Gerando código para melhoria do Teste Funcional Automatizado de ferramentas de Gerenciamento de Processos de Negócio

Jéssica Lasch de Moura[1], **Andrea Schwertner Charão**[1]

[1]Centro de Tecnologia – Universidade Federal de Santa Maria (UFSM)
Santa Maria – RS – Brazil

[2]Laboratório de Sistemas de Computação (LSC) – Universidade Federal de Santa Maria

{jmoura,andrea}@inf.ufsm.br

Abstract. *This paper describes a strategy created to get artifacts used in the implementation of automated functional testing of Business Process Management tools. The objective of this work was to facilitate the tests and improve the scope of the same, in order to improve the applications in question. With this strategy, it was possible to obtain important information about the processes and generate important elements for functional testing, making it the fastest and most complete test case.*

Resumo. *Este trabalho descreve um estratégia criada para obter artefatos utilizados na execução de testes funcionais automatizados de ferramentas de Gestão de Processos de Negócio. O objetivo deste trabalho era facilitar a execução dos testes bem como melhorar a abrangência dos mesmos, visando a melhoria das aplicações em questão. Com esta estratégia, foi possível obter informações importantes sobre os processos e gerar elementos importantes para o teste funcional, tornando a etapa de teste mais rápida e completa.*

1. Introdução

- BPM
- Teste com BPM pouco abordado + importância dos testes;
- Citar trabalho anterior (abordagem utilizadas para ferramentas web em geral)= teste com selenium e cucumber se mostrou mais promissor;
  - Pode ser trabalhoso criar os elementos necessários para o teste completo;
- Para melhoria na criação dos testes: analisar o diagrama no formato BPMN, através do parser criado, para gerar informações
  - Facilitar/tornar mais rápida a criação dos elementos
  - Aumento na cobertura dos testes (cria todos cenários possíveis)
- Objetivo: melhorar e facilitar o teste funcional automatizado de aplicações bpm;

2. Business Process Management - *BPM*

2.1. Business Process Model and Notation - *BPMN*

O padrão Business Process Model and Notation, ou BPMN, foi criado com o objetivo de fornecer uma notação facilmente compreensível por todos os usuários, desde os analistas

que criam os rascunhos iniciais dos processos até os desenvolvedores responsáveis por implementar os processos e, finalmente, para os usuários que irão gerenciar e monitorar esses processos[Model 2011]. A versão BPMN 2.0, a mais recente, define um padrão XML para arquivos contendo dados sobre o modelo e o funcionamento do processo bem como a sua representação visual[Kurz et al.], isto permite que o XML seja analisado para obter-se informações importantes sobre os processos. A maioria das ferramentas BPM disponibiliza a exportação do processo em formato XML seguindo o padrão BPMN. No padrão BPMN um processo é descrito como um diagrama de elementos de fluxo, que são: Tasks (Tarefas), Events (Eventos), Gateways e Sequence Flows [Kurz et al.]. Na Figura 1, podem ser vistos os principais elementos que compoem um diagrama BPMN e que serão importantes para este trabalho.

