AutoFun: An Automated Model-based Functional Testing Tool

Anderson Domingues
Pontifícia Universidade Católica do Rio Grande do Sul
Av. Ipiranga, 6681 – Porto Alegre – RS – Brazil
anderson.domingues@acad.pucrs.br

Elder M. Rodrigues
Universidade Federal do Pampa - UNIPAMPA
Av. Tiaraju, 810 – Alegrete – RS – Brazil
elderrodrigues@unipampa.edu.br

Maicon Bernardino
Universidade Feevale
ERS–239, 2755 – Novo Hamburgo – RS – Brazil
bernardino@acm.org

ABSTRACT

Software testing is recognized as a fundamental activity for assuring software quality. However, software testing is one of the most time consuming and expensive activities of software development process. Software testing automation is an approach to increase testing effectiveness and to reduce testing effort and execution time. In this paper, we present AutoFun, a model-based functional testing tool developed in collaboration with two IT companies, that aims to support the automatic generation of both textual test cases and scripts from System Under Test (SUT) models. We also present an example of use in which our tool is applied to generated scripts in OpenScript format from UML models of a web application.

CCS Concepts

•**Software and its engineering** → **Software testing and debugging;**

1. INTRODUÇÃO

Teste de Software é reconhecido pela comunidade de engenharia de software como uma das atividades chaves para se garantir certo nível de qualidade a um software. Da mesma forma, a etapa de teste de software é reconhecida como uma das mais demoradas e caras executadas durante o processo de desenvolvimento de um software. Embora teste de software contribua para o aumento da qualidade de software, é importante a realização sistemática da atividade de teste, utilizando técnicas e critérios de teste bem definidos [10] [20]. Atualmente, o analista de teste pode lançar mão de uma variedade de técnicas de teste, critérios e ferramentas para apoiar e sistematizar o processo de teste em suas empresas. No entanto, muitas equipes de teste ainda executam o processo de teste de maneira *ad hoc*, levantando manualmente requisitos de teste a partir de documentos textuais de requisitos (*e.g.* documentos MD-50), escrevendo os requisitos de teste de forma textual e ambígua ou apenas criando modelos mentais do que precisa ser testado, para então criar casos de teste. Ainda, em alguns casos, os testadores precisam identificar quais requisitos precisam ser testados, apenas com base em informações resumidas que são repassadas em reuniões ou em rascunhos elaborados durante discussões com analistas de teste. Em um cenário adverso, testadores possuem à disposição os documentos previamente elaborados pelos analistas de teste, para então desenvolver os scripts de teste. Além disso, testadores precisam entender os casos de teste, testar manualmente ou escrever scripts de teste, arquivos de configuração de ferramentas e selecionar dados de entrada. Essas atividades, além de custosas são propensas a erros e falhas, podendo impactar na eficácia e eficiência do processo de teste de software.

Neste contexto, Teste Baseado em Modelo (TBM) [22] é uma abordagem que foi desenvolvida para apoiar e automatizar as atividades de teste, centralizando as informações de teste em modelos do sistema. A adoção de TBM apresenta várias vantagens para as equipes de teste, quando comparada com outras técnicas de teste. Por exemplo, o uso de modelos pode reduzir a possibilidade da interpretação errônea dos requisitos de teste e reduzir também a carga de trabalho dos analistas de teste e testadores, uma vez que a adoção de TBM pode possibilitar a geração automatizada dos casos de teste e scripts de teste. No entanto, para que uma equipe de teste consiga explorar todos os benefícios da adoção de TBM, o suporte de uma ferramenta é vital. Nos últimos anos diversas ferramentas de TBM foram propostas [5] [15], muitas destas focadas em dar suporte ao teste funcional.

A etapa de teste funcional pode tirar proveito da adoção de uma ferramenta TBM, uma vez que o uso de modelos pode auxiliar o testador a ter um melhor entendimento sobre as funcionalidades da aplicação a ser testada. Além disso, a adoção de uma ferramenta TBM que suporte a geração de scripts de teste ajuda a mitigar o gargalo na produtividade dos times de teste funcional. É importante destacar que, apesar de existirem diversas ferramentas para apoiar as atividades executadas durante a etapa de teste funcional, a maioria dessas ferramentas se baseia no uso da técnica de Capture & Replay (CR) [12].

Neste trabalho reporta-se a experiência de nossa equipe de projeto de pesquisa na implementação da ferramenta AutoFun para automação das atividades da etapa de teste funcio-

SAST, September 19-20, 2016, Maringa, Parana, Brazil

DOI: http://dx.doi.org/10.1145/2993288.2993298

nal no contexto de duas empresas de TI. Particularmente, o estudo descreve os requisitos básicos definidos pelas empresas, as decisões de projeto tomadas no desenvolvimento da ferramenta, bem como apresentar em detalhes as atividades suportadas pela ferramenta. Apesar de a ferramenta AutoFun ter sido desenvolvida com base no contexto específico de duas empresas, a ferramenta endereça requisitos que não são cobertas por outras ferramentas de código aberto e até mesmo por ferramentas comerciais.

Este trabalho está organizado como segue. Na Seção 2, apresentamos o referencial teórico sobre teste funcional, teste baseado em modelos e o contexto de desenvolvimento da ferramenta AutoFun. Na Seção 3, apresenta-se os requisitos da ferramenta proposta. As decisões de projeto são apresentadas na Seção 4, seguidas pelo detalhamento da ferramenta, na Seção 5. Na Seção 6, ilustra-se o funcionamento de ferramenta proposta por meio de um exemplo de uso. Finalmente, na Seção 7, apresenta-se a conclusão e trabalhos futuros, respectivamente.

2. REFERENCIAL TEÓRICO

Nesta seção, apresentamos o referencial teórico sobre teste funcional, teste baseado em modelos e o contexto de desenvolvimento da ferramenta.

2.1 Teste Funcional

Teste funcional, também chamado de teste caixa-preta, é uma técnica de teste centrada na derivação de casos de teste ou scripts de teste a partir da especificação externa do SUT, tais como requisitos e documentos de projeto [9]. Seu objetivo é induzir o sistema a se desviar do seu comportamento esperado, através da submissão de entrada externas. Assim, o objetivo do teste funcional é validar o comportamento do sistema com os resultados esperado descritos em seus requisitos funcionais [1]. Estes requisitos são, usualmente, reescritos ou derivados na forma de casos de teste, onde cada caso de teste representa normalmente um fluxo de execução do sistema. Cada caso de teste pode ser executado manualmente por um testador ou utilizado como entrada para ferramentas de automação de teste, compondo um ou mais planos de teste. Ferramentas como *Microsoft Test Manager* (MTM) [18], possuem interface de usuário que permite a inserção manual de casos de teste. Algumas outras, como por exemplo OpenScript, parte do pacote *Oracle Test Suite* (OATS) [21], se baseiam na técnica de *Capture & Replay* (CR) [12], coletando informações das ações dos usuário e suas interações com os objetos da interface, em uma determinada sessão. Neste caso, a sessão é gravada pela módulo de captura (*Capture*) da ferramenta e pode, posteriormente, ser executada de forma automatizada através do módulo de execução (*Replay*).

