Test Script Diagram – Um modelo para geração de scripts de testes

Francisco Nauber B. Góis [2], Pedro Porfírio Muniz Farias[1], Rafael Braga Oliveira[2]

[1] Mestrado em Informática Aplicada – Universidade de Fortaleza (Unifor)
Caixa Postal 60811-905 – Fortaleza – CE – Brazil

[2]Serviço Federal de Processamento de Dados (Serpro) – Fortaleza, CE – Brazil

{francisco.gois,rafael.oliveira}@serpro.gov.br, porfirio@unifor.br

Abstract. *This article describes and presents the specifications of the Test Script Diagram (TSD). Each TSD is a model for the generation of test scripts that has use cases as input and associates test data grouped in equivalence classes to them. A tool called TestKase was developed in order to help the elaboration of the TSDs and to automate the generation of the test scripts. This article also presents results of the usage of the TSD, which were built with aid of TestKase in 66 systems. The TSD can be used after the requirements elicitation simultaneously with other phases of the system development.*

Resumo. *Este artigo descreve e apresenta as especificações do Test Script Diagram (TSD). Cada TSD é um modelo para geração de scripts de testes que utiliza como insumo casos de uso e associa a eles dados de testes agrupados em classes de equivalência. Para auxiliar a elaboração dos TSDs e automatizar a geração dos scripts de testes, foi construída uma ferramenta denominada TestKase. Este artigo apresenta também resultados da utilização do TSD, construídos com o auxílio do Testkase, em 66 sistemas. O TSD pode ser utilizado após a elicitação de requisitos em paralelo com outras fases do desenvolvimento do sistema.*

1. Introdução

A necessidade de se aprimorar a qualidade de produtos de software tem incentivado um contínuo aperfeiçoamento de atividades de testes. Sem prejuízo dos demais aspectos, pode-se afirmar que a qualidade de um software está correlacionada à qualidade dos testes aos quais é submetido.

Dentro da atividade de testes de software, é fundamental a geração de casos de testes. É necessário que os casos de testes proporcionem adequada cobertura, de modo a verificar, para um significativo conjunto de exemplos de dados de entrada, se os requisitos especificados foram devidamente implementados. Geralmente escritos sob a forma de casos de uso, esses requisitos são recursos utilizados pelos projetistas de testes para a construção dos casos de testes. Por essa razão, diversos trabalhos têm proposto derivar casos de testes a partir de casos de uso [Neto, 2008].

A geração manual de casos de testes com a adequada cobertura, independentemente da abordagem utilizada, é uma tarefa repetitiva, meticulosa e laboriosa, e, por isso, bastante suscetível a falhas. Alternativamente, a geração de casos de testes pode ser realizada automaticamente, promovendo maior agilidade e

confiabilidade ao processo. Uma abordagem usual, denominada Teste Baseado em Modelo (MBT, do inglês *Model Based Test*), preconiza a construção de um modelo que represente os requisitos da aplicação, e, a partir desse modelo, promove a geração sistemática de casos de testes [Heumann, 2001].

Casos de testes podem ser descritos por scripts de testes manuais. A Listagem 1 exibe um exemplo de script de teste. Tais scripts de testes são instruções em linguagem natural indicando ações a serem realizadas manualmente para validar funcionalmente o sistema. Os scripts podem ser utilizados por uma equipe diferente da equipe de desenvolvimento. A equipe independente de teste, geralmente, não tem acesso ao código-fonte da aplicação.

Listagem 1. Script de teste gerado a partir do TSD

```
Entre no Sistema
Digite o valor Pedro no campo Login
Digite o valor Teste no campo Senha
Selecione o menu Inserir Cadastro
Verifique se o formulário de cadastro é apresentado
Digite o campo Nome com valor Pedro
Digite o campo Telefone com o valor 9999-99-99
Digite o campo Correio com valor xxxxxx@yahoo.com.br
```

No presente estudo, apresenta-se um modelo gráfico para geração de scripts de testes a partir de casos de uso. O modelo utilizado foi denominado Diagrama de Scripts de Testes (TSD, do inglês *Test Script Diagram*). Foi construída ferramenta TestKase que facilita a criação do TSD e automatiza a geração dos scripts de testes.

A Figura 1 apresenta o processo de geração de casos de testes a partir do TSD. O processo se inicia (legenda ❶) com a geração de uma versão inicial dos TSDs a partir dos casos de uso. Os projetistas de testes editam graficamente o TSD (legenda ❷), agrupando passos em subdiagramas e especificando filtros e os dados de testes a serem utilizados. Finalmente, são gerados os scripts de testes a partir do TSD (legenda ❸). A partir do TSD, o TestKase também gera estimativa de esforço de teste (legenda ❹).

