Geração de Casos para Testes Automatizados de Aplicações de Gerenciamento de Processos de Negócio a partir de Modelos em BPMN

Trilha de Trabalhos Técnicos

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

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

{jmoura,andrea}@inf.ufsm.br

Abstract. *This article proposes an approach to generate cases for automated testing of web applications implemented with the support of BPMS, from BPMN models, aiming to shorten the building test scripts effort. The work is primarily focused on functional testing and has the following objectives: (i) generate a table of application execution paths from the flow analysis in the BPMN model and (ii) generate the initial code of test scripts to be run on a given Web application testing framework. Throughout the article, we describe the design and implementation of tools developed to achieve these goals, targeting the automation testing using Selenium and Cucumber tools. The approach was applied to a process repository and showed to be able to meet the necessary requirements.*

Resumo. *Este artigo propõe uma abordagem para geração de casos para testes automatizados de aplicações Web implementadas com o apoio de BPMS, a partir de modelos BPMN, visando abreviar o esforço de construção de* scripts *de teste. O trabalho é focado principalmente em testes funcionais e tem como objetivos específicos: (i) gerar uma tabela de caminhos de execução da aplicação a partir da análise de fluxos no modelo BPMN e (ii) gerar o código inicial dos* scripts *de teste, a serem executados em um dado arcabouço de teste de aplicações Web. Ao longo do artigo, descreve-se o projeto e implementação de ferramentas desenvolvidas para atingir esses objetivos, tendo como alvo a automação de testes usando as ferramentas Selenium e Cucumber. A abordagem foi aplicada a um repositório de processos e mostrou-se capaz de cumprir os requisitos estabelecidos.*

1. Introdução

Testes automatizados constituem um tema recorrente em comunidades interessadas em qualidade de software [Delamaro et al. 2007, Dustin et al. 2009]. Em comparação com testes manuais, a automação de testes de software pode trazer benefícios como, por exemplo, a repetibilidade, o aumento da cobertura e a redução do esforço na execução dos testes [Rafi et al. 2012, Pinheiro et al. 2015]. Quando executada satisfatoriamente, a automação de testes é vantajosa para reduzir o tempo desta etapa no ciclo de vida do software, diminuindo o custo e aumentando a produtividade do desenvolvimento como um todo, além de, principalmente, aumentar a qualidade do produto final.

Existem, no entanto, limitações associadas aos testes automatizados. Em um estudo abrangendo a academia e a comunidade de prática de testes de software, os principais problemas atribuídos aos testes automatizados foram relativos ao alto investimento inicial em configuração, escolha de ferramentas e treinamento da equipe [Rafi et al. 2012]. Outro estudo nesta linha chama atenção para a dificuldade de criação de *scripts* de teste [Wiklund et al. 2014].

Dentre as diversas classes de software que podem ser alvo de testes automatizados, tem-se as aplicações de gerenciamento de processos de negócio (*Business Process Management* – BPM). Designa-se por BPM o conjunto de conceitos, métodos e técnicas para suportar a análise, modelagem, execução, monitoramento e otimização dos processos de negócio [Weske 2012]. Nesse contexto, surgiram os sistemas de BPM (*Business Process Management Systems* ou *Suites* – BPMS), que tipicamente oferecem recursos para definição e modelagem de processos em BPMN (*Business Process Model and Notation*), controle da execução e monitoramento de atividades dos processos [Forrester Research 2013]. Há uma tendência dos BPMS em abreviar o desenvolvimento de software [Winter Green Research 2013], oferecendo agilidade na produção de aplicações Web que executam os processos expressos em BPMN.

Tarefas como verificação e testes ainda são consideradas um desafio na área de BPM [Van der Aalst 2013]. De fato, o teste automatizado de aplicações de BPM é pouco abordado, tanto pela comunidade da área de BPM [Weske 2012] como da área de testes de software [Graham and Fewster 2012]. Em um trabalho anterior [de Moura and Charão 2015], buscou-se explorar este tema e constatou-se algumas dificuldades na realização de testes automatizados de carga e funcionais, em aplicações Web para execução de um mesmo processo, implementadas com o auxílio de dois BPMS *open source* distintos: Bonita BPMS e Activiti. Os resultados corroboraram problemas já levantados por outros autores [Rafi et al. 2012, Wiklund et al. 2014], além de apontar aspectos relacionados especificamente a aplicações Web criadas e executadas com o apoio de BPMS. Em particular, notou-se que a criação de *scripts* de teste para processos com muitas tarefas poderia exigir um esforço demasiado. Neste contexto, buscando reduzir este esforço, tomou-se por hipótese que a criação de casos de teste para um dado processo poderia ser abreviada, usando como entrada o modelo BPMN de tal processo.