Além de simular a interação de um usuário com o sistema, um script de teste pode ser parametrizado, *i.e.*, ter seus parâmetros substituídos por variáveis ou *placeholders*, com a finalidade de exercitar diferentes padrões de dados de entrada. Por exemplo, um script escrito especificamente para validar o login de usuário em um sistema, pode ter os campos o nome de usuário e senha parametrizados, possibilitando testar o acesso de uma série de usuários apenas submetendo diferentes combinações de usuários e senhas à ferramenta de teste. Para que a parametrização dos scripts com diferentes dados de entrada tenha sucesso, é necessário que seja também realizada, em alguns casos, a correlação entre os dados de diferentes campos, como por exemplo a correlação de um nome de usuário a sua senha.

Existem ferramentas que apoiam, com diferentes graus de automação, a geração de scripts para teste funcional. Geralmente, essas ferramentas são desenvolvidas com base nas técnicas de *Capture & Replay* (CR) ou Teste Baseado em Modelos (TBM). Algumas destas ferramentas podem requerer um analista de teste para criar casos de teste com base no documento de requisitos do sistema. Outras, podem apenas utilizar os modelos do sistema para a geração de scripts de teste. Embora exista esta diferença entre as técnicas de CR e TBM, poucas ferramentas exploram ambas, ou até mesmo a interoperabilidade entre estas.

2.2 Teste Baseado em Modelos

Teste baseado em modelo (TBM) é uma técnica usada para apoiar a geração automática de casos de teste/scripts a partir de modelos do SUT [22]. Reutilizar ou criar modelos do sistema para representar uma abstração do seu comportamento e dados, é uma abordagem promissora para verificar propriedades relacionadas à qualidade. Isto se deve ao fato de que modelos normalmente promovem um melhor entendimento do sistema e possibilitam que casos de teste sejam gerados durante a etapa de desenvolvimento do sistema [11]. Portanto, com intuito de se beneficiar das vantagens na utilização da técnica TBM, o analista de teste deve projetar ou reutilizar alguns modelos do sistema, nos quais devem ser adicionadas informações de teste funcional (*e.g.* dados de entrada e saída esperada) e, então, utilizar uma ferramenta de suporte a TBM que recebe como entrada os modelos de teste para posterior geração de scripts de teste.

Apesar dos benefícios propiciados pela adoção de ferramentas que suportem ferramentas TBM, nas últimas décadas, a maioria das ferramentas de teste funcional comerciais utiliza a técnica CR, como por exemplo as ferramentas HPE Unified Functional Testing (UFT) [17] – antigamente conhecido por QuickTest Professional (QTP) –, Microsoft Visual Studio [19], IBM Rational Functional Tester [6], Selenium [16], OpenScript [21] e Microsoft Test Manager (MTM) [18]. No entanto, somente nos últimos anos surgiram ferramentas TBM para dar suporte a geração de casos de teste e scripts, tais como as ferramentas FOKUS!MBT [14], TEMA [2] e QuickCheck [7].

2.3 Contexto

A ferramenta AutoFun foi desenvolvida em colaboração com duas empresas da área de TI. Nessa colaboração, nosso grupo de pesquisa teve como objetivo principal investigar formas inovadoras para mitigar o esforço despendido na escrita dos casos de teste e na geração dos scripts de teste funcional. Adicionalmente, uma das empresas parceiras utilizava diversas ferramentas para apoiar seu processo de teste. Em caso de substituição de uma dessas ferramentas, o impacto na produtividade da equipe era considerável, pois a substituição de uma ferramenta exigia treinamento de diversos membros das equipes, bem como em muitos casos, a refatoração de milhares de casos de teste. Por este motivo, propomos aos nossos parceiros o desenvolvimento de uma ferramenta cujo objetivo principal seria reutilizar artefatos, especificamente modelos, já desenvolvidos pelas equipes de desenvolvimento, e cujas ferramentas de modelagem fossem bem conhecidas pela equipe. Além disso, a ferramenta deveria possuir uma arquitetura modular, que facilitasse a in-

tegração com outras ferramentas de teste. Desta forma, mudanças nas tecnologias de teste não teriam impacto significativo na equipe de teste pois analistas e testadores interagem a maior parte de tempo com a ferramenta de modelagem, e não possuem acesso a detalhes internos de diferentes ferramenta de teste, *e.g.*, arquivos de código fonte dos scripts, arquivos de configuração, entre outros.

3. REQUISITOS

Com o objetivo de definir as funcionalidades básicas de ferramentas de Teste Baseado em Modelos, foi realizado um mapeamento sistemático de literatura [8]. Como resultado desse trabalho, mapeou-se as diversas ferramentas disponíveis, bem como seus modelos de entrada, técnicas de teste, domínios de aplicação e atividades TBM. Além disso, avaliou-se com duas empresas parceiras quais seriam os requisitos de uma ferramenta TBM para a automação de suas atividades de teste. Com base nas necessidades das empresas e neste estudo, os seguintes requisitos básicos foram identificados:

RQ1) *A ferramenta deve fazer uso de modelos de entrada que minimizem os esforços de treinamento das equipes de projeto, desenvolvimento e teste no uso de diferentes notações e ferramentas de modelagem.*

Uma preocupação por parte da empresa quanto a adoção de uma ferramenta de TBM é que a adoção de ferramentas de prateleira, normalmente, leva ao uso de modelos e ferramentas de modelagem que ainda não são utilizados nas fases de projeto e documentação do software. Além disso, a adoção de diferentes modelos requer, consequentemente, a adoção de um ambiente de modelagem específico, o que por sua vez se traduz em um aumento de custos com licenças de software e treinamento. Por estes motivos, é desejável que a ferramenta tenha suporte aos modelos de entrada que são normalmente utilizados em outras etapas do processo de desenvolvimento, ao invés de um modelo específico de teste.

RQ2) *A ferramenta deve prover suporte a geração automatizada dos modelos de teste.*

As vantagens relacionadas a adoção de uma ferramenta TBM são normalmente maiores quando se reutilizam os modelos já desenvolvidos durante a etapa de projeto. No entanto, em muitas situações, esses modelos não se encontram imediatamente disponíveis, requerendo que a equipe de teste construa os modelos de teste a partir dos requisitos de teste levantados na especificação. Como a atividade de modelagem requer um esforço razoável e é propensas a erros, é mandatório que a ferramenta possibilite a geração dos modelos a partir da interação do testador com o sistema.

RQ3) *A ferramenta deve gerar os casos e scripts de teste.*

A ferramenta deve ser flexível, possibilitando ao analista de teste escolher entre a geração de casos de teste em formato texto ou gerar scripts de teste instrumentalizados - com os dados necessários para a execução. No entanto, a geração de scripts de teste executáveis é uma tarefa trabalhosa, por necessitar modelar detalhadamente e instrumentalizar os mesmos. Por este motivo, quando o número de casos de teste é pequeno (por exemplo, quando precisa se testar poucos casos de teste de regressão), pode não ser vantajoso trabalhar com a geração automatizada de scripts, mas sim apenas com a geração de casos de teste textuais.

