Utilização do Algoritmo de QSM para automação de testes a partir de cenários Gherkin

Aline Zanin, Henry C. Nunes, Avelino F. Zorzo, Flavio M. de Oliveira

[1]Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Av. Ipiranga, 6681 - Prédio 32 - Porto Alegre - RS - Brasil

aline.zanin@acad.pucrs.br, henry.nunes@outlook.com,

avelino.zorzo@pucrs.br, flavio.mo.oliveira@gmail.com

Abstract. *One of the present challenges in software development is the representation of requirements in a clear and intuitive way and at the same time make it productively viable in software development and testing - which imply in intensive use of automation. To ease this work, various techniques are utilized, as an example, the requirements writing in languages as Gherkin and the creation of formal models. In this work, we aim to unify the advantages offered by both techniques through the proposal of a model for automated generation of test scripts. This model includes the generation of Finite State Machines (FSM), with Gherkin scenarios, using the Query-driven State Merging (QSM) algorithm. Using the resulting FSM, we can use Model-based Testing (MBT) techniques for generation of a more robust set of test cases, which can be used to generate test scripts, in an automatic way too. In this work, we performed a literature review and verified the proposed model through an example of use. In this example, we perceived the viability of the model and the contribution for academy and industry.*

Resumo. *Um dos desafios presentes no desenvolvimento de software é a representação de requisitos de forma clara e intuitiva, por um lado, e ao mesmo tempo viabilizar produtividade no desenvolvimento e testes de sistemas - o que implica o uso intensivo de automação. Para facilitar este trabalho, diversas técnicas são utilizadas, por exemplo, a escrita de requisitos em linguagens como Gherkin e a criação de modelos formais. Neste trabalho, buscamos unir as vantagens oferecidas por ambas as técnicas por meio da proposta de um modelo para geração automatizada de scripts de teste. Este modelo contempla a geração de Máquinas de Estados Finitos (MEFs), a partir de cenários Gherkin, utilizando o algoritmo Query-driven State Merging (QSM). A partir da MEF gerada, utilizamos técnicas de Teste Baseado em Modelos (Model-based Testing - MBT) para gerar um conjunto robusto de casos de teste, a partir dos quais geram-se scripts de teste, de forma também automática. Para o desenvolvimento deste trabalho, nos baseamos em uma revisão de literatura, e posteriormente validamos o modelo proposto por meio de um exemplo de uso. Neste exemplo de uso percebe-se a viabilidade da utilização deste modelo e a contribuição para academia e para indústria.*

1. Introdução

O desenvolvimento de software é um trabalho complexo que envolve diversas habilidades e práticas. Este trabalho é costumeiramente realizado por mais de uma pessoa, formando

equipes de desenvolvimento. Nestas equipes, todos os envolvidos precisam comunicar-se de forma eficaz, para que possa ser realizado um trabalho colaborativo e atinja-se êxito nos projetos de desenvolvimento de sistemas. Além disso, a comunicação dos membros da equipe com o cliente é de extrema importância, uma vez que software é desenvolvido visando atender alguma necessidade de algum segmento ou cliente específico.

Desta forma, são necessários recursos para que o profissional responsável por elicitar os requisitos do software junto ao cliente, possa representá-los de forma clara e intuitiva. Isto porque é necessário que o mesmo requisito, utilizado pelas equipes de desenvolvimento para guiar seu trabalho, seja visível e compreensível pelo cliente. Desta forma, o cliente pode verificar se o software que está sendo produzido é o software desejado.

Neste contexto diversas são as técnicas utilizadas para facilitar a representação de requisitos. Uma delas é a utilização de descrição em linguagem semi-natural por meio da Linguagem de Domínio Específico (DSL - Domain-specific Language) Gherkin. Esta representação tem sua escrita facilitada, dada a simplicidade e a proximidade com o idioma comumente utilizado. Esta prática é consolidada especialmente em equipes ágeis, por meio da técnica de Desenvolvimento Guiado por Comportamento (*Behavior-driven Development -BDD*). Esta técnica permite a utilização de uma documentação viva que estrutura o desenvolvimento do sistema de acordo com os requisitos escritos. Contudo, esta técnica não efetua a geração automatizada de *scripts* de teste com alta cobertura e não propicia ao cliente e aos desenvolvedores, uma visão gráfica do fluxo do sistema. Em geral, os testes definidos a partir de especificações em Gherkin limitam-se a testes de aceitação.