Assim, o presente trabalho propõe uma abordagem para geração de casos para testes automatizados de aplicações Web implementadas com o apoio de BPMS, a partir de modelos BPMN, visando abreviar o esforço de construção de *scripts* de teste. O trabalho é focado principalmente em testes funcionais e tem como objetivos específicos: (i) gerar uma tabela de caminhos de execução da aplicação a partir da análise de fluxos no modelo BPMN e (ii) gerar o código inicial dos *scripts* de teste, a serem executados em um dado arcabouço de teste de aplicações Web.

O restante deste artigo está organizado como segue. Na seção 2, apresenta-se uma fundamentação para o trabalho, apresentando-se conceitos e termos associados a BPM, seguidos de um embasamento sobre testes funcionais automatizados. A seção 3 discute trabalhos relacionados e, em seguida, a seção 4 apresenta a abordagem proposta, incluindo aspectos de projeto e implementação. Na seção 5 são apresentados resultados e exemplos obtidos na avaliação da abordagem. Por fim, a seção 6 tece considerações finais sobre o trabalho.

2. Fundamentação

2.1. Termos e Conceitos Associados a BPM

O termo BPM pode ser usado com ênfases diferentes, dependendo do contexto. Em geral, é um termo usual em Administração e Engenharia de Produção, embora também seja de importância para as áreas de Computação e Tecnologia da Informação [Ramos and Bessa 2015]. Qualquer que seja a ênfase, há uma convergência de entendimento quanto ao ciclo de vida de aplicações de BPM, que engloba as atividades de análise, modelagem, execução, monitoramento e otimização de processos de negócio [ABPMP 2012].

No suporte a esse ciclo de vida, os sistemas de BPM (BPMS) têm se afirmado como ferramentas essenciais. Desde as origens do termo BPMS [Smith and Fingar 2003], surgiu uma ampla diversidade de sistemas no mercado, incluindo desde BPMS proprietários de grandes fabricantes de software como IBM, SAP e Oracle, até sistemas com versões de código aberto, como por exemplo Bonita BPM, Activiti e Camunda. Tais ferramentas oferecem recursos para modelagem, controle de execução e monitoramento de processos, cada uma com suas especificidades. Esses e outros BPMS possuem atualmente um aspecto em comum: a representação de processos em 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*" [OMG 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 2016]. Isto permite que o XML seja analisado para obter-se informações sobre os processos. A maioria dos BPMS 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. Na Figura 1, podem ser vistos os principais elementos que compõem um diagrama BPMN e que serão importantes para este trabalho.

As *Activities* constituem a menor parte de um diagrama em BPMN, sendo utilizadas quando o diagrama não pode ser dividido mais detalhadamente. Quando uma *Activity* 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 de cada tarefa. 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 [OMG 2011]. Existem três tipos principais de eventos: *Start Event* (indica o início do processo), *End Event* (indica um fim do processo) e *Intermediate Event* (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 fluxos que partem do *Gateway* poderão ser seguidos, já em *Gateways* do tipo *Inclusive* um ou mais fluxos podem ser seguidos. Em *Gateways* do tipo

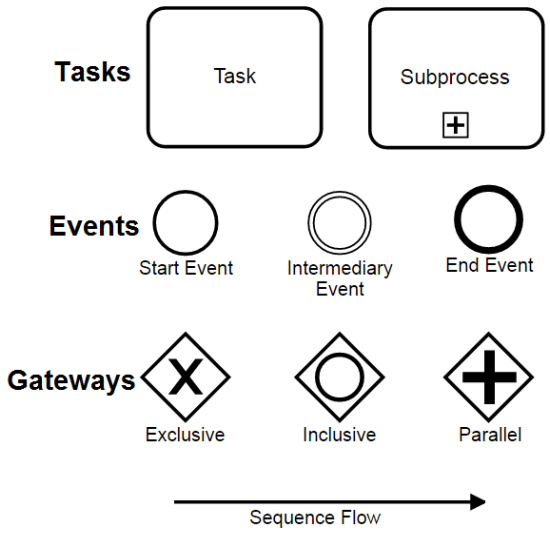


Figura 1. Principais elementos de um processo de acordo com o padrão BPMN

Parallel, todos os fluxos são executados ao mesmo tempo.