RQ4) *A ferramenta deve permitir a correlação de dados de teste funcional.*

A ferramenta deve permitir a correlação de dados de teste aos parâmetros de dados associados aos modelos de teste. Por exemplo, muitas vezes o sistema deve explorar o teste de diferentes perfis de usuários para avaliar o comportamento funcional de um mesmo cenário de teste. Neste caso, deve-se gerar scripts de teste parametrizados, em que arquivos de dados externos, como por exemplo um arquivo CSV (*Comma-Separated Values*) ou TXT. Além disso, a ferramenta deve ser capaz de selecionar os dados de entrada, utilizando técnicas de seleção de dados, correlacionados aos parâmetros de dados do modelo de teste.

RQ5) *A ferramenta deve suportar o uso de diferentes técnicas de seleção de dados de teste.*

É fundamental que a ferramenta implemente uma dentre as diversas técnicas de seleção de dados de entrada, para a correlação dos dados de teste. Assim, a ferramenta deve permitir o uso de uma técnica de seleção (*e.g.*, randômico, sequencial, único) para cada campo de dados, bem como em situações onde existam dependências entre os dados informados em diferentes campos.

RQ6) *A ferramenta deve gerar scripts de teste em formato utilizado por ferramentas de teste funcionais comumente adotadas pela indústria*

A ferramenta deve gerar scripts de teste no formato utilizado por ferramentas de teste funcional disponíveis no mercado. Minimanente, a ferramenta deve ser projetada para gerar scripts para as ferramentas que são atualmente utilizadas pelas equipes de teste das empresas, mas a fácil inclusão de suporte a diferentes formatos de scripts é um requisito desejável.

4. DECISÕES DE PROJETO

Baseado na lista de requisitos básicos, levantados juntamente com as empresas parceiras, foram discutidas as decisões de projeto para o desenvolvimento da ferramenta da teste funcional AutoFun. Assim, nesta seção serão discutidas cada das decisões de projeto (DD) tomadas durante o desenvolvimento da ferramenta, com o objetivo de atender aos requisitos descritos na Seção 3.

DD1) *Desenvolver a ferramenta utilizando uma arquitetura modular* (RQ1, RQ6)

Para facilitar a adição de novas funcionalidades, como por exemplo o suporte a diferentes formatos de modelos de entrada, bem como a geração de scripts de teste para diferentes ferramentas foi decidido que a ferramenta deveria ter uma arquitura modular, baseada em componentes. Desta forma, a inclusão de uma nova funcionalidade (um novo formato de modelo ou de script de teste) implica apenas na substituição dos componentes relacionados a essas funcionalidades, simplificando a complexidade de evolução do software.

DD2) *Utilizar modelos UML para a modelagem das informações de teste* (RQ1)

Como as duas empresas fazem uso de modelos UML durante as demais etapas do processo de desenvolvimento, foi decidido utilizar uma abordagem que permitisse o reuso de alguns desses modelos. Vale ressaltar que a UML é um padrão de modelagem de análise e projeto de sistemas muito utilizado atualmente pela indústria. Isto também corrobo-

rou para que UML fosse adotada como padrão de entrada para a ferramenta. Além disso, o fato de a equipe de desenvolvimento já possuir *expertise* no uso da linguagem e na utilização do ambiente de modelagem reduz o custo com licenças e treinamentos. Destaca-se ainda que grande parte das iterações do testador será com a ferramenta de modelagem UML e poucas iterações com a ferramenta de teste, reduzindo assim a curva de aprendizado no uso da ferramenta.

DD3) *Realizar engenharia reversa nos scripts da ferramenta OATS para gerar, automaticamente, os modelos de teste* (RQ2)

A modelagem de diagramas UML é uma etapa que consome tempo e esforço do testador. Assim, é necessário implementar a funcionalidade de engenharia reversa dos scripts descritos em linguagem Java, utilizados em conjunto com OpenScript (OATS). Como o OpenScript é uma das ferramentas de teste funcional adotadas por uma das empresas, existe um repositório com centenas de scripts de teste. Assim, a criação manual desses modelos exigiria um grande esforço por parte da equipe de teste. Além disso, caso a equipe de teste precise criar os modelos de uma aplicação a partir do zero, pode ser mais produtivo utilizar o módulo de captura do OpenScript para se gravar as interações do testador com a aplicação, e então, utilizar a funcionalidade de engenharia reversa para gerar os modelos de teste.

DD4) *Gerar casos de teste textuais e scripts para a ferramenta OATS* (RQ3, RQ6)

Geração automatizada de scripts de teste é um dos principais requisitos de uma ferramenta de teste. Dado o contexto do projeto, optou-se por dar suporte a geração de scripts para as ferramentas já utilizadas pelas equipes de teste. Assim, a ferramenta AutoFun é utilizada para a geração dos scripts, mas a execução dos mesmos é realizada pelo motor da ferramenta de teste OpenScript. Entretanto, como em algumas situações a execução do teste deve ser executada manualmente, optamos também por gerar os casos de teste de forma textual.

DD5) *Utilizar arquivos CSV para a correlação de dados* (RQ3, RQ4)

A correlação de dados na ferramenta deve ser realizada fazendo uso de arquivos com a extensão CSV, que é um formato utilizado em diversas ferramentas de teste, facilitando assim o reuso de arquivos de dados de teste.

DD6) *Utilizar as técnicas randômico, sequencial e único para a seleção de dados* (RQ5)

A ferramenta suporta a definição das técnicas randômico, sequencial e único para a seleção dos dados de teste a partir de um arquivo CSV.

5. AUTOFUN - UMA FERRAMENTA PARA TESTE FUNCIONAL BASEADO EM MODELOS

A ferramenta AutoFun foi desenvolvida com o objetivo de prover suporte a automatização usando a abordagem TBM e a sua integração a uma ferramenta comercial de teste funcional. Além disso, a AutoFun pode ser facilmente estendida para apoiar a geração de casos de teste para diversas ferramentas de testes funcionais, tais como *Microsoft Test Manager* (MTM) [18] para teste manual e *Selenium* [16] para teste automatizado. A Figura 1 descreve uma proposta de processo de teste de funcional baseado em modelos. Este processo é composto por oito atividades principais, realizada por um perfil (*Testador Funcional*), que interage com três ferramentas: Astah [3], AutoFun e OpenScript [21].

O processo de teste funcional baseado em modelos proposto pode ser iniciado de duas maneiras. A primeira, por meio da atividade *Gravar Arquivo OpenScript*, permite ao testador gravar um script no formato *OpenScript*, utilizando o módulo de *Capturing*, da ferramenta *OpenScript*. Isto é eficaz quando a equipe de teste precisa realizar testes de regressão ou retestar módulos do sistemas provenientes de uma nova funcionalidade adicionada ou alterada. De outra forma, a atividade *Projetar Modelos do SUT* permite a equipe de teste se engajar no projeto de software desde os primeiros estágios de desenvolvimento. Assim, os modelos são desenvolvidos nas etapas de análise e projeto, não sendo necessário o SUT estar pronto.