Estas duas características não compreendidas pela técnica de BDD, podem ser supridas pela aplicação da técnica de Teste Baseado em Modelos (*Model-based Testing – MBT*). Esta técnica consiste na geração de artefatos de teste a partir de modelos comportamentais, geralmente modelos de transição de estados. Em particular, no contexto deste estudo, trabalhamos com Máquinas de Estados Finitos (MEFs). Para que esta técnica possa ser aplicada, os requisitos do sistema precisam ser descritos em modelos comportamentais. Desta forma, além de propiciar a geração de artefatos de teste, a técnica propicia ao cliente e à equipe de desenvolvimento uma visão mais robusta de como será o fluxo comportamental do sistema que está sendo desenvolvido.

Entretanto, a técnica de MBT possui algumas desvantagens que em determinados contextos de projetos dificultam a sua aplicação. Uma delas é o tamanho dos modelos criados. Isto porque, para funcionalidades complexas, o modelo que precisa ser criado para representá-las se torna extenso e sua compreensão é dificultada. Além disso, para uma equipe que já utiliza Gherkin em um contexto de BDD, construir uma MEF equivale a um retrabalho e consequente aumento no custo do projeto.

Neste contexto, este trabalho propõe uma estratégia para utilização conjunta da DSL Gherkin, e de MBT. Esta estratégia permitirá uma melhor representação dos requisitos do sistema e a geração automatizada de artefatos de teste. Buscando contribuir com a redução do problema supracitado de geração de modelos demasiadamente complexos, propomos a geração automática de MEFs a partir de especificações Gherkin. Utilizamos o algoritmo *Query-driven State Merging – QSM* [Dupont et al. 2008] que propõe a indução de máquinas de estados finitos de forma que representem adequadamente a funcionalidade que está sendo modelada e testada, porém de forma otimizada reduzindo a

quantidade de estados da MEF, evitando retrabalho e, posteriormente, gerando artefatos de teste com alta cobertura.

Assim, este artigo se propõe a responder a seguinte questão de pesquisa: Aplicar o algoritmo de QSM, em MEF geradas a partir de requisitos escritos no formato Gherkin, pode otimizar os artefatos de testes gerados pela técnica de MBT? Para responder a esta pergunta, realizamos primeiramente uma revisão de literatura buscando identificar trabalhos relacionados e que pudessem contribuir com o modelo proposto. Posteriormente, efetuou-se a criação de um *parser* para geração de MEF a partir de Gherkin. Este *parser* baseou-se na proposta central de Gherkin, onde informações representadas da cláusula *when* representam ações, e valores representados na cláusula *then* representam resultados. A aplicação do algoritmo de QSM, por sua vez, baseou-se no proposto por [Dupont et al. 2008] e a geração de sequência de testes no método de geração de sequência de testes Wp [Fujiwara et al. 1991]. Por fim as sequências geradas pelo método Wp são convertidas em *drivers* de teste.

Este trabalho está organizado da seguinte forma: a Seção 2 apresenta a fundamentação teórica que embasa este trabalho; a Seção 3 apresenta a estratégia proposto por este trabalho; a Seção 4 apresenta um exemplo de uso da estratégia e a Seção 5 apresenta as considerações finais e trabalhos futuros.

2. Fundamentação Teórica

Teste Baseado em Modelos *(Model-based Testing (MBT))* [El-Far and Whittaker 2002] é uma técnica de testes que consiste na utilização de modelos para a geração de artefatos de testes. MBT pode ser aplicado para todos os tipos de testes, sendo que, conforme o tipo de testes que será empregado, deve-se definir qual o melhor modelo para utilizar. Para a realização de testes funcionais, que é o foco deste trabalho, são exemplos de modelos que podem ser utilizados: diagrama de atividades da UML, diagrama de casos de uso da UML e especificações formais como as Máquinas de Estados Finitos (MEF).