Um *Sequence Flow* é usado para exibir a ordem em que os demais elementos são executados em um processo. Cada *Sequence Flow* possui apenas uma origem (ou fonte) e um destino (ou alvo), que podem ser *Activities*, *Events* ou *Gateways*. Analisando os elementos *Sequence Flow* de um diagrama, é possível identificar todo o fluxo de execução do processo.

2.2. Testes Funcionais Automatizados de Aplicações Web

Teste funcional é uma técnica 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, requisitos e regras de negócio presentes no software [Delamaro et al. 2007], verificando a existência de erros e, com isso, auxiliando na melhoria da qualidade do software.

Uma das principais medidas para o teste de software é sua cobertura. A cobertura de teste mede sua abrangência 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 [Sun et al. 2016, Rayadurgam and Heimdahl 2001, 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].

Criar testes funcionais com uma boa cobertura pode se tornar uma tarefa exaustiva e trabalhosa, motivando a realização de testes funcionais automatizados. Com a popularização de aplicações Web, no entanto, testes automatizados revelaram novos desafios, devido à natureza distribuída, a heterogeneidade e a dinamicidade das aplicações [Garousic et al. 2013]. Estas características estão presentes em grande parte das aplicações de BPM construídas com o apoio de BPMS, uma vez que a maioria destes sistemas é voltado para a Web. Porém, testes automatizados de software não fazem parte dos recursos comumente oferecidos pela maioria dos BPMS, restando assim a opção de

se utilizar ferramentas de testes automatizados voltadas para aplicações Web em geral.

Para executar os testes funcionais automatizados em aplicações Web, pode-se utilizar ferramentas livres como Selenium [Selenium 2015], Watir [Watir 2015] ou Geb [Geb 2015]. Em um trabalho anterior [de Moura and Charão 2015], os testes funcionais utilizando a ferramenta Selenium, aliada ao Cucumber-JVM [Cucumber Limited 2015a], se mostraram promissores quando aplicados a etapas de um processo construído com dois diferentes BPMS. A escolha dessas ferramentas foi motivada pelo grande número de referências ao Selenium em fóruns especializados, corroboradas em trabalhos acadêmicos [Chiavegatto et al. 2013, Pinheiro et al. 2015]. Assim, manteve-se essa escolha no presente trabalho.

2.2.1. Testes Automatizados com *Selenium* e *Cucumber-JVM*

O processo para a realização 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. No presente trabalho, como forma de delimitá-lo, utiliza-se somente a linguagem Java.

Os testes utilizando 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 (*features*) são escritos na linguagem Gherkin [Cucumber Limited 2015b] 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 executadas nessa fração, bem como dos resultados esperados para validar a aplicação. Por exemplo, após um *Gateway* exclusivo com dois fluxos disponíveis, tem-se dois cenários possíveis, um cenário executando cada fluxo.

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, invocará um método em 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 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 um "passo" correspondente a cada etapa de todos

os cenários criados. Esta etapa pode ser trabalhosa, principalmente quando for necessário testar diversos cenários que um processo pode ter ou quando o processo for extenso.

3. Trabalhos Relacionados

Os sistemas BPM geralmente oferecerem recursos para definição e modelagem de processos em BPMN, controle da execução e monitoramento de atividades dos processos [Forrester 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. Visivelmente, tais recursos são um diferencial no desenvolvimento de aplicações de BPM, em comparação ao desenvolvimento de software em geral. No entanto, aplicações de BPM também estão sujeitas a defeitos e, por isso, podem se beneficiar de avanços na área de testes de software.

Existem trabalhos que chamam a atenção para a importância e para a as dificuldades trazidas pela execução de testes, bem como trazem alternativas para a melhoria da seleção de casos de teste [Böhmer and Rinderle-Ma 2015]. Esses trabalhos fortalecem a ideia da importância de informações 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. Também 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 [Ly et al. 2015, Van der Aalst et al. 2012] e que, novamente, são mais voltados para os processos em si e sua administração do que para sistemas/software.