5.1 Projetar os Modelos do SUT

Projetar os Modelos do SUT é a atividade responsável na modelagem da especificação de *Requisitos* para geração dos artefatos de teste. Nesta atividade, diagramas UML são projetados para suportar nas atividades seguintes a geração de scripts para a ferramenta OpenScript. Estes modelos incluem dados de teste para imitar o comportamento de usuários na sua interação com o SUT.

Esta atividade é baseada em dois diagramas UML amplamente utilizados pelas empresas: Caso de Uso (UC) e Diagrama de Atividades (AD). Um diagrama de UC pode ser mapeado para um cenário de teste de funcional, em que cada elemento *ator* representa um perfil de usuário e cada elemento de *caso de uso* descreve uma funcionalidade completa do sistema realizada por um ou mais atores. É importante destacar que um elemento de *caso de uso* descreve um comportamento detalhado, que é decomposto em um diagrama de atividades correspondente. Este diagrama é usado para modelar a interação dinâmica dos usuários com um sistema, representando-o como uma sequência de ações. Além disso, um ou vários casos de teste podem ser derivados a partir de um único AD. Normalmente, um AD com um fluxo linear de ações resultará em um único caso de teste. No caso de um AD com fluxos alternativos, vários casos de teste podem ser gerados. O número de fluxos alternativos está relacionado com o número de elementos de decisão incluídos no AD. Além disso, poderia ser útil se esses modelos fossem desenhados por analista de negócios ou analista de sistema e fossem entregues às equipes de teste. Assim, o custo para a concepção do modelo de teste poderia ser distribuído por todas as fases do processo de desenvolvimento de software e, ao mesmo tempo, melhoraria a qualidade da especificação do sistema.

5.2 Gravar Arquivo OpenScript

Em outra perspectiva, se o SUT já estiver sido implementado, é possível gravar um arquivo OpenScript, manualmente, executando o módulo de gravaçãp (*capturing*) da ferramenta OpenScript. O perfil *Testador Funcional* executa as interações com o SUT para imitar o comportamento dos usuários em diferentes cenários (casos de teste). O resultado desta atividade é a geração de um arquivo *OpenScript Padrão*, o qual é utilizado como entrada para a próxima atividade.

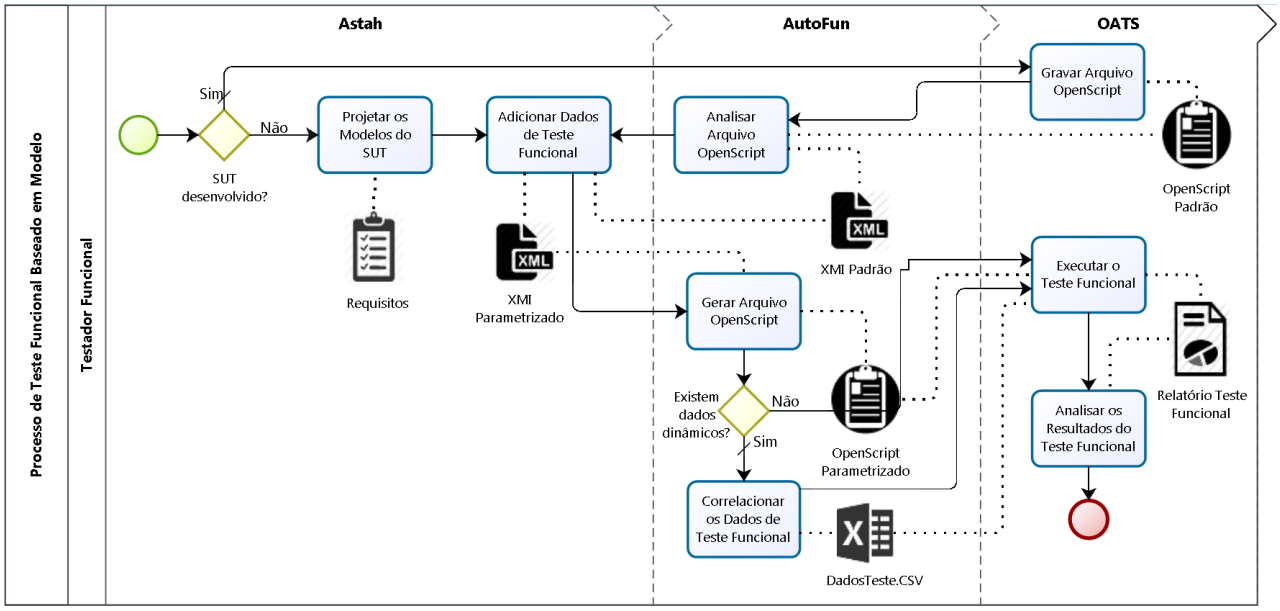

Figura 1: Processo de Teste Funcional Baseado em Modelos

5.3 Analisar Arquivo OpenScript

Esta atividade é executada usando a ferramenta AutoFun, em que os dados são analisados pela ferramenta a partir do arquivo em formato *OpenScript* gerado. Este processamento tem como resultado a geração de um arquivo *XMI Padrão*, o qual contém os dados de testes funcionais extraídos do script analisado. Este arquivo XMI gerado servirá de entrada para a atividade seguinte.

5.4 Adicionar Dados de Teste Funcional

Uma vez criado o arquivo *XMI Padrão* que contém a especificação XML do modelo de teste do SUT, esta atividade consiste em agregar dados de teste funcional nos diagramas UC e AD. Para este propósito, usam-se estereótipos e anotações de teste funcional tendo como base o perfil UML *Testing Profile* (UTP) [4]. O uso desta abordagem permite descrever os dados de testes funcionais necessários para gerar cenários e scripts de teste. Estes dados de testes são anotados em oito anotações distribuídas em oito estereótipos, tal como segue:

TDOBJECT: Indica o objeto utilizado na transição selecionada, nesta anotação também deverá ser indicado o nome do objeto de acordo com a sintaxe a seguir: Sintaxe: {*ObjectType; ObjectName*}; Exemplo de uso: {*link;web_a_Agreements*}. Aqui, web_a_Agreements é um objeto do tipo link.

TDACTION: Indica a ação a ser executada pelo objeto informado na anotação TDOBJECT. Também poderão ser informados parâmetros para execução da ação, caso necessário. Caso exista mais de uma ação para o objeto, estas deverão ser separadas por ","(vírgula), de acordo com a sintaxe básica definida abaixo: Sintaxe: {*Action_0; Param_a0*}; Exemplo de uso: {*click*}, uma ação sem passagem de parâmetros; {*waitForPage;null*}, uma ação com um parâmetro; {*click*}, {*pressTab*}, {*setText;textoexemplo*}, três ações definidas para o objeto.