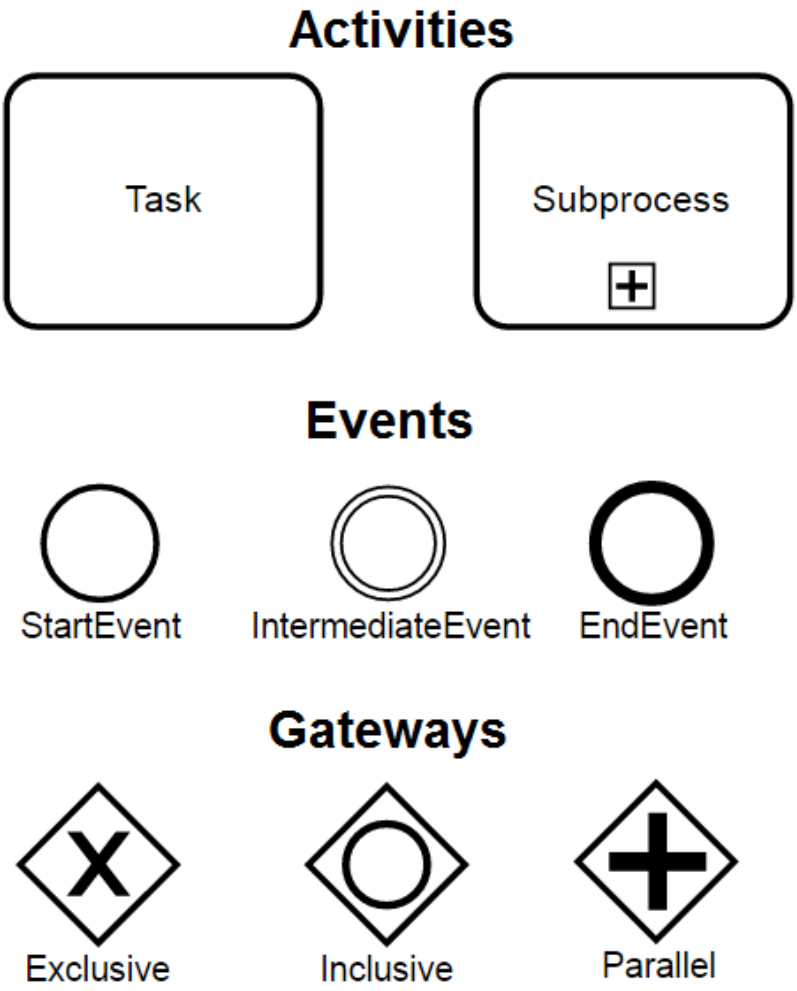

Figura 1. Principais elementos de um processo de acordo com o padrão BPMN.

Uma *Activiti* é a menor parte de um diagrama BPM, e é utilizada quando o diagrama não pode ser dividido mais detalhadamente. Quando uma *Activiti* está inserida no contexto de um processo, ela é chamada de *Task*. Geralmente as Tasks são executadas por um usuário final ou por uma aplicação. Existem diferentes tipos de Tasks para representar os diferentes comportamentos que cada tarefa pode representar. Por exemplo, uma *user-Task* representa uma tarefa em que um usuário deve executar uma ação. Um elemento do

tipo *SubProcess* é uma abstração de um pequeno conjunto de *Tasks*.

Um *Event* é algo que ”acontece”durante o curso de um processo[Model 2011]. Existem três tipos principais de eventos: *Start Event* (indica o início do processo), *End Event* (indica um fim do processo) e *Intermediate Events* (indica um evento entre o início e o fim do processo).

Gateways são usados para controlar o fluxo do processo. Em gateways do tipo *Exclusive*, apenas um dos caminhos que partem do gateway poderão ser seguidos, já em gateways do tipo *Inclusive* um ou mais caminhos podem ser seguidos. Em gateways do tipo *Parallel* todos os caminhos são tomados em paralelo.

Um elemento *Sequence Flow* é usado para exibir a ordem em que os demais elementos são executados em um processo. Cada *sequenceFlow* possui um atributo *sourceRef* que indica de onde este *sequenceFlow* vem, ou sua ”fonte”, e um atributo *targetRef* que indica para onde ele vai, ou seja, seu alvo. Cada *sequenceFlow* possui apenas possui apenas uma fonte e um alvo. Analisando os elementos *sequenceFlow* e analisando os dois atributos citados acima é possível identificar todo o fluxo do processo.

2.2. BPM e Teste

O termo BPM pode ser usado com ênfases diferentes, às vezes com foco em tecnologia (software) e outras vezes em gestão. Mesmo assim, a área tem convergido no entendimento do ciclo de vida de aplicações de BPM, que envolve as atividades de análise, modelagem, execução, monitoramento e otimização [AB 2009].

Os sistemas de BPM (BPMS) têm se afirmado como ferramentas essenciais para suporte a atividades desse ciclo de vida. Atualmente, pode-se dizer que um típico BPMS oferece recursos para definição e modelagem de processos em BPMN, controle da execução e monitoramento de atividades dos processos [Forrester Research 2013]. Há uma tendência dos BPMS em abreviar o desenvolvimento de software, por exemplo através de geradores de formulários Web associados a tarefas dos processos [Winter Green Research 2013]. Nota-se, no entanto, que a preocupação com testes não fica evidente nas ferramentas BPMS. De fato, examinando-se o material promocional e a documentação disponível sobre os principais BPMS, observa-se uma ênfase em etapas de modelagem e execução.