4. Abordagem Proposta

O objetivo deste trabalho é auxiliar na geração de casos para testes funcionais automatizados de aplicações Web construídas com o apoio de BPMS, a partir de modelos BPMN. Neste sentido, um aspecto importante para o teste funcional é identificar quais fluxos ou etapas serão testadas, pois estes determinam caminhos de execução da aplicação e podem definir ou limitar o teste, interferindo assim na qualidade e na cobertura dos testes.

Com o objetivo de obter informações capazes de auxiliar na identificação dos fluxos, projetou-se e implementou-se uma ferramenta que analisa os arquivos XML no formato BPMN, exportados por ferramentas BPMS. Para tratar essas informações, optou-se por analisar o processo de modo a transformá-lo em uma tabela contendo todos os possíveis fluxos que podem ser executados no processo, para então permitir que sejam realizadas outras análises. A partir da tabela, pode-se gerar outros elementos, conforme a necessidade de cada domínio. Neste trabalho, aproveita-se os resultados da tabela de fluxos como entrada para uma ferramenta concebida e implementada para gerar o código inicial dos *scripts* de teste, para uso com Selenium e Cucumber. Uma visão geral desta abordagem é ilustrada na Figura 2, em que as etapas de análise do XML e de tratamento de resultados da tabela correspondem a funcionalidades das ferramentas implementadas, descritas nas seções a seguir.

4.1. Geração da Tabela de Fluxos

Para analisar um arquivo BPMN, foi elaborada uma estratégia que percorre sua representação em XML e manipula as informações necessárias. Enquanto o arquivo é

Figura 2. Visão geral da abordagem proposta

percorrido, os elementos BPMN são analisados através da nomenclatura padrão especificada nas *tags* XML. Ao fim da execução, é gerado um arquivo em formato Excel, contendo uma tabela com todos os possíveis cenários/fluxos do processo. Na implementação da ferramenta de análise, foi utilizada a linguagem Java, que possui uma opção de API consolidada para a análise de documentos XML (JAXP).

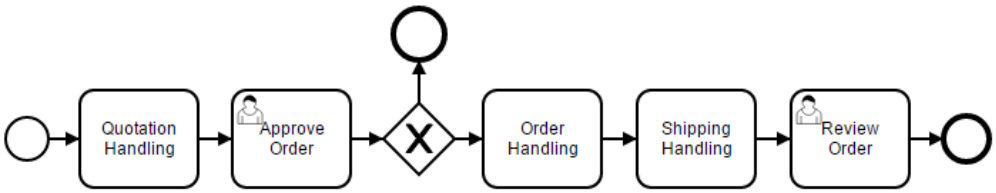
O elemento BPMN mais importante para a análise é o *Sequence Flow*. Cada *Sequence Flow* possui um atributo `sourceRef`, que indica de onde este *Sequence Flow* vem, ou sua ”fonte”, e um atributo `targetRef`, que indica para onde ele vai, ou seja, seu alvo. Por conter esses atributos, os elementos deste tipo permitem percorrer todo o diagrama. Ao iniciar a execução da ferramenta de análise, 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 sejam analisados todos ou somente alguns processos.

Na implementação da ferramenta, foi criada a classe *Node* que define um tipo de objeto que guarda informações básicas sobre as tarefas do processo e um *array* de objetos dessa classe. Durante a execução, esse *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 houver tarefas encontradas no XML, 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 fluxos possíveis pelos quais o processo pode passar. O método que cria os fluxos percorre recursivamente o *array* de adjacências criado anteriormente e “constrói” um novo fluxo possível, 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 elemento do tipo *End Event*. Ao encontrar uma tarefa final, a informação é armazenada e é iniciada a construção de um novo fluxo possível.

Para criar a tabela de fluxos, cada tarefa existente no processo representará uma coluna na tabela e cada fluxo representará uma nova linha. Para cada fluxo, se a tarefa estiver presente a coluna será marcada com “X”, caso contrário a coluna será marcada com um “-”. Caso mais de um processo seja avaliado ao mesmo tempo, a tabela referente a cada processo estará separada individualmente no arquivo final.

Na Figura 3, pode ser visto um resultado da execução da ferramenta de análise para um exemplo de processo, que possui cinco *Tasks* e dois fluxos possíveis. Com a geração desta tabela, podem ser realizadas diversas análises sobre o processo: número de fluxos possíveis, identificar gargalos, dentre outras.