TDPROTOCOL: Indica o tipo de aplicação que será utilizada pelo OpenScript, podendo ser Web ou Forms. Apesar de a ferramenta OpenScript suportar tanto *Web* quanto *Forms*, nossa ferramenta, por ora, gera apenas scripts para o protocolo *Web*. Sintaxe: *ProtocolType*; Exemplo de uso: *web ou forms*, indica o tipo de aplicação que será utilizada.

TDPROPERTIES: Indica o nome do arquivo *Object Library* utilizado para a construção do script OATS e referência de objetos. Sintaxe: *ObjectLibraryName*; Exemplo de uso: *TestGoogle*, nome do arquivo utilizado; *Error*, nome do arquivo utilizado.

TDWAIT: Poderá ser indicado o tempo de espera (*delay*) a ser aplicado sobre uma transição. Caso a anotação não seja definida no modelo, será então utilizado um valor padrão configurado na ferramenta AutoFun. Sintaxe: *ThinkTimeSeconds*; Exemplo de uso: *10.5*, definindo tempo de espera de 10.5 segundos.

TDVERIFY: Indica que um objeto deve ser validado antes de continuar a execução do script de teste. Será adicionada uma construção no script OpenScript que irá verificar a existência do objeto que foi definido na anotação TDOBJECT. O script somente seguirá a execução a partir do momento em que o objeto passar a existir. Sintaxe: *true*; Exemplo de uso: *true*, indica que a existência do objeto da transição, definido na anotação TDOBJECT que será verificada.

TDLOOPCONDITION: Deverá ser indicada a condição para execução de um laço. Um laço englobará um conjunto de ações definidas dentro de uma etapa. Uma etapa possui um conjunto de ações que serão executadas por um mesmo objeto. No modelo UML de um diagrama de atividades, um laço pode ser representado por um subdiagrama. Sintaxe: *LoopCondition*; Exemplo de uso: *forms.button == null*, indica que o laço

de repetição irá executar enquanto o objeto contiver o valor *null*.

TDITERATIONS: Deverá ser indicado o número de vezes que uma iteração ocorrerá. O número de iterações deverá ser definido na mesma transição em que um objeto é utilizado. Sintaxe: *IterationsTotal*; Exemplo de uso: *5*, será criado um laço com cinco iterações.

Como resultado desta atividade tem-se a geração de um arquivo *XMI Parametrizado*, o qual é utilizado como entrada para a próxima atividade.

5.5 Gerar Arquivo OpenScript

Os dados de teste funcional, armazenados em um arquivo *XMI Parametrizado* e gerados na atividade predecessora, são analisados e carregados em uma estrutura de dados em memória. Esta estrutura é utilizada para gerar scripts de teste funcional para a ferramenta OATS, ou seja, *OpenScript Parametrizado*. Neste contexto, esta atividade visa resolver os fluxos de atividades modelados nos diagramas UML transformando-os em casos de testes abstratos. Vale ressaltar, que é nesta etapa do processo em que os fluxos alternativos do AD são resolvidos gerando um conjunto de sequências de teste, os quais são instrumentalizados em casos de testes concretos, ou seja, scripts de teste funcional OpenScript.

Previamente, antes da realização deste processamento, a ferramenta AutoFun faz a validação do modelo de teste de entrada, de acordo com as regras de modelagem estabelecidas. a) Se as transições dos diagramas de atividades possuem as anotações obrigatórias. Caso contenha transições sem anotações, serão listadas na tela as transições e a qual diagrama que pertence com uma mensagem de erro. Caso tenha transições que não possuam uma anotação opcional será listada na tela as transições e a qual diagrama pertence com uma mensagem de aviso. Se na listagem houver pelo menos uma mensagem de erro, não será possível gerar os casos de teste. Todavia, havendo apenas mensagens de aviso, o processamento pode ser realizado. b) Se cada caso de uso possui um diagrama de atividades referenciado com o mesmo nome. Caso contrário, serão listados os casos de uso que não possuam uma referência para um diagrama de atividades.

5.6 Correlacionar Dados de Teste Funcional

Em algumas aplicações, existem parâmetros que são gerados de forma dinâmica, por exemplo, ID da sessão, que são criados pelo servidor para fins de segurança. Estes tipos de parâmetros devem ser correlacionados após a geração dos casos de teste. Além disso, o testador deve recuperar ou gerar esses dados. Em alguns casos, existem alguns provedores de aplicações que estão disponíveis em uma lista de parâmetros de dados correlacionados para a sua aplicação, o que facilita a geração de scripts de teste. Assim, nesta atividade o testador deve selecionar e associar, manualmente, os dados de teste (*DadasTeste.CSV*) para cada parâmetro identificado no script de teste *OpenScript Parametrizado*.

5.7 Executar Teste Funcional

A ferramenta OpenScript recebe como entrada um script de teste parametrizado, gerado na atividade *Gerar Arquivo OpenScript*, bem como carrega os dados de teste do arquivo *DadasTeste.CSV* associado na atividade *Correlacionar os Dados de Teste Funcional*. Uma vez interpretados estes arquivos pela ferramenta OATS é possível executar os testes funcionais do SUT modelado. Vale ressaltar que a análise de critérios de testes funcionais, tais como Particionamento em Classes de Equivalência, Análise do Valor Limite, Critérios Baseados em Fluxo de Controle, Critérios Baseados em Fluxo de Dados são realizadas pela própria ferramenta OATS. Ou seja, tais critérios não fazem parte do escopo das atividades da ferramenta AutoFun. Ao final da execução do teste, esta atividade tem como resultado o *Relatório de Teste Funcional*, o qual apresenta os resultados do teste consolidados por meio de gráficos e tabelas.

5.8 Analisar Resultados do Teste Funcional

Finalmente, a última atividade do processo são analisados os resultados da execução do teste funcional. O testador é responsável por verificar e validar se os requisitos funcionais estabelecidos para o SUT foram ou não atendidos com base nos resultados compilados no *Relatório de Teste Funcional*. Caso os testes falhem, o processo deve ser reiniciado, após a refatoração destes testes.

6. EXEMPLO DE USO: EPESI

Esta seção apresenta um exemplo de uso da ferramenta AutoFun, aplicando o processo de teste de funcional com TBM, como descrito na Seção 5. Neste exemplo, utilizamos uma solução de código-fonte aberto para CRM (*Customer Relationship Management*), ou seja, uma solução para gerenciamento de relacionamento com clientes, chamada EPESI. EPESI [13] é um *framework* PHP/Ajax que foi desenvolvido pela Telaxus LLC para fornecer um conjunto de ferramentas para o desenvolvimento rápido de aplicações *web*, integrado com banco de dados relacional, orientado para a gestão de informações empresariais. Sua finalidade é armazenar, organizar e compartilhar dados entre as pessoas de uma mesma empresa ou organização.