Para todas as formas de utilização, MBT permite reduzir custos com geração de *scripts* de teste e correção de defeitos, permitindo ainda criar uma rastreabilidade entre modelo e requisito, de forma que, quando ocorre uma alteração de requisito de sistema, se torna possível recriar os *scripts* de teste de forma automatizada, mitigando o problema gerado pelo alto custo de manutenção dos artefatos de teste.

Neste mesmo contexto de buscar automatizar os processos de criar melhor rastreabilidade entre requisito e produto, diversas equipes tem utilizado a escrita de requisitos utilizando a linguagem de domínio específico (*Domain-specific Language - DSL*) Gherkin [Wynne et al. 2017]. Gherkin permite que os requisitos sejam escritos utilizando linguagem semi-natural, aproximando-se da linguagem do usuário e fazendo uso de palavras chaves pré-definidas que auxiliam na descrição do comportamento esperado para o sistema, sendo elas: GIVEN (representando o estado anterior do sistema), WHEN (representando uma ação que será executada), THEN (representando o resultado esperado). Esta DSL, em conjunto com ferramentas que dão suporte a sua utilização *e.g.* Cucumber permite a aplicação da técnica de *Behavior-driven Development (BDD)*, que proporciona a rastreabilidade dos requisitos com o código implementado.

Contudo, embora Gherkin e MBT sejam ferramentas que tem em comum, realizar automação de alguns processos e melhorar a rastreabilidade dos requisitos, elas costumeiramente não são aplicadas em conjunto. Neste sentido o algoritmo *Query-driven*

State Merging (QSM) pode ser uma importante ferramenta para o relacionamento destas duas técnicas. O algoritmo de QSM é um algoritmo de indução que tem por objetivo gerar MEFs a partir de um modelo de entrada. Neste trabalho, o modelo de entrada é uma especificação em Gherkin.

Para a geração das MEFs, o algoritmo considera um processo de indução onde inicialmente são inseridos na MEF os cenários positivos e posteriormente os cenários negativos. Para cada cenário negativo inserido, o algoritmo considera um processo de indução onde são verificadas e excluídas redundâncias de estados. O resultado é uma MEF que mapeia os comportamentos descritos na especificação de entrada.

2.1. Trabalhos Relacionados

A fim de complementar os nossos estudos buscamos trabalhos diretamente relacionados, contudo, não localizamos trabalhos que apresentem utilização de Gherkin e QSM e por este motivo relacionamos trabalhos que apresentam Gherkin ou QSM e que explorem soluções para realização de testes de software.

Seijas *et al.* [Seijas et al. 2016] no trabalho *Model extraction and test generation from JUnit test suites*, descreve uma técnica para realizar inferência em MEFs que modelam testes unitários de sistemas legados e analisa como gerar novos testes a partir destes modelos. A técnica é implementada utilizando a ferramenta James Tool que aplica a técnica em códigos Java e códigos de teste.

Sivanandan *et al.* (2014) [Sivanandan and B 2014], apresentam o trabalho *"Agile development cycle: Approach to design an effective Model-based Testing with Behaviour-driven automation framework"*. Este trabalho apresenta uma estratégia para geração de cenários no formato Gherkin a partir de modelos. De forma similar o trabalho "Sky-fire: Model-based Testing With Cucumber" proposto por Li *et al.* (2016) [Li et al. 2016] também gera cenários Gherkin a partir de modelos, neste caso diagramas da UML e não especificação formal em MEFs.

Entin *et al.* (2015) [Entin et al. 2015] apresentam o trabalho "*A process to increase the model quality in the context of model-based testing"*, neste trabalho os autores apresentam a criação automatizada de modelo com base nos requisitos escritos na linguagem Gherkin. O trabalho apresentado pelos autores se assemelha com o trabalho que apresentamos neste estudo por gerar *script* a partir de cenários Gherkin, contudo, este trabalho não utiliza o algoritmo de QSM para otimizar os artefatos gerados.

3. Proposta de modelo para aplicação de MBT a partir de cenários Gherkin

O algoritmo QSM, conforme mencionado anteriormente, foi originalmente proposto para efetuar a transição de diagramas de atividade para máquinas de estados. Nesta seção demonstramos como realizamos a migração de cenários Gherkin para MEF.