Quotation Handling	Approve Order	Order Handling	Shipping Handling	Review Order
X	X	-	-	-
X	X	X	X	X

Figura 3. Exemplo de processo analisado e a tabela de fluxos resultante

4.2. Dificuldades e Limitações da Análise

A maioria das dificuldades de implementação da ferramenta de análise são relacionadas à estrutura dos processos e como a implementação interpretaria essas características. Na criação dos fluxos possíveis, uma dificuldade ocorreu devido aos desvios causados por elementos do tipo *Gateway*. Para isso, para cada Node criado no *array* é guardado o tipo do elemento que o antecede. Assim, na criação dos fluxos, elementos que partem de um desvio de fluxo precisaram ser tratados para criar os fluxos corretamente, por exemplo: se dois elementos, X e Y, vêm de um *Gateway* do tipo exclusivo, os fluxos obtidos até estes elementos serão duplicados, ou seja, metade dos fluxos passarão apenas por X e metade dos fluxos passarão apenas por Y.

No caso de *Gateways* do tipo paralelo, onde os fluxos são executados ao mesmo tempo, foi necessária fazer uma escolha para representar estes fluxos na tabela. Assim, decidiu-se por expressar cada fluxo em paralelo separadamente. Estimou-se que esta opção facilitaria na visualização dos fluxos para os testes já que, para a execução dos testes funcionais automatizados, é analisado cada fluxo como um "cenário" diferente. Também optou-se por tratar elementos *Subprocess*, que representam um pequeno conjunto de tarefas, como uma única tarefa.

Devido ao fato da estratégia implementada ser executada recursivamente e percorrer o processo baseado no fluxo dos elementos do tipo *Sequence Flow*, 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.

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* XML do Java identifique os elementos. Assim, foi necessário preparar métodos para tratar este tipo de situação, removendo os elementos encontrados antes dos nomes de *tags*.

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

4.3. Geração de Código para os Testes Funcionais

A tabela de fluxos contém informações úteis para a criação de *scripts* de teste. A partir dela, pode-se automatizar a geração de elementos importantes para testes usando Selenium e Cucumber: o arquivo de *features*, contendo os cenários de teste, e o arquivo de *steps*.

O arquivo contendo os cenários de teste é essencial para o teste funcional com Cucumber, no entanto a criação dos arquivos de *features* contendo os cenários e do arquivo de *steps* pode ser trabalhosa, como já foi mencionado. Assim, de acordo com a visão geral da abordagem proposta (Figura 2), a tabela resultante da análise do arquivo BPMN pode ser processada para gerar os arquivos com códigos, que posteriormente deverão ser completados para se tornarem *scripts* de teste completos.

Para criar estes arquivos, a ferramenta de geração de código faz um tratamento dos resultados da tabela de fluxos. Cada fluxo que foi obtido na análise, e que consta na tabela de fluxos, foi considerado um cenário diferente. Como se trata de testes funcionais, apenas tarefas que podem ser executadas por um usuário foram ser avaliadas (`userTask`,`manualTask`...). Assim, para cada fluxo é criado um novo cenário no arquivo de *features*, utilizando a notação do Cucumber, e contemplando apenas as tarefas que podem ser executadas por um usuário. O método correspondente a cada passo do cenário é criado ao mesmo tempo e inserido no arquivo de *steps*.

5. Avaliação

Para avaliar a abordagem proposta, foram buscados processos já existentes, de diferentes domínios, contendo ao menos três tarefas e uma divisão de fluxo com ou sem *Gateways*. Procurou-se também processos que tivessem algumas diferenças de notação, para verificar como a solução implementada trataria essas diferenças. Visando utilizar um repositório de processos amplamente acessível, os processos escolhidos foram obtidos no *site* da *Object Management Group* (OMG) e estão disponíveis em `http://www.omg.org/spec/BPMN/20100602/2010-06-03/`.

A Tabela 1 apresenta uma visão geral dos processos utilizados na avaliação. As informações referentes ao número de tarefas e de fluxos foram extraídas das tabelas geradas pela ferramenta desenvolvida. Esses números foram conferidos manualmente e estão de acordo com os diagramas dos processos. A divisão de fluxos é importante para os testes, pois delimita quantos fluxos existem no processo, ou seja, quantos fluxos podem ser testados.

Com a análise de um arquivo BPMN através da ferramenta desenvolvida, é possível obter uma tabela relacionando as tarefas e os fluxos possíveis dentro do processo. Por exemplo, o processo na Figura 4 possui *Gateways* de diferentes tipos, gerando assim várias divisões de fluxo e aumentando os fluxos que podem ser seguidos.