- Teste em geral é pouco abordado em bpm;
- Por serem aplicações como qualquer outra, aplicações BPM podem se beneficiar com a execução de testes;
- Alguns trabalhos relacionados

Existem trabalhos que chamam a atenção para a importância e para a as dificuldades trazidas por um bom teste, bem como trazem alternativas para a melhoria da seleção de casos de teste[**?**]. Esses trabalhos fortalecem a ideia da importância de artefatos para facilitar a criação de testes para processos ou sistemas BPM, no entanto, os trabalhos geralmente abordam a criação de casos de teste baseados em modelos de processo e não em sistemas em si, não é descrita uma forma de automatizar ou executar os casos de teste criados. Existem também trabalhos que visam maneiras de automatizar e melhorar o monitoramento da conformidade dos modelos de processos [**?**, **?**] e que, novamente, são mais voltados para os processos em si e sua administração do que para sistema/software.

3. Gerador de(?)

O objetivo deste trabalho é melhorar e facilitar o teste funcional automatizado de aplicações BPM. Um aspecto importante para o teste funcional são quais fluxos ou etapas serão testadas, pois estes podem definir ou limitar o que será testado, interferindo assim na qualidade e na cobertura dos testes. No entanto, a construção dessas informações pode ser uma tarefa complexa.

Com o objetivo de obter maiores informações que pudessem auxiliar no processo de teste, decidiu-se analisar os arquivos XML no formato BPMN exportados pelas ferramentas BPMS para extrair destes arquivos informações úteis.

Para analisar os arquivos foi desenvolvida uma ferramenta que percorre os arquivos XML e manipula as informações necessárias. A ferramenta foi criada utilizando a linguagem Java, principalmente por esta linguagem já possuir uma opção sólida para a análise de documentos XML[**?**]. Os processos utilizados para a validação foram obtidos no site da *Object Management Group* (OMG) e estão disponíveis no site da organização. Basicamente, a ferramenta criada percorre o arquivo BPMN e analisa os elementos através da nomentaclatura padrão das *tags* no arquivo e então executa uma lógica para criar os diferentes cenários de teste. Ao fim da execução, é gerado um arquivo Excel contendo uma tabela com os todos os possíveis cenários/caminhos do processo.

3.1. Funcionamento do(?)

O principal elemento para o funcionamento da ferramenta é o *Sequence Flow*. Por conter os atributos targetRef e sourceRef os elementos deste tipo permitem percorrer todo o diagrama. Ao iniciar a execução, é solicitado ao usuário o caminho para o arquivo BPMN e a ID do processo a ser avaliado. Um diagrama pode conter mais de um processo e, nesse caso, o usuário pode escolher que todos processos sejam avaliados.

A classe *Node* define um tipo de objeto que guarda informações básicas sobre as tarefas do processo e um *array* de objetos da mesma classe. Durante a execução do um array de objetos da classe Node é preenchido formando assim um array de adjacências. O método principal percorre o processo recursivamente através dos elementos do tipo *Sequence Flow*, enquanto um tarefa for encontrada, um objeto da classe Node é criado e o método é chamado recursivamente de modo a retornar o array de adjacências para este objeto.

A partir do array de adjacências são criados os caminhos possíveis pelos quais o processo pode passar. O método que cria os caminhos percorre recursivamente o array de adjacências criado anteriormente e ”constrói”um novo caminho, tendo como condição de parada o encontro de uma das últimas tarefas a serem executadas. Uma tarefa é uma das últimas quando o elemento que a sucede no fluxo for *End Event*. Ao encontrar uma tarefa final, o caminho é armazenado e é iniciada a construção de um novo caminho.

Os caminhos obtidos são base para a criação da tabela de caminhos possíveis. Para criar a tabela com os caminhos possíveis, cada tarefa existente no processo representará uma coluna na tabela e cada caminho representará uma nova linha. Para cada caminho, se a tarefa estiver presente a coluna será marcada com ”X”caso contrário a coluna será marcada com um ”0”. Caso mais de um processo seja avaliado ao mesmo tempo, a tabela referente à cada processo estará separada individualmente no arquivo final. Na Tabela 1,

Tabela 1. Exemplo de tabela resultante

Quotation Handling	Approve Order	Order Handling	Shipping Handling	Review Order
X	X	0	0	0
X	X	X	X	X

pode ser visto uma tabela resultante da execução do parser para um processo com cinco *tasks* e dois caminhos possíveis.