3.1. Criação de cenários Gherkin

Nesta etapa aplica-se a criação dos cenários no formato Gherkin em sua estrutura tradicional, conforme a interpretada pela ferramenta Cucumber [Cucumber 2017], não sendo previsto por este trabalho nenhuma alteração nesta etapa. Nesta representação, os requisitos do sistema são separados por funcionalidades (*features*) e por cenários (*scenarios*). Uma *feature* representa a funcionalidade que está sendo desenvolvida e cada cenário descreve um comportamento esperado para aquela funcionalidade. Um cenário sempre possui um título e é formado pelas cláusulas GIVEN, WHEN e THEN, podendo ser acrescido

de cláusulas AND. GIVEN representa o estado atual do sistema no momento em que o cenário é executado, WHEN representa uma ação que será feita no sistema e está sendo descrita neste cenário e THEN representa o resultado do sistema para esta ação.

3.2. Geração de Sequências e criação de MEFs

Para cada *feature*, que contempla um conjunto de cenários, é criada uma MEF simulando um fluxograma em formato de árvore. O início da *feature* representa o estado inicial da MEF e a partir dele, cada um dos cenários Gherkin derivará um ramo da árvore. Para este fim, são criadas sequências que representam a ordem em que os comportamentos descritos em cada cenário serão executados. No momento da geração da MEF, cada item da sequência irá ser inserido na transição anterior a um estado e representar a entrada de um estado, sendo que, os estados são criados numerados em ordem crescente. A transição de saída de cada estado irá receber como valor a confirmação da entrada, isto porque, entradas representam ações e saídas, resultados esperados. Dado que todas as sequências criadas a partir do Gherkin são positivas, podemos considerar o resultado esperado para cada item delas como sendo a confirmação da ação (por exemplo: entrada abrir, saída aberto).

3.3. Eliminação de Não Determinismos

Em algumas situações podem ser geradas MEF com situações de não determinismo, ou seja, situações em que uma mesma entrada leva a dois ou mais estados distintos. Desta forma para resolver este problema são unidos os estados distintos em um único estado e as transições são removidas.

3.4. Redução da MEF por meio de induções

Esta etapa tem por objetivo remover redundâncias na MEF e com isso otimizar os artefatos de testes produzidos posteriormente. A redução da MEF por meio de induções é realizada por meio de múltiplas tentativas de união de estados. O processo de indução inicia sempre pelo estado de menor valor hierárquico sendo este estado comparado a todos os outros estados da MEF verificando as uniões possíveis.

A fim de verificar se uma união é possível, são geradas sequências contendo o prefixo do estado de menor hierarquia, e todos os sufixos do estado que está sendo comparado com este. Estas sequências representam fluxos do sistema modelado na MEF e são chamadas de *Queries*. Após gerar as *Queries* o algoritmo gera um conjunto de perguntas ao usuário sobre a aplicabilidade destas *Queries*. Caso o usuário considere positivas todas as *Queries*, ou seja, considere que com a união dos estados da MEF os fluxos resultantes são válidos, os estados são unidos, senão a MEF permanece inalterada.

3.5. Geração de Sequências de Teste com o método Wp

Um vez gerada a MEF ela é submetida como entrada para a aplicação do método Wp. O método Wp é um método de geração de sequências de testes, proposto por Fujiwara(1991) [Fujiwara et al. 1991], que tem por objetivo gerar sequências (caminhos) capazes de distinguir os estados da MEF por meio de seu comportamento. Para MEFs determinísticas e totalmente especificadas, o método garante a detecção de defeitos de transferência (transição para estado incorreto) e defeitos de saída (saída incorreta em uma transição). Desa forma, o método garante uma alta cobertura de toda a funcionalidade em teste. A partir destas sequências podem ser produzidos artefatos de execução de teste, como por exemplo *drivers* JUnit.

3.6. Geração de *Scripts* de Teste

A partir das sequências geradas pelo método Wp, produziu-se um *parser* para a criação do *driver* de teste, fazendo uso da biblioteca JUnit. Para cada uma das sequências foi gerado um método de teste e para cada item da sequência utilizou-se o comando *assertEquals*. Neste *assert* busca-se verificar se a resposta recebida é a confirmação da entrada enviada.