Ao executar a ferramenta sobre o arquivo BPMN referente ao processo da Figura 4, foi obtida a Tabela 2. Nesta tabela, cada coluna representa uma tarefa do diagrama e cada linha representa um fluxo possível, facilitando a visualização do processo. A geração da tabela pode auxiliar a reduzir o tempo de análise necessário para executar o teste funcional dos processos, possibilitando que os fluxos possíveis sejam vistos rapidamente.

Tabela 1. Resumo dos processos utilizados na avaliação

Nome do Processo	Número de tarefas	Número de Gateways	Tipo de Gateway	Fluxos possíveis
Hardware Retailer	8	3	Paralelo, Exclusivo e Inclusivo	5
Nobel Prize – Nobel Assembly	3	0	Sem divisão de fluxos	1
Incident Management (Whole Collab) – 2nd level support agent	3	1	Exclusivo	2
Incident Management (Whole Collab) – Trouble Ticket System	6	2	Exclusivo	3
Nobel Prize - Nobel Commitee for medicine	10	1	Exclusivo	2
Order Fulfillment	7	1	Exclusivo	2
Pizza - Pizza Customer	5	1	Exclusivo	2
Travel Booking	10	4	Exclusivo	7

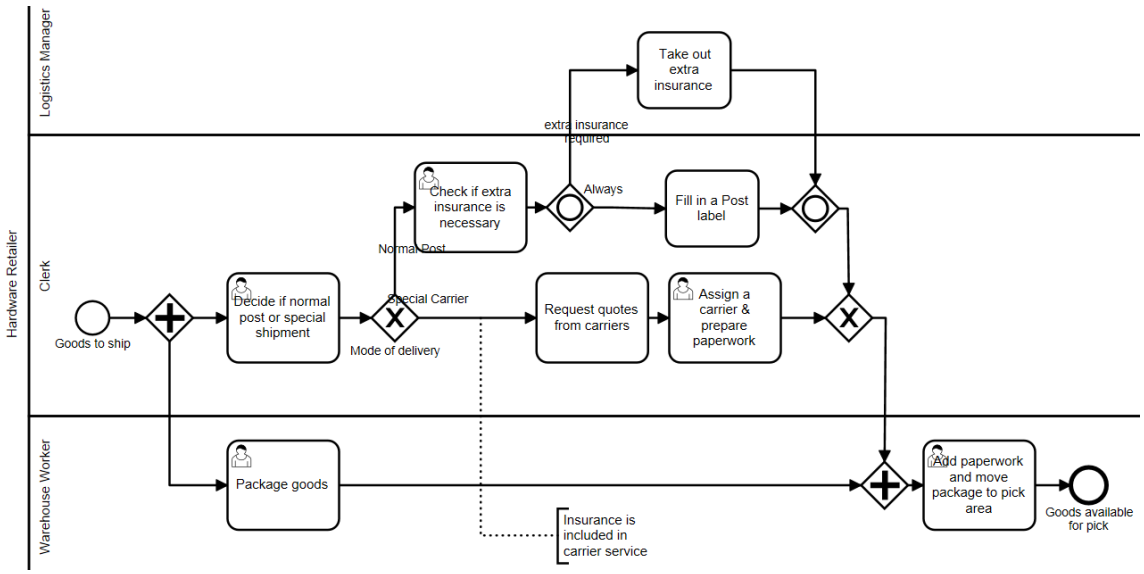


Figura 4. Diagrama do processo "Hardware Retailer". Fonte: OMG

A tabela também permite que sejam feitas análises importantes para o teste funcional do processo como, por exemplo: identificar gargalos, identificar quais tarefas são executadas mais frequentemente, identificar os caminhos mais importantes para os testes, entre outros. Por exemplo, na Tabela 2 pode-se dizer que o teste da primeira e da última tarefa do processo é importante, pois a grande maioria dos fluxos possíveis passarão por essas tarefas. Da mesma forma que permite análises importantes para os testes, a tabela gerada também auxilia na criação de testes mais completos, pois permite visualizar todos os fluxos possíveis de serem testados dentro de um processo.