3.2. Dificuldades/Limitações

Algumas ferramentas exportam o diagrama para o formato BPMN inserindo um "tipo" antes no nome de cada tag XML, por exemplo, a tag de nome task pode estar representada como "semantic:task" e isso pode impedir que o parser do Java identifique os elementos. Assim, foi necessário preparar o parser para tratar este tipo de situação.

Apesar de utilizarem o mesmo padrão BPMN, algumas ferramentas podem utilizar nomes diferentes para representar tarefas no arquivo XML. Por isso, dependendo do BPMS que for utilizado, pode ser necessário substituir o nome das tarefas tratadas no código da ferramenta.

Na criação dos caminhos, uma dificuldade ocorreu devido aos desvios causados por elementos do tipo *Gateway*. Para isso, para cada Node criado no array é guardada o tipo do elemento que o antecede. Assim, na criação dos caminhos, elementos que partem de um desvio de fluxo precisaram ser tratados para criar os caminhos corretamente, por exemplo: se dois elementos, X e Y, vem de um gateway exclusivo, os caminhos obtidos até estes elementos serão duplicados, ou seja, metade dos caminhos passaram apenas por X e metade dos caminhos passarão apenas por Y.

Devido ao fato da ferramenta ser executada recursivamente e percorrer o processo baseado no fluxo dos elementos do tipo *sequenceFlow*, ocorreram problemas em processos onde uma parcela do processo também é executada recursivamente. Estes problemas foram causados devido ao fluxo nunca encontrar um "fim". Para solucionar esses problema foi criado um "delimitador de recursão"que define e controla o número de vezes em que o mesmo *sequenceFlow* será executado.

4. Teste Funcional de Software

O teste funcional é um tipo de teste que permite verificar as saídas de um sistema produzidas a partir de entradas pré-definidas. Este tipo de teste permite testar as funcionalidades, requerimentos e regras de negócio presentes no software[Molinari 2003] verificando a existência de erros, o que auxilia na melhoria da qualidade do software.

Uma das principais medidas para o teste de software é a cobertura de teste. A cobertura de teste mede a abrangência do teste e pode ser expressa pela cobertura dos casos de testes ou pela cobertura do código executado. Existem diversos trabalhos que abordam a importância da cobertura de testes de software[Zhu et al. 1997, Bieman et al. 1996], inclusive ligando o crescimento da qualidade e confiabilidade do software ao crescimento da cobertura dos testes[Malaiya et al. 2002].

No entanto, as atividades executadas para criar testes funcionais com uma boa cobertura podem muitas vezes se tornar exaustivas e trabalhosas, dificultando assim a

execução dos testes de forma adequada. Com o objetivo de melhorar a qualidade da análise e o tempo de execução dos testes, foram criados os testes automatizados, que proporcionam a execução dos testes mais rapidamente [Fantinato et al. 2005]. Quando executada corretamente, a automação de teste é uma das melhores formas de reduzir o tempo de teste no ciclo de vida do software, diminuindo o custo e aumentando a produtividade do desenvolvimento de software como um todo, além de, consequentemente, aumentar a qualidade do produto final.

Para executar os testes funcionais automatizados em aplicações Web, pode-se utilizar ferramentas livres como Selenium[**?**], Watir[**?**] ou Geb[**?**]. No trabalho anterior[de Moura and Charão 2015], onde os testes funcionais se mostraram mais promissores, foi utilizada a ferramenta Selenium, aliada ao Cucumber-JVM[**?**] para descrição dos testes. A escolha foi motivada pelo grande número de referências ao Selenium na Web, confirmadas por um trabalho que apresentou resultados satisfatórios com Selenium e Cucumber[Pannu , Chiavegatto et al. 2013].

4.1. Teste Funcional Automatizado utilizando *Selenium* e *Cucumber-JVM*

O processo para a execução de um teste funcional utilizando o Selenium aliado ao Cucumber-JVM é composto de cinco etapas: captura da interação do usuário com a aplicação e exportação do código gerado, criação dos cenários de teste, criação das definições dos passos de teste, implementação dos métodos para cada passo e, por fim, execução do teste.

Para efetuar a captura da interação do usuário com a aplicação é utilizado o Selenium IDE (*Integrated Development Environment*), que permite gravar as ações do usuário conforme elas são executadas. Assim, a funcionalidade que deseja ser testada deve ser executada para que os passos sejam gravados. Após a captura da interação é possível exportar o *script* utilizando diversas linguagens de programação disponíveis, neste caso foi escolhida a linguagem Java.