4. Exemplo de Uso

Para demonstração da aplicação da estratégia proposta foi utilizado um software didático que simula a funcionalidade de um estojo de canetas. Este estojo tem capacidade de armazenar uma única caneta e possui os estados aberto e fechado. O estojo está inicialmente sempre fechado e vazio, sendo que, para efetuar qualquer operação o estojo precisa ser aberto.

Abrir: Consiste na operação de abrir o estojo. Esta operação é pré-requisito para as operações de Adicionar e Remover que serão descritas na sequência. Desta forma só podem ser acionadas as funções de Adicionar e Remover quando o estojo estiver aberto; além disso, uma vez aberto o estojo ele não poderá ser aberto novamente sem antes ter sido fechado.

Fechar: Consiste na operação de fechar o estojo, podendo ser executada apenas quando o estojo está aberto. Estando fechado não é possível adicionar e nem remover uma caneta no estojo.

Adicionar: Adiciona uma caneta no estojo (a capacidade do estojo é de um único item). Após adicionada uma caneta, ela pode ser removido com a operação remover.

Remover: Remove um item no interior do estojo; como efeito, o estojo está vazio, podendo ser adicionada uma caneta.

Vazio: Operação que retorna verdadeiro caso o estojo esteja vazio.

4.1. Cenários Gherkin

O primeiro passo para a aplicação do modelo proposto neste trabalho é a escrita dos requisitos em formato de cenários Gherkin. Para o software porta-canetas, foram criados quatro cenários Gherkin, sendo eles Colocar Caneta (colocandoCaneta), Colocar e Retirar Caneta (colocaRetira), Abrir e Fechar Estojo (abreFecha) e Retira Caneta (retiraCaneta) os quais podem ser visualizados a seguir na *feature* EstojoCaneta:

```
Feature: EstojoCaneta
  Scenario: colocandoCaneta
    Given Abrir
    When Depositar
    And Fechar
    Then NaoVazio
  Scenario: colocaRetira
    Given Abrir
    When Depositar
    When Remover
    When Fechar
    Then Vazio
  Scenario: abreFecha
```

```
  Given Abrir
  Then Fechar
Scenario: retiraCaneta
  Given Vazio
  When Abrir
  And Depositar
  And Remover
  Then Vazio
```

4.2. Geração de Sequências e criação de MEF

A partir dos cenários descritos utilizando Gherkin, são criadas sequências que representam os comportamentos descritos no cenário. Para os cenários listados acima são criadas quatro sequências, sendo elas: *<Abrir, Depositar, Fechar, NaoVazio>*, *<Abrir, Depositar, Remover, Fechar, Vazio>*, *<Abrir, Fechar>*, *<Vazio, Abrir, Depositar, Remover, Vazio>*.

A partir destas sequências criou-se uma MEF, sendo que foi criado um estado inicial para a máquina e a partir dele foram derivados ramos em uma estrutura de árvore. Cada ramo da árvore formada consiste em umas das sequências positivas recebidas, como pode ser visto na Figura 1.

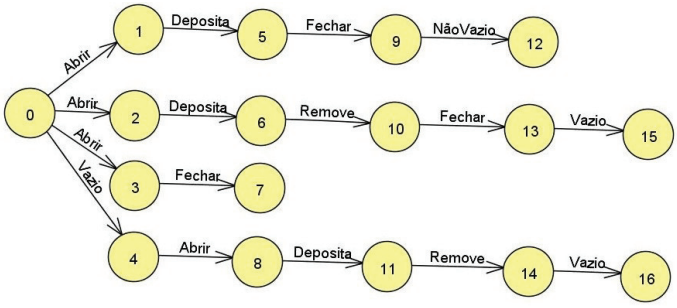

Figura 1. Primeira MEF gerada

4.3. Eliminação de Não Determinismos

A criação da máquina de estados dessa forma pode resultar em casos de não determinismo, ou seja, possuir estados onde um mesmo valor de entrada[1] leva a máquina para diferentes estados. Um exemplo pode ser observado no estado 0, que possui transições para os estados 1, 2 e 3 utilizando a mesma entrada do alfabeto, a entrada *Abrir*.