Para melhor entendimento do SUT, sumarizou-se as principais características da solução Web: a) Painéis customizáveis com Applets; b) Empresas - catálogo de contatos das empresas; c) Contatos - catálogo de contatos dos empregados; d) Calendário compartilhado com alertas; e) Lista de tarefas compartilhadas; f) Consulta de chamadas telefônicas; g) Gerenciamento de documentos - anotações e arquivos; h) Serviço de cliente de E-mail - Roundcube IMAP; i) Preenchimento de formulários exclusivos - Click2Fill; j) Controle e monitoramento de mudanças - Módulo Cão de Guarda; k) Sistema de permissões avançado; l) Caixas de mensagens.

6.1 Modelos de Teste - UML

Para que se possa demostrar o funcionamento da ferramenta AutoFun para este este exemplo de uso, precisou-se criar modelos UML para a solução. Utilizamos a ferramenta Astah [3], a qual possibilita a importação/exportação de arquivos no formato XMI – formato utilizado como entrada para a ferramenta AutoFun para a geração de scripts de teste. A Figura 2 mostra o diagrama de casos de uso (UC) da solução, que possui seis casos de uso e um ator (*User*). Neste diagrama, cada caso de uso está associado a um diagrama de atividades, que está anotado com dados de teste. Por razões de espaço, utiliza-se aqui apenas o caso de uso *Create Task*, que por sua vez está associado a um diagrama de atividades (AD), de mesmo nome, mostrado na Figura 3.

Existem algumas regras de modelagem que devem ser seguidas para que o arquivo XMI gerado pela ferramenta Astah seja corretamente interpretado pela ferramenta AutoFun. Estas regras também se estendem a outras ferramentas, desde que estas implementem, também, o padrão XMI. Estas regras são: a) Cada caso de uso deve possuir um *hyperlink* para um diagrama de atividades de mesmo nome; b) O diagrama de casos de uso deve possuir, pelo menos, um ator; c) O diagrama de casos de uso deve possuir, pelo menos, um caso de uso; d) Todo caso de uso deve estar associado, pelo menos, a um ator;

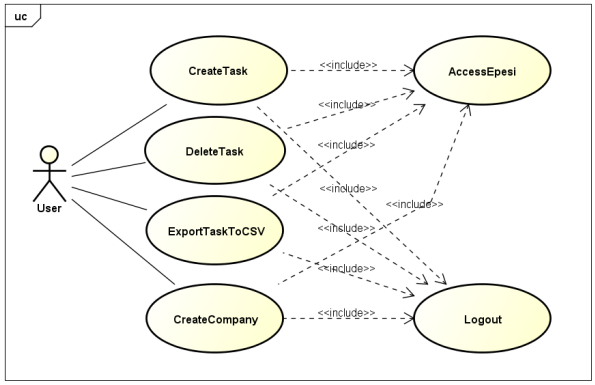

Figura 2: Diagrama de Casos de Uso do EPESI

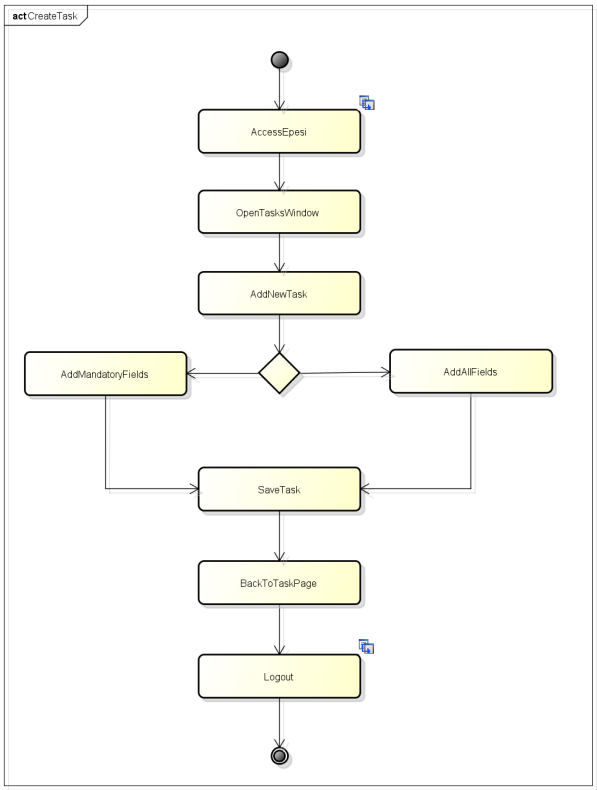

Figura 3: Diagrama de atividades *Create Task* do EPESI

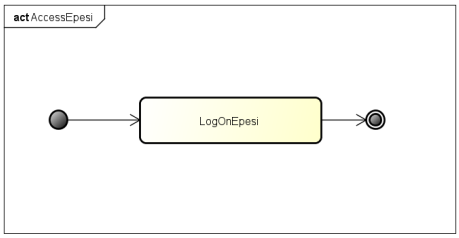

Figura 4: *AccessEpesi*

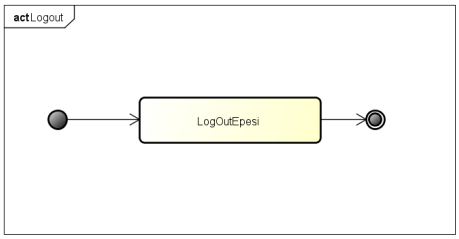

Figura 5: *Logout*

```
Caso de Teste 1:
  AccessEpesi -> OpenTaskWindow -> AddNewTask ->
  AddMandatoryFields -> SaveTask ->
  BackToTaskPage -> LogOutEpesi
Caso de Teste 2:
  AccessEpesi -> OpenTaskWindow -> AddNewTask ->
  AddAllFields -> SaveTask ->
  BackToTaskPage -> LogOutEpesi
```

Figura 6: Casos de teste abstratos derivados do caso de uso *Create Task*

O diagrama de atividades (AD), que é representado na Figura 3, contém dois fluxos para a adição de novas tarefas no sistema EPESI. O primeiro segue a atividade *AddMandatoryFields*, enquanto o segundo, e *AddAllFields*.

O diagrama possui oito atividades, sendo que duas delas contém *hyperlinks* para outros diagramas, aqui identificados como subdiagramas de atividades, são elas: *AccessEpesi* e *Logout*. Esta estratégia de modelagem é importante para otimizar o reuso de diagramas de atividades entre os diversos cenários de teste a serem modelados para o SUT. Por exemplo, a atividade *AccessEpesi* tem um *hyperlink* para o AD que esta representado na Figura 4, enquanto que *Logout* para o AD da Figura 5.

Vale a pena destacar que a ferramenta AutoFun resolve os fluxos alternativos dos diagramas de atividades no momento em que os traduz em casos de teste. Conforme exemplo da Figura 3, em que existem dois possíveis fluxos principais para se executar o caso de uso cadastrar uma tarefa (*Create Task*) são gerados. A Figura 6 apresenta a sequência de teste de cada um dos casos de testes abstratos gerados a partir do processamento do AD da Figura 3.