Os testes utilizando o Cucumber são compostos, basicamente, por dois arquivos: arquivos que especificam as funcionalidades *features* e por arquivos de definição de passos *steps*.

Os arquivos com as funcionalidades são escritos utilizando a linguagem Gherkin[Cucumber Limited 2015] e são compostos por cenários, os cenários representam uma fração da aplicação que vai ser testada. Um cenário também pode ser descrito como a definição, em ordem de execução, das etapas que são executados nessa fração, bem como dos resultados esperados para validar a aplicação. Por exemplo, após um *gateway* exclusivo com dois caminhos disponíveis, já é possível obter dois cenários possíves: um cenário executando cada caminho.

Para que cada etapa do cenário seja executada, é necessária a criação de *steps* que irão traduzir os passos definidos na linguagem Gherkin para ações que vão interagir com o sistema. Cada *step*, geralmente, irá chamar um método Java que irá efetivamente executar a interação com a aplicação. A implementação destes métodos pode ser feita apenas utilizando o código extraído através do Selenium após a captura das ações do usuário.

Assim, mesmo utilizando o Cucumber e Selenium para facilitar a criação dos

testes, ainda é preciso analisar o processo e extrair os cenários de teste que devem ser criados manualmente no arquivo de *features*. Além dos cenários, também deve ser criado manualmente o arquivo de *steps*, contendo uma ”passo”correspondente a cada etapa de todos os cenários criados. Esta etapa pode ser trabalhosa, principalmente quando for necessário testar todos diversos cenários que um processo pode ter ou quando o processo for extenso.

5. Resultados

Com a análise do arquivo BPMN através do parser criado foi possível obter dois arquivos essenciais para o teste funcional automatizado utilizando o Selenium e o Cucumber-JVM: o arquivo de *features* contendo os cenários e o arquivo de *steps* contendo métodos que para efetuar a execução de cada etapa do cenário.