Nesse trabalho também é abordada a criação, a partir da tabela gerada, de códigos para teste funcional de processos utilizando as ferramentas de automação Selenium e Cucumber-JVM. Para executar o teste completo do processo, testando todas as possibilidades e representando as informações obtidas na tabela, 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 em um processo que possui várias *Tasks*. Por fim, também seria necessário criar os *steps* para cada etapa do cenário manualmente.

Executando a ferramenta desenvolvida sobre o arquivo BPMN referente ao dia-

Tabela 2. Tabela obtida através do arquivo BPMN

Decide if normal...	Request quotes...	Assign a carrier...	Check if extra...	Take out extra insurance	Fill in...	Package goods	Add paperwork ...
X	X	X	-	-	-	-	X
X	-	-	X	X	-	-	X
X	-	-	X	-	X	-	X
-	-	-	-	-	-	X	X
X	-	-	X	X	X	-	X

grama da Figura 4, obtém-se os cenários de teste representados no trecho da Figura 5.

```
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 5. Cenários de teste gerados

Para o teste funcional do processo em questão, 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 fluxo no processo da Figura 4 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, mas esta divisão não foi levada em conta pela ferramenta pois as tarefas subsequentes a essa divisão não são tarefas executadas pelo usuário.

Para cada etapa nos cenários criados, é criado um método correspondente dentro do arquivo de *steps*, chamado `StepsDefinition`. Uma parte do arquivo de *steps*

resultante da análise do processo em questão pode ser visto na Figura 6. Como pode ser visto neste código, para a etapa de cenário chamada *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("ˆI am on the task Decide if normal post or special shipment$")*. Os métodos podem ser preenchidos 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 6. Trecho do arquivo de *steps* resultante

Neste exemplo, apenas um processo foi analisado, mas arquivos BPMN podem conter vários processos. Caso seja necessário gerar os dados para todos os processos dentro de um arquivo, o arquivo contendo a tabela resultante será composto por várias tabelas, uma para cada processo analisado. Como é gerado o código de teste referente a todos os fluxos que contêm *Tasks* executadas por usuários, é possível executar o teste para todos os fluxos ou, por outro lado, também é possível utilizar a tabela gerada para verificar quais os fluxos mais importantes e utilizar os códigos gerados referentes apenas a esses fluxos.

Nesse trabalho, foi possível obter uma tabela importante para o teste dos processos, permitindo que sejam obtidas informações úteis para a execução de testes completos e com boa cobertura. As informações geradas também podem auxiliar na análise e a reduzir o tempo necessário para efetuar os testes. Como um exemplo de utilização dos dados obtidos, foram gerados dois arquivos essenciais para o teste funcional automatizado de aplicações BPM: o arquivo com os cenários de teste e o arquivo com os métodos para executar cada etapa dos cenários.

6. Conclusão

Esse trabalho buscou contribuir para abreviar o esforço de criação de testes funcionais automatizado de aplicações de BPM. Foi possível obter uma tabela com todos fluxos possíveis dentro de um dado processo. A criação da tabela possibilita a criação de testes mais completos e com boa cobertura. A tabela gerada também permite que sejam feitas análises importantes sobre o processo como, por exemplo, identificar gargalos.

Além da obtenção da própria tabela, a abordagem permite que as informações sejam usadas como entrada na criação de outros elementos. As informações podem ser reutilizadas das mais diversas formas, dependendo de qual estratégia/ferramenta de teste se deseja utilizar. Nesse trabalho decidiu-se por abordar o teste funcional com as ferramentas Selenium e Cucumber. Em um trabalho anterior, a execução do teste funcional automatizado em aplicações BPM com as ferramentas mencionados se mostrou promissora, mas trabalhosa. A abordagem proposta auxilia nesse ponto, permitindo que se utilizem as informações obtidas para gerar códigos necessários para a execução dos testes, auxiliando na automação destes.

A criação automatizada da tabela e dos dois arquivos auxilia a criação dos testes para os sistemas BPM, diminuindo o tempo de análise necessário para a criação dos elementos para o teste dos processos bem como tornando o teste mais completo, pois verifica todos os fluxos que um processo pode seguir. Assim, a qualidade e a cobertura dos testes também pode ser melhorada pois todos os fluxos e cenários possíveis são analisados. Por consequência, a qualidade dos sistemas também pode ser beneficiada com um processo de teste mais completo.