Além das regras descritas para o diagrama de casos de usos, os diagramas de atividades também contém regras de modelagem que devem ser atendidas. São elas: a) Toda transição deve conter uma anotação. As anotações de dados de teste são inseridas em todas as transições (arestas), exceto, nas transições que antecedem os seguintes elementos: uma atividade com *hyperlink*, ou seja, uma atividade que será decomposta em um subdiagrama; estado final; e os elementos *decision* ou *fork*. b) Todo AD deve possuir um estado inicial; c) Todo AD deve conter, pelo menos, um estado

final; d) Todo AD deve conter, pelo menos, uma atividade.

6.2 Geração OpenScripts OATS

Com o propósito de ilustrar parte do fluxo de atividades apresentado no AD da Figura 3, a Figura 7 apresenta a visão da *interface* gráfica da funcionalidade de adicionar uma nova tarefa (*Create Task*) ao sistema EPESI. Uma das alternativas do processo de geração dos casos de teste é a geração de sua representação no formato textual, a fim de que o testador possa usar estas informações para seleção e priorização de casos de teste quando aplicado ao teste de regressão. A Tabela 1 apresenta três colunas: Diagrama, Transição e Ação, Objeto e Valor. A coluna **Diagrama** identifica qual AD os dados de teste foram modelados. A coluna **Transição** identifica em qual das associações entre dois elementos UML as anotações foram adicionadas. Conforme descrito da Seção 5.4, os estereótipos e anotações são rotulados nos elementos de transição entre as atividades do AD. Na última coluna, formada pela tríplice informação de Ação, Objeto e Valor, representa, respectivamente as anotações **TDAction**, **TDObject** e **TDProperties**.

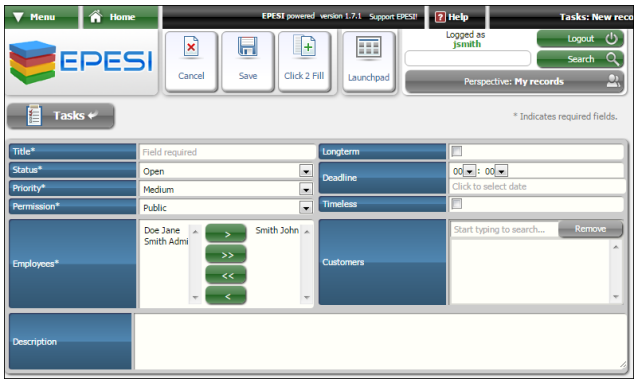

Figura 7: Tela do sistema EPESI para cadastrar uma nova tarefa (*Create Task*)

A Figura 8 demonstra um trecho de código OpenScript gerado usando a ferramenta AutoFun, gerando como resultado o *OpenScript Parametrizado*, tal qual descrito na atividade *Gerar Arquivo OpenScript* do processo de teste funcional baseado em modelos (Figura 3). O trecho apresentado instrumentaliza a atividade *AddMandatoryFields* apresentado no AD da Figura 3. Por exemplo, na linha 36 do script da Figura 8 é informado a string "Auto Flowers" que é submetido a uma busca dos possíveis clientes da tarefa. Na próxima atividade (linhas 37–40) é selecionado a opção "[Company] Auto Flowers LLC". Em seguida, o usuário informa a descrição da tarefa, "Need to start a new report". Por fim, o usuário clica no botão "Save" para persistir os dados e voltar para o menu de opções. Vale ressaltar, que os dados de teste informados no script exemplo, poderiam ser substituído por parâmetros que seriam correlacionados ao um arquivo de dados externo, *e.g.* *.CSV.

7. CONCLUSÃO E TRABALHOS FUTUROS

Este trabalho apresentou os requisitos básicos e as decisões de projeto, levantados e definidos em colaboração com empresas de TI, no contexto de uma proposta de desenvolvimento de uma ferramenta automatizada de teste funcional baseado em modelos. Com base nos necessidades dos nossos parceiros identificamos que as ferramentas existentes, sejam comercial ou *open-source*, não atendem a todos os requisitos identificados. Por este motivo apresentamos as nossas decisões de projeto no desenvolvimento de uma ferramenta de suporte a TBM que atende a todos esses requisitos. Este trabalha ainda apresenta as principais atividades suportadas pela ferramenta, bem como um exemplo de uso exemplificando a aplicação da mesma para a geração de casos de teste e scripts para testar uma aplicação web. Apesar do trabalho ter sido desenvolvido com base em um contexto específico, o mesmo apresenta uma contribuição para área de teste, em especial para as equipes de teste que estão identificando os requisitos necessários em uma ferramenta de TBM, seja para planejar o desenvolvimento de uma nova ferramenta a partir do zero ou adotar uma solução de prateleira. É importante destacar que os requisitos elicitados, decisões de projeto e soluções de modelagem apresentados podem ser facilmente reutilizadas no desenvolvimento de outras ferramentas TBM. Como trabalhos futuros, pretendemos dar continuidade ao desenvolvimento da ferramenta, incluindo novas funcionalidades como suporte a aplicações de área de trabalho ("*forms*"). Adicionalmente, estamos na etapa de planejamento de um estudo de caso a ser executado em um dos projetos de uma das empresa parceiras. Os resultados desse estudo irão nos ajudar a identificar o que precisa ser melhorado em novas versões da ferramenta. Adicionalmente, estamos discutindo com nossos parceiros a possibilidade de disponibilizar a ferramenta AutoFun sob uma licença *open-source*, o que permitiria que a mesma fosse utilizada e evoluída por outras empresas.

```
1
2  public class script extends IteratingVUserScript {
3    @ScriptService oracle.oats.scripting.modules.utilities.api.UtilitiesService utilities;
4    @ScriptService oracle.oats.scripting.modules.browser.api.BrowserService browser;
5    @ScriptService oracle.oats.scripting.modules.functionalTest.api.FunctionalTestService ft;
6    @ScriptService oracle.oats.scripting.modules.webdom.api.WebDomService web;
7
8    public void initialize() throws Exception {
9       browser.launch();
10   }
11
12   /**
13    * Add code to be executed each iteration for this virtual user.
14   */
15   public void run() throws Exception {
16     ... /*Snippet of code commented*/
17     beginStep("[10] Epesi - Tasks: New record (/index.php)", 0){
18       web.window(46,
19         "/web:window[@title='Epesi - Tasks: New record']")
20         .waitForPage(null);
21         {think(1.716);}
22       web.textBox(47,
23         "/web:window[@title='Epesi - Tasks: New record']
24          /web:document[@index='0']
25          /web:form[@id='libs_qf_fcdbf1881b6c710f39a1e3ff4246469c'
26          /web:input_text[@id='title'"")
27          .setText("Write a new report");
28          .pressTab();
29       web.textBox(48,
30          "...
31          /web:input_text[@name='customers_search'"")
32          .click();
33       web.textBox(49,
34          "...
35          /web:input_text[@name='customers_search']")
36          .setText("Auto Flowers");
37       web.element(50,
38          "...
39          /web:span[@text='[Company] Auto Flowers LLC']")
40          .click();
41       web.textArea(51,
42          "...
43          /web:textarea[@id='description'"")
44          .setText("Need to start a new report.");
45       web.element(52,
46          "...
47          /web:div[@text='\r\nSave ']")
48          .click();
49      }
50       endStep();
51      ...
52    }
53 }
```