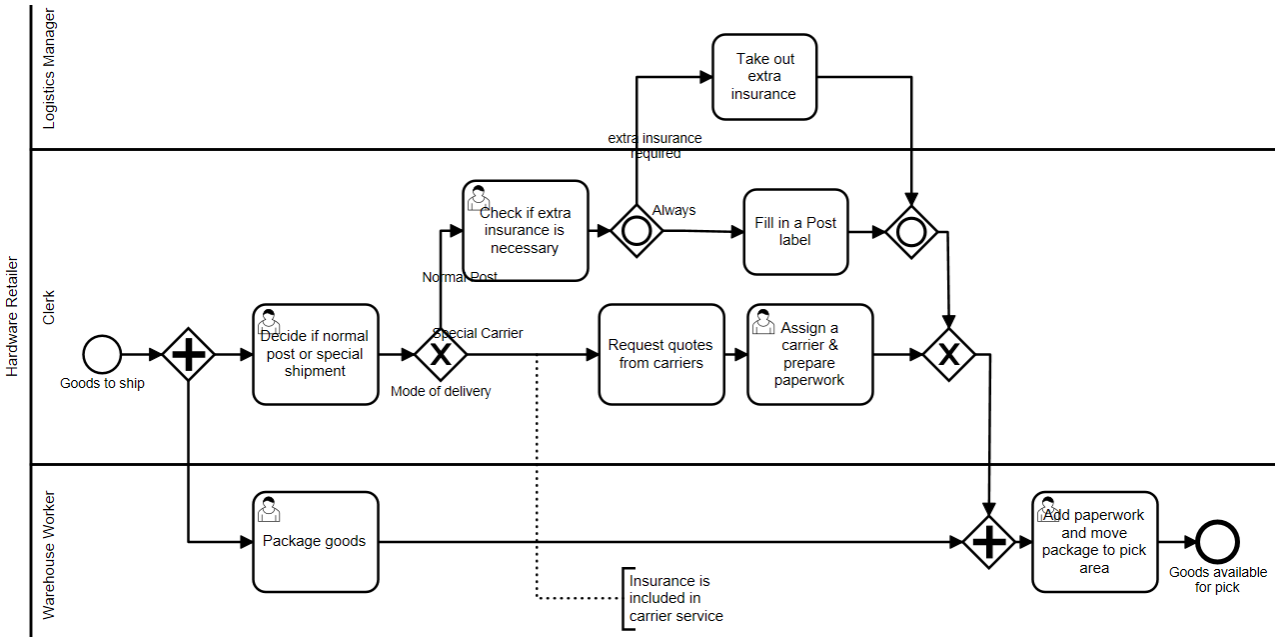

Figura 2. Exemplo de Processo. Adaptado de: Object Management Group

Por exemplo, o processo na Figura 2 possui *gateways* de diferentes tipos, gerando assim várias divisões de fluxo e aumentando os caminhos que podem ser seguidos. Para executar o teste completo do processo, testando todas as possibilidades e com boa cobertura, seria necessário planejar os cenários de teste e criá-los manualmente. Para criar os cenários também é necessário identificar quais tarefas são executadas por um usuário e o processo possui várias *tasks*. Por fim, também seria necessário criar os *steps* para cada etapa do cenário manualmente.

Executando o parser criado utilizando o arquivo BPMN referente ao diagrama da Figura 2 como exemplo, obtemos os cenários de teste representados na Figura 3. Para o teste funcional do processo, três cenários são possíveis, representando as duas principais divisões de fluxo encontradas no processo. A primeira divisão de fluxo formada pelo *gateway* do tipo paralelo logo no início do processo, está contida no cenário de forma que os cenários *Hardware Retailer 0* e *Hardware Retailer 1* fazem parte de um fluxo e o cenário *Hardware Retailer 2* representa o segundo fluxo possível. A segunda divisão de

```
Feature: Testing BPM Processes

Scenario: Hardware Retailer 0
Given I am on task Decide if normal post or special shipment
When
Then
When I am on task Assign a carrier & prepare paperwork
Then
When I am on task Add paperwork and move package to pick area
Then


Scenario: Hardware Retailer 1
Given I am on task Decide if normal post or special shipment
When
Then
When I am on task Check if extra insurance is necessary
Then
When I am on task Add paperwork and move package to pick area
Then

Scenario: Hardware Retailer 2
When I am on task Package goods
Then
When I am on task Add paperwork and move package to pick area
Then
```

Figura 3. Cenários obtidos através do arquivo BPMN

fluxo no processo da Figura 2 é formada pelo *gateway* do tipo exclusivo, e essa divisão é representada pelos cenários *Hardware Retailer 0* e *Hardware Retailer 1*. Há ainda uma terceira divisão de fluxo no processo, esta divisão não foi levada em conta pelo parser pois as tarefas subsequentes a essa divisão não são tarefas executadas pelo usuário.

Como foi mencionado anteriormente, apenas tarefas executadas por usuários são levadas em conta para o teste funcional, e por isso nem todas as tarefas do processo estarão presentes nos cenários de teste.

Para cada etapa nos cenários criados será criado um método correspondente dentro do arquivo de *steps* chamado *stepDefinition*. Uma parte do arquivo de *steps* resultante da análise do processo em questão pode ser vista na Figura 4. Como pode ser visto nesta figura, para a etapa *I am on the task Decide if normal post or special shipment* foi criado um método com um nome genérico e contendo a anotação *@Given("^I am on the task Decide if normal post or special shipment$")* . Os métodos podem ser preenchidos

Tabela 2. Tabela obtida através do arquivo BPMN

Decide if normal...	Request quotes...	Assign a carrier & prepare paperwork	Check if extra...	Take out extra insurance	Fill in a Post label	Package goods	Add paperwork and...
X	X	X	0	0	0	X	X
X	0	0	X	X	0	X	X
X	0	0	X	0	X	X	X
X	0	0	0	0	0	X	X
X	0	0	X	X	X	X	X

posteriormente com os dados gravados pelo Selenium na etapa de captura da interação com a aplicação.

```
import cucumber.api.java.en.*;
import cucumber.runtime.PendingException;
public class StepsDefinition{

@Before
public void beforeScenario(){
}

@After
public void afterScenario() {
}

//Methods for process Hardware Retailer
@Given("^I am on the task Decide if normal post or special shipment$")
public void method0() throws Exception {}

@when("^name$")
public void method1() throws Exception {}

@Then("^name$")
public void method2() throws Exception {}

@When("^I am on the task Assign a carrier & prepare paperwork$")
public void method3() throws Exception {}
```

Figura 4. Métodos obtidos através do arquivo BPMN

Neste exemplo, apenas um processo foi analisado, mas arquivos BPMN podem conter vários processos. Neste caso, se for necessário gerar os dados para todos os processos dentro de um arquivo, os arquivos de *features* e *steps* conterão, respectivamente, os cenários e métodos para todos os processos dentro do arquivo BPMN.