A fim de solucionar este problema, estas transições são removidas, fazendo a união dos estados que tem a mesma origem e, quando submetidas a mesma entrada, levam para estados diferentes. Caso esta união crie novos casos de não-determinismo o processo é repetido. Em nosso exemplo os estados 1, 2 e 3 são unidos em um único estado. Como regra, em toda união de estados é preservado o valor na hierarquia do menor número. Portanto eles são fundidos no estado 1. Como resultado passou a existir um novo não determinismo do estado 1 para o estado 5 e 6 quando a entrada do alfabeto é *Depositar*. Deste modo, o processo de unir os estados acontece novamente agrupando os estados no estado 5. Como resultado final se tem a máquina de estados que está representada na Figura 2.

[1]Considere entrada o valor que é descrito na transição antes

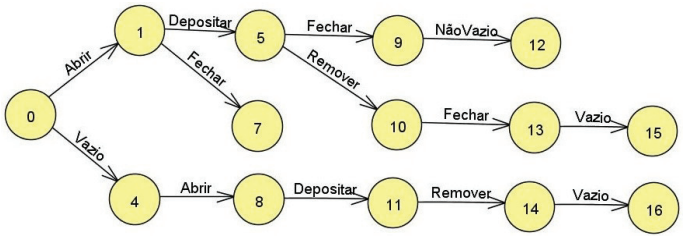

Figura 2. MEF Determinística

4.4. Redução da MEF por meio de induções

Para a MEF da Figura 2, o estado de menor hierarquia é o estado 0 e o de maior hierarquia o estado 16. Por este motivo o processo de indução começa comparando o estado 0 a todos os demais estados da MEF. A fim de verificar se uma união é possível, são geradas sequências contendo o prefixo do estado de menor hierarquia, e todos os sufixos do estado que está sendo comparado com este. Por exemplo: Para a MEF da Figura 2, a tentativa de união do estado 1 com o estado 5 resultaria nas seguintes *Queries*: *<Abrir, Fechar, NaoVazio>, <Abrir, Remover, Fechar, Vazio>*, isto porque o prefixo do estado 1 é ¡abrir¿ e os sufixos do estado 5 são: *<¡fechar, NaoVazio¿>* e *<¡Remover,fechar, vazio¿>*. Após gerar as *Queries* o algoritmo de QSM gera uma pergunta ao usuário sobre a sua validade. Caso o usuário considere positivas todas as *Queris*, ou seja, considere que com a união dos estados da MEF os fluxos resultantes são válidos, os estados são unidos, senão a MEF permanece inalterada.

Ao concluir as induções do estado de menor hierarquia com todos os demais estados da MEF, o algoritmo elege um novo estado como sendo o de menor hierarquia, considerando sempre o imediatamente posterior na ordem hierárquica e todo o processo de indução é repetido. Assim sucessivamente para todos os estados da MEF. Desta forma para a MEF da Figura 2 seria executada a seguinte ordem de tentativas:*<(0,1), (0,2),(0,3)...(0,16),(1,2),(1,3)...(1,16)(2,3)(2,4)...>*.

Após realizadas todas as uniões e gerada a MEF final, para ser possível aplicar métodos de geração de sequências de testes, mais especificamente o método Wp, foi necessário adicionar as informações de saída *output* para a MEF. Conforme supracitado a saída da MEF é uma confirmação da entrada e foi inserida de forma manual. Na Figura 3 é possível visualizar a MEF final que foi resultado de todas as uniões de estados efetuadas contendo as saídas.

Figura 3. Resultado Output

4.5. Aplicação do Método Wp

A aplicação do método Wp [Fujiwara et al. 1991] resultou nas seguintes sequências de testes: **Sequência 1 :** [Vazio, Abrir]; **Sequência 2 :** [Abrir, Vazio, Abrir]; **Sequência 3 :**

[Abrir, Vazio, Vazio]; **Sequência 4 :** [Abrir, Fechar, Abrir]; **Sequência 5 :** [Abrir, Depositar, Remover, Abrir];**Sequência 6 :** [Abrir, Depositar, Remover, Vazio];**Sequência 7 :** [Abrir, Depositar, Fechar, NaoVazio, Abrir] ;**Sequência 8 :** [Abrir, Depositar, Fechar, NaoVazio, Vazio] ;**Sequência 9 :** [Abrir, Depositar, Fechar, NaoVazio, Fechar]. Importante notar que para aplicação do método é necessário a MEF completamente especificada. Isso pode ser alcançado gerando transições vazias para as transições inexistentes.

4.6. Geração de *script* JUnit

Utilizando as sequências geradas e as saídas da MEF criada anteriormente, foi possível criar um *parser* para gerar os *drivers* de teste. Para isto, foi utilizada a biblioteca JUnit. Para cada entrada em uma sequência de testes, foi criado um *assertEquals* comparando se o resultado da ação é igual à saída especificada na MEF. Abaixo apresentamos um exemplo de *Script* para a sequência 5: [Abrir, Depositar, Remover, Abrir]. As demais sequências podem ser acessadas em: https://goo.gl/mDT1Ee.

```
@Test
public static void _Test_Sequence_5(){
assertEquals("Aberto", Abrir());
assertEquals("Depositado", Depositar());
assertEquals("Removido", Remover());
assertEquals("Aberto", Abrir());}
```

5. Considerações finais e Trabalhos Futuros

Neste trabalho apresentamos uma estratégia para a geração de *scripts* de testes a partir de cenários Gherkin. A estratégia utiliza o algoritmo QSM em conjunto com o método de geração de sequências de testes Wp para aplicação da técnica de MBT. Este trabalho está em fase inicial, sendo desenvolvido em parceria com uma empresa e estando em fase de testes com cenários maiores do que o utilizado como exemplo. Desta forma, considera-se como principal ameaça à validade dos resultados o fato do teste ter sido executado com uma *feature* muito simples, o que pode não apresentar os mesmos resultados do que em uma aplicação real. Além disso, também identificamos duas limitações na estratégia, sendo elas a inserção manual das saídas (*outputs*) na MEF e a alta participação do usuário respondendo as perguntas de validação das *Queries*. Ambas as situações pontuadas são consideradas limitações por envolverem trabalho repetitivo do usuário, o que consome tempo e pode prejudicar a aplicabilidade da proposta.

Projeta-se como trabalhos futuros para este estudo, a automação da geração de saídas (*outputs*) para a MEF, bem como a otimização do método para reduzir ou eliminar as perguntas feitas para o usuário na aceitação das *Queries*. Além disso, está em fase de estruturação a realização de um estudo de caso com uma aplicação real.

6. Agradecimentos

Agradecemos a CAPES pelo financiamento dos estudos através da bolsa de doutorado de Aline Zanin e a empresa DBServer pelo financiamento parcial deste trabalho.

Referências

Cucumber (2017). Gherkin. Disponível em: https://cucumber.io/docs/reference Acesso em 08 de setembro de 2017.

Dupont, P., Lambeau, B., Damas, C., and Lamsweerde, A. v. (2008). The qsm algorithm and its application to software behavior model induction. *Applied Artificial Intelligence*, 22(1-2):77–115.

El-Far, I. K. and Whittaker, J. A. (2002). Model-based software testing. *Encyclopedia of Software Engineering*.

Entin, V., Winder, M., Zhang, B., and Claus, A. (2015). A process to increase 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*.

Fujiwara, S., Khendek, F., Amalou, M., Ghedamsi, A., et al. (1991). Test selection based on finite state models. *IEEE Transactions on Software Engineering*, 17(6):591–603.

Li, N., Escalona, A., and Kamal, T. (2016). Skyfire: Model-based testing with cucumber. In *Proceedings of the IEEE International Conference on Software Testing, Verification and Validation*, pages 393–400.

Seijas, P. L., Thompson, S., and Francisco, M. Á. (2016). Model extraction and test generation from junit test suites. In *Proceedings of the 11th International Workshop on Automation of Software Test*, pages 8–14. ACM.

Sivanandan, S. and B, Y. C. (2014). Agile development cycle: Approach to design an effective model based testing with behaviour driven automation framework. In *Proceedings of the 20th Annual International Conference on Advanced Computing and Communications*.

Wynne, M., Hellesoy, A., and Tooke, S. (2017). *The Cucumber Book: Behaviour-Driven Development for Testers and Developers*. Pragmatic Programmers. Pragmatic Bookshelf.