Como trabalhos futuros, tem-se a melhoria de alguns aspectos técnicos da abordagem como melhorar, por exemplo, o tratamento a processos com recursão e reconhecimento dos diferentes nomes de *tags*. Também podem ser avaliadas outras formas de utilizar as informações extraídas dos modelos BPMN, de forma a gerar outros elementos.

Referências

ABPMP (2012). *Guide to the Business Process Management Body of Knowledge (BPM CBOK)*. Association of Business Process Management Professionals, 2nd edition.

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.

Böhmer, K. and Rinderle-Ma, S. (2015). A genetic algorithm for automatic business process test case selection. In *On the Move to Meaningful Internet Systems: OTM 2015 Conferences*, pages 166–184. Springer.

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 (2015a). Cucumber-JVM Reference.

Cucumber Limited (2015b). Gherkin Reference.

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.

Delamaro, M. E., Maldonado, J. C., and Jino, M. (2007). *Introdução ao Teste de Software*. Campus Elsevier.

Dustin, E., Garrett, T., and Gauf, B. (2009). *Implementing Automated Software Testing: How to Save Time and Lower Costs While Raising Quality*. Addison-Wesley Professional, 1st edition.

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

Garousic, V., Mesbahb, A., Betin-Canc, A., and Mirshokraie, S. (2013). A systematic mapping study of web application testing. *Information and Software Technology*, 55(8):1374–1396.

Geb (2015). Groovy Browser Automation.

Graham, D. and Fewster, M. (2012). *Experiences of Test Automation: Case Studies of Software Test Automation*. Addison-Wesley.

Kurz, M. (2016). Bpmn model interchange: The quest for interoperability. In *Proceedings of the 8th International Conference on Subject-oriented Business Process Management*, S-BPM '16, pages 6:1–6:10, New York, NY, USA. ACM.

Ly, L. T., Maggi, F. M., Montali, M., Rinderle-Ma, S., and van der Aalst, W. M. (2015). Compliance monitoring in business processes: Functionalities, application, and tool-support. *Information systems*, 54:209–234.

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.

OMG (2011). Business process model and notation (BPMN). Disponível em: http://www.omg.org/spec/BPMN/2.0.

Pinheiro, V. S. F., Valentim, N. M. C., and Vincenzi, A. M. R. (2015). Um comparativo na execução de testes manuais e testes de aceitação automatizados em uma aplicação web. In *XIV Simpósio Brasileiro de Qualidade de Software*, pages 260–267.

Rafi, D. M., Moses, K. R. K., Petersen, K., and Mäntylä, M. V. (2012). Benefits and limitations of automated software testing: Systematic literature review and practitioner survey. In *Proceedings of the 7th International Workshop on Automation of Software Test*, AST '12, pages 36–42, Piscataway, NJ, USA. IEEE Press.

Ramos, L. P. and Bessa, A. (2015). Uma abordagem de gestão e desenvolvimento de automatização de processos de negógio com apoio de bpms. In *XIV Simpósio Brasileiro de Qualidade de Software*, pages 137–151.

Rayadurgam, S. and Heimdahl, M. P. E. (2001). Coverage based test-case generation using model checkers. In *Engineering of Computer Based Systems, 2001. ECBS 2001. Proceedings. Eighth Annual IEEE International Conference and Workshop on the*, pages 83–91. IEEE.

Selenium (2015). Selenium browser automation.

Smith, H. and Fingar, P. (2003). *Business Process Management: The Third Wave*. Meghan-Kiffer Press.

Sun, Y., Memmi, G., and Vignes, S. (2016). A model-based testing process for enhancing structural coverage in functional testing. In *Complex Systems Design & Management Asia*, pages 171–180. Springer.

Van der Aalst, W. (2013). Business process management: A comprehensive survey. *ISRN Software Engineering*, 2013(507984).

Van der Aalst, W., Adriansyah, A., and van Dongen, B. (2012). Replaying history on process models for conformance checking and performance analysis. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(2):182–192.

Watir (2015). Web Application Testing in Ruby.

Weske, M. (2012). *Business Process Management: Concepts, Languages, Architectures*. Springer, 2nd edition.

Wiklund, K., Sundmark, D., Eldh, S., and Lundvist, K. (2014). Impediments for automated testing – an empirical analysis of a user support discussion board. In *2014 IEEE Seventh International Conference on Software Testing, Verification and Validation*, pages 113–122.

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

Zhu, H., Hall, P. A., and May, J. H. (1997). Software unit test coverage and adequacy. *ACM Computing Surveys*, 29(4):366–427.