Na Tabela 2 pode ser vista a tabela de caminhos possíveis obtida através do arquivo BPMN. Nesta tabela, cada coluna representa uma tarefa do diagrama e cada linha representa um caminho possível, facilitando a visualização do processo. Todas as tarefas do diagrama estão contidas na tabela, não apenas as executadas por usuários.

Com estes resultados se dizer que este trabalho atingiu seu objetivo, pois foi

possível obter dois arquivos muito importantes para a execução do teste funcional automatizado: o arquivo com os cenários de teste e o arquivo com os métodos para executar cada etapa dos cenários. A criação automatizada dos dois arquivos permite facilitar e melhorar a criação dos testes para os sistemas BPM, diminuindo o tempo de análise dos processos bem como tornando o teste mais completo ao mesmo tempo em que verifica todos os possíveis fluxos que um processo pode seguir. Assim, a qualidade e a cobertura dos testes também é melhorada pois todos os caminhos e cenários possíveis com tarefas executadas por usuários são criados. Por consequencia, a qualidade dos sistemas também pode ser beneficiada com um processo de teste mais completo e facilitado.

6. Conclusão

- Mesmo utilizando ferramentas para teste automatizado, criação dos elementos do teste podem ser trabalhosas;
  - O parser criado auxilia nesse ponto.
- Importante ter uma boa cobertura;
- BPMN parser analiza a notação para gerar elementos para teste automatizado = facilita o teste e aumenta a cobertura.
- Trabalhos futuros (tirar repeticoes de cenários, tentar gerar mais coisas/informações, tabelinha só com tarefas executadas pelos usuários..)

7. Referências

Referências

[AB 2009] AB (2009). *Guide to the business process management common body of knowledge (BPM CBOK®): ABPMP BPM CBOK®-[version 2.0-second release]*. ABPMP.

[Bieman et al. 1996] Bieman, J. M., Dreilinger, D., and Lin, L. (1996). Using fault injection to increase software test coverage. In *Software Reliability Engineering, 1996. Proceedings., Seventh International Symposium on*, pages 166–174. IEEE.

[Chiavegatto et al. 2013] Chiavegatto, R. et al. (2013). Especificação e automação colaborativas de testes utilizando a técnica BDD. In *XII Simpósio Brasileiro de Qualidade de Software*, pages 334–341.

[Cucumber Limited 2015] Cucumber Limited (2015). Gherkin Reference.

[de Moura and Charão 2015] de Moura, J. L. and Charão, A. (2015). Automação de testes em aplicações de bpms: um relato de experiência. In *XIV Simpósio Brasileiro de Qualidade de Software*, pages 212–219.

[Fantinato et al. 2005] Fantinato, M., da Cunha, A. C. R., Dias, S. V., Cardoso, S. A. M., and Cunha, C. A. Q. (2005). Autotest–um framework reutilizável para a automação de teste funcional de software. *Cad. CPqD Tecnologia*, 1(1):119–131.

[Forrester Research 2013] Forrester Research (2013). The forrester wave: BPM suites, Q1 2013.

[Kurz et al.] Kurz, M., Menge, F., and Misiak, Z. Diagram interchangeability in bpmn 2.

[Malaiya et al. 2002] Malaiya, Y. K., Li, M. N., Bieman, J. M., and Karcich, R. (2002). Software reliability growth with test coverage. *Reliability, IEEE Transactions on*, 51(4):420–426.

[Model 2011] Model, B. P. (2011). Notation (bpmn) version 2.0. *OMG Specification, Object Management Group*.

[Molinari 2003] Molinari, L. (2003). *Testes de software: produzindo sistemas melhores e mais confiáveis: qualidade de software: soluções, técnicas e métodos*. Érica.

[Pannu] Pannu, Y. S. Test automation using cucumber and selenium webdriver.

[Winter Green Research 2013] Winter Green Research (2013). Business process management (BPM) cloud, mobile, and patterns: Market shares, strategies, and forecasts, worldwide, 2013 to 2019.

[Zhu et al. 1997] Zhu, H., Hall, P. A., and May, J. H. (1997). Software unit test coverage and adequacy. *Acm computing surveys (csur)*, 29(4):366–427.