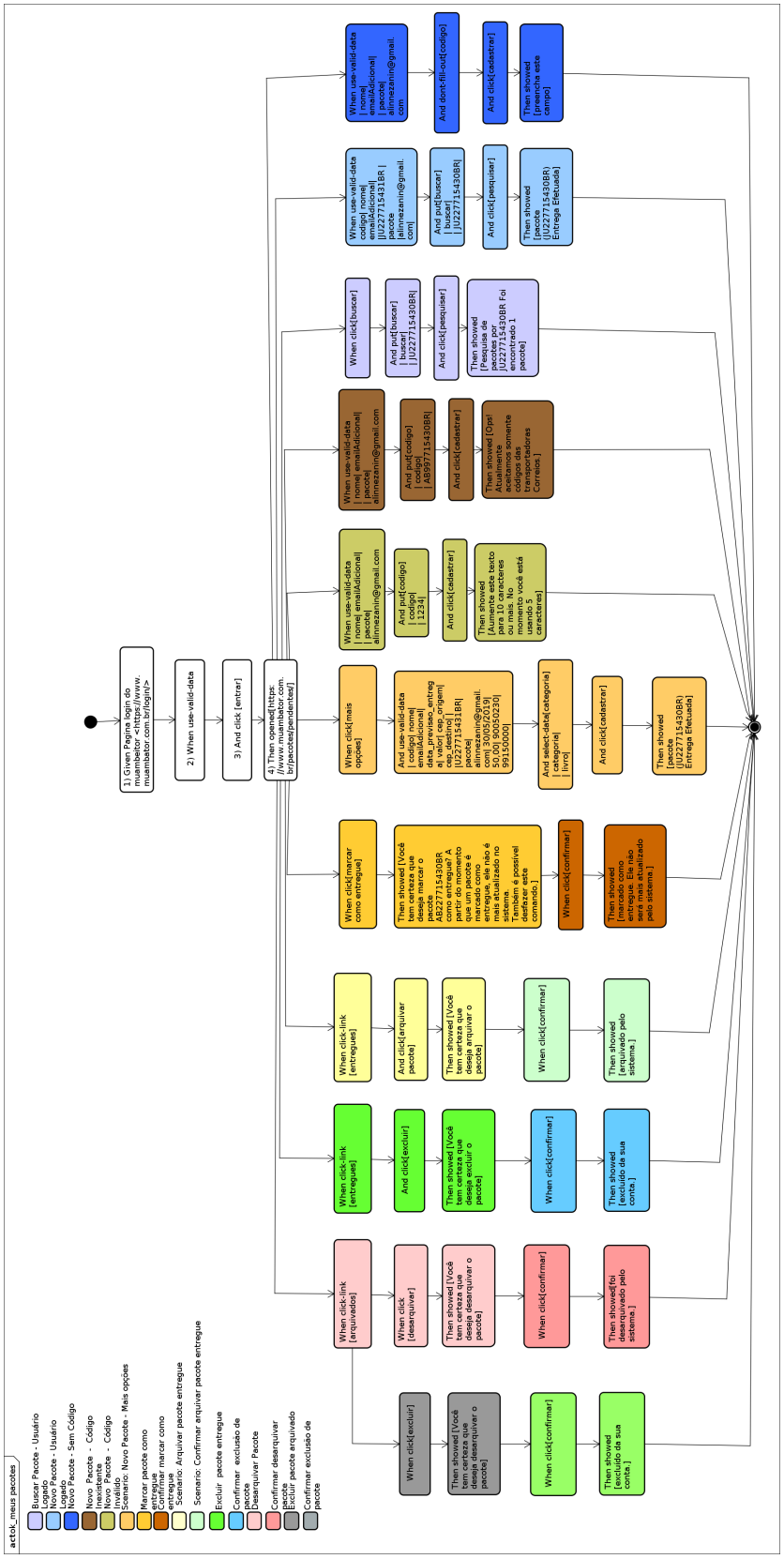

Figura G.1 – meus pacotes

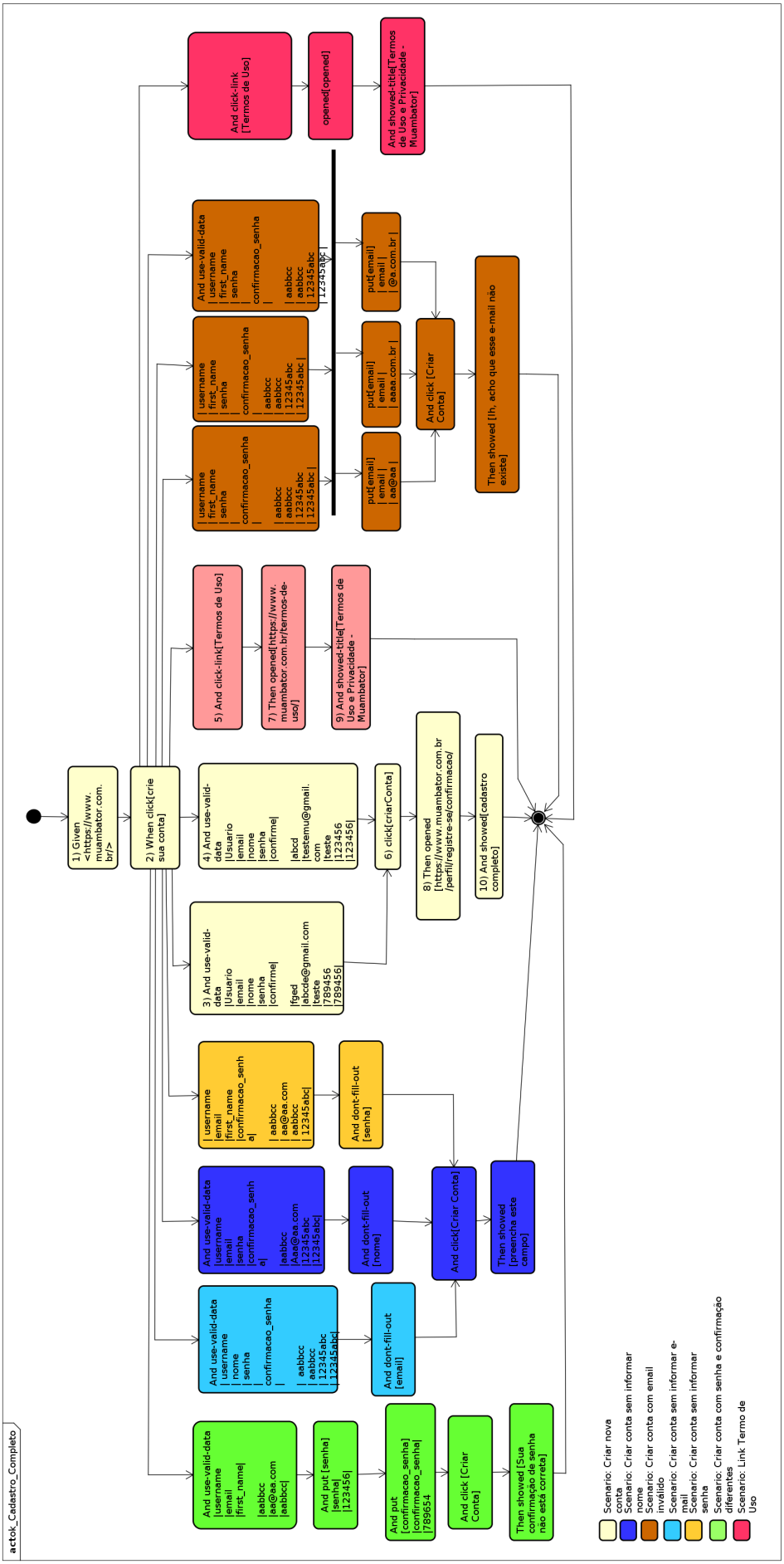

Figura G.2 – Cadastro

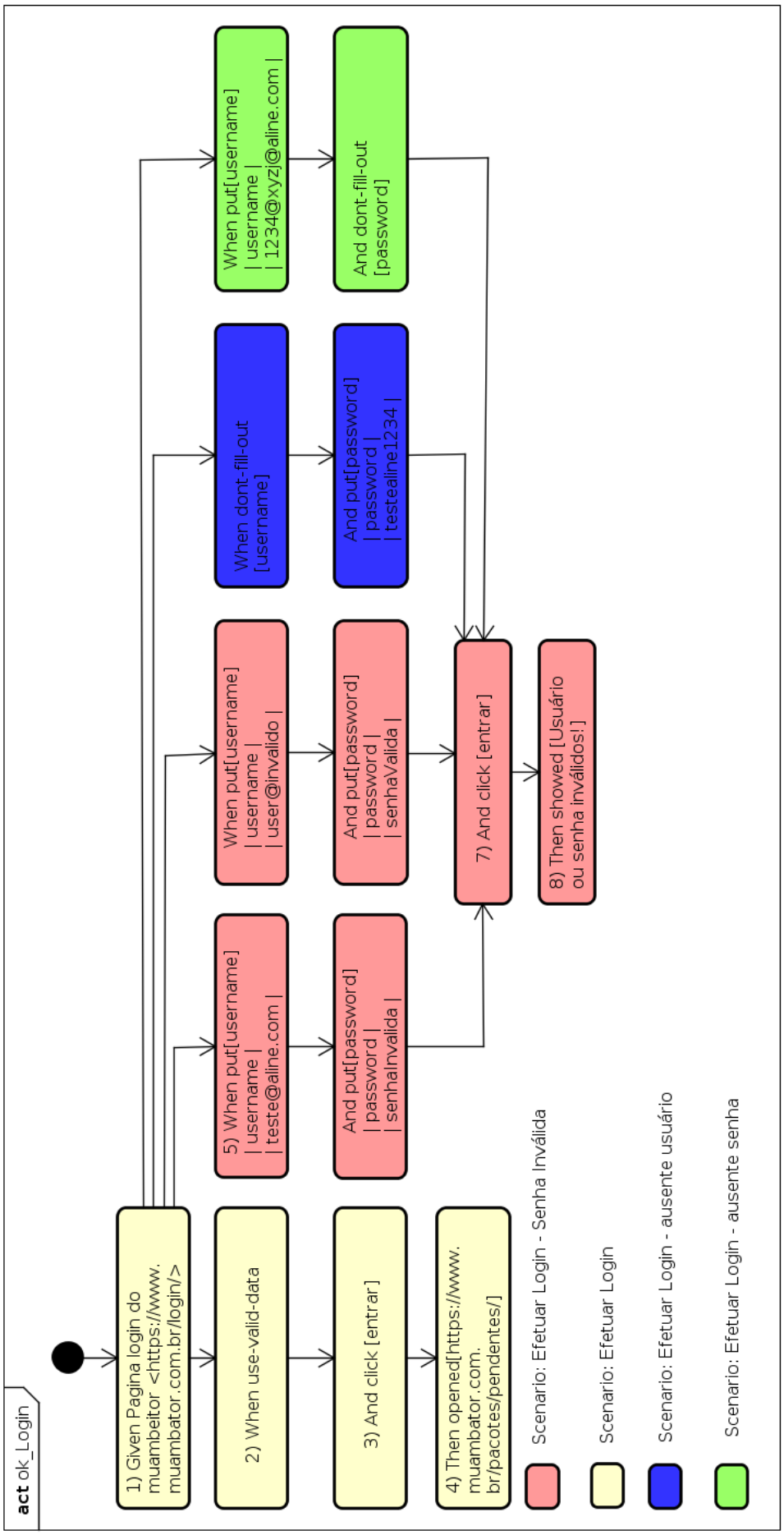

Figura G.3 – Login

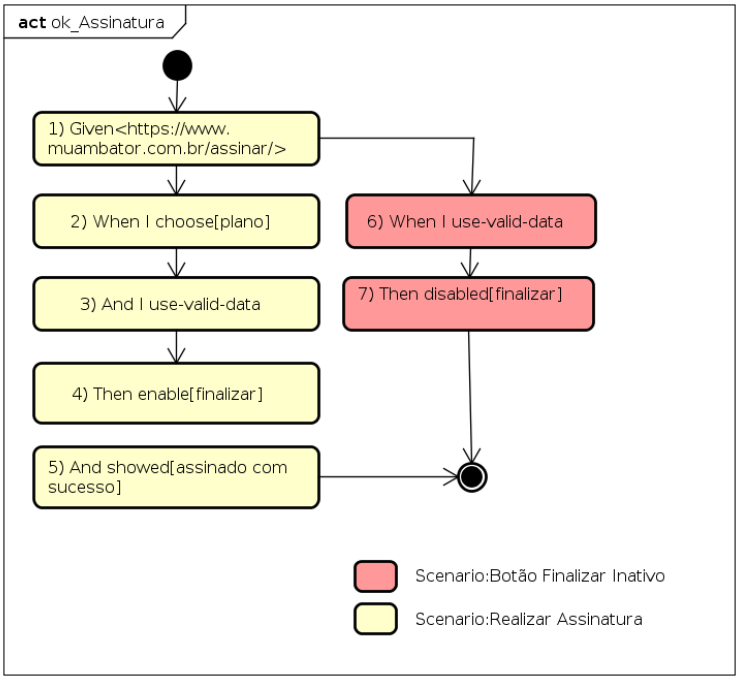

Figura G.4 – Assinatura

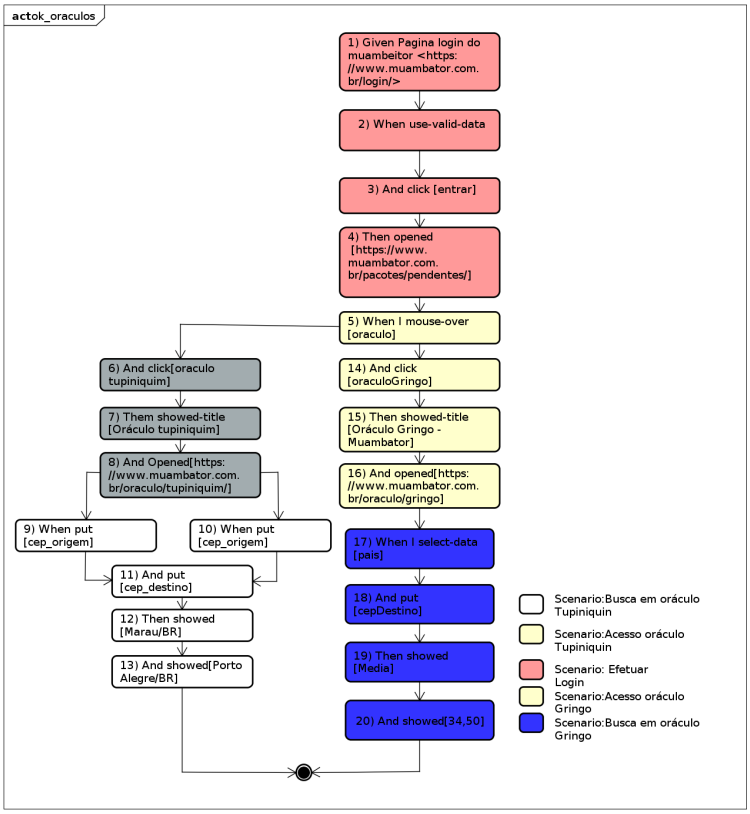

Figura G.5 – Oráculo

Pontifícia Universidade Católica do Rio Grande do Sul
Pró-Reitoria de Graduação
Av. Ipiranga, 6681 - Prédio 1 - 3º. andar
Porto Alegre - RS - Brasil
Fone: (51) 3320-3500 - Fax: (51) 3339-1564
E-mail: prograd@pucrs.br
Site: www.pucrs.br