Figura 8: Trecho de código fonte OpenScript gerado com base no AD da Figura 3

Tabela 1: Anotações adicionadas no diagrama de atividades da Figura 3

Diagrama	Transição	Ação → Objeto → Valor
AccessEpesi	Início → LogOnEpesi	navigate → "/web:window[@index='0']"→ "http://demo.epesibim.com/index.php" waitForPage → "/web:window[@index='0']"→ null click → "/web:input_submit[@name='submit_button']"→ null
CreateTask	AccessEpesi → OpenTaskWindow	waitForPage → "/web:window[@index='0']"→ null click → "/web:span[@text='Tasks']"→ null
CreateTask	OpenTasksWindow → AddNewTask	waitForPage → "/web:window[@index='0']"→ null click → "/web:div[@text='Add new']"→ null
CreateTask	AddNewTask (Decision Node) → AddMandatoryFields	waitForPage → "/web:window[@index='0']"→ null setText → "/web:input_text[@id='title']"→ "Write a new report" setText → "/web:input_text[@name='customers_search']"→ "Auto Flowers" click → "/web:span[@text='[Company] Auto Flowers LLC']"→ null setText → "/web:textarea[@id='description']"→ "Need to start a new report."
CreateTask	AddNewTask (Decision Node) → AddAllFields	waitForPage → "/web:window[@index='0']"→ null setText → "/web:input_text[@id='title']"→ "Write a new report" pressTab → "/web:input_text[@id='title']"→ null pressTab → "/web:select[@id='status' and multiple mod 'False']"→ null pressTab → "/web:select[@id='priority' and multiple mod 'False']"→ null pressTab → "/web:select[@id='permission' and multiple mod 'False']"→ null pressTab → "/web:input_checkbox[@id='longterm']"→ null pressTab → "/web:input_checkbox[@id='timeless']"→ null setText → "/web:input_text[@name='customers_search']"→ "Auto Flowers" click → "/web:span[@text='[Company] Auto Flowers LLC']"→ null setText → "/web:textarea[@id='description']"→ "Need to start a new report."
CreateTask	AddMandatoryFields → SaveTask	click → "/web:div[@text='\r\nSave ']"→ null
CreateTask	AddAllFields → SaveTask	click → "/web:div[@text='\r\nSave ']"→ null
CreateTask	SaveTask → BackToTaskPage	waitForPage → "/web:window[@index='0']"→ null
Logout	Início → LogOutEpesi	waitForPage →"/web:window[@index='0']"→ null click → "/web:span[@text='Logout']"→ null

8. REFERÊNCIAS

[1] B. Agarwal, M. Gupta, and S. Tayal. *Software Engineering And Testing: An Introduction (Computer Science)*. Jones & Bartlett Publishers, 2009.

[2] A. K. Antti Jääskeläinen, Heikki Virtanen. TEMA. Available in: <http://tema.cs.tut.fi>, 2016.

[3] Astah. Software Astah: Software Design Tools for Agile teams with UML, ER Diagram. Available in: <http://astah.net/>, 2016.

[4] P. Baker, Z. R. Dai, J. Grabowski, O. Haugen, I. Schieferdecker, and C. Williams. *Model-Driven Testing: Using the UML Testing Profile*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[5] C. Catal and D. Mishra. Test case prioritization: a systematic mapping study. *Software Quality Journal*, pages 445–478, 2013.

[6] D. Chadwick, C. Davis, M. Dunn, E. Jessee, A. Kofaldt, K. Mooney, R. Nicolas, A. Patel, J. Reinstrom, K. Siefkes, P. Silva, S. Ulrich, and W. Yeung. *Using Rational Performance Tester Version 7*. IBM Redbooks, 2008.

[7] K. Claessen and J. Hughes. Quickcheck: A lightweight tool for random testing of haskell programs. *SIGPLAN Notes*, pages 268–279, Sept. 2000.

[8] E. de Macedo Rodrigues. PLeTs - A Product Line of Model-based Testing Tools. Master's thesis, Tese (Doutorado em Ciência da Computação) - Faculdade de Informática - PUCRS, Porto Alegre, 2013.

[9] M. E. Delamaro, J. C. Maldonado, and M. Jino. *Introdução ao Teste de Software*. Elsevier, Rio de Janeiro, RJ, Brazil, 2007.

[10] R. DeMillo, R. Lipton, and F. Sayward. Hints on Test Data Selection: Help for the Practicing Programmer. *Computer*, 1:34–41, Apr 1978.

[11] A. C. Dias Neto, R. Subramanyan, M. Vieira, and G. H. Travassos. A Survey on Model-Based Testing Approaches: A Systematic Review. In *1st ACM International Workshop on Empirical Assessment of Software Engineering Languages and Technologies*, New York, NY, USA, 2007. ACM.

[12] O. El Ariss, D. Xu, S. Dandey, B. Vender, P. McClean, and B. Slator. A Systematic Capture and Replay Strategy for Testing Complex GUI Based Java Applications. In *7th International Conference on Information Technology: New Generations*, pages 1038–1043, apr. 2010.

[13] EPESI. Software EPESI: FREE open source CRM and PHP/Ajax framework. Available in: <http://www.epesi.org/>, 2016.

[14] Fraunhofer, Fokus. Software Fokus!MBT. Available in: <http://www.fokusmbt.com>, 2016.

[15] V. Garousi, A. Mesbah, A. Betin-Can, and S. Mirshokraie. A systematic mapping study of web application testing. *Information and Software Technology*, 55(8):1374–1396, 2013.

[16] A. Holmes and M. Kellogg. Automating Functional Tests Using Selenium. In *Proceedings of the Conference on AGILE 2006*, pages 270–275, Washington, DC, USA, 2006. IEEE Computer Society.

[17] HP®. HP Unified Functional Testing: GUI Testing Tutorial for Web Applications. Available in: <http://technodivine.com/downloads/Help-and-Guides/02-QTP/05-UFT-Documentation-12.02/04-UFT_GUI_Tutorial_Web.pdf>, 2016.

[18] Microsoft®. Microsoft Test Manager (MTM). Available in: <https://www.visualstudio.com/features/testing-tools-vs>, 2015.

[19] Microsoft®. Software Microsoft Visual Studio. Available in: http://www.visualstudio.com/, 2016.

[20] G. J. Myers and C. Sandler. *The Art of Software Testing*. Wiley, New York, NY, USA, 2004.

[21] Oracle®. Oracle Application Testing Suite (OATS). Available in: <http://www.oracle.com/technetwork/oem/app-test/etest-101273.html>, 2016.

[22] M. Utting and B. Legeard. *Practical Model-Based Testing: A Tools Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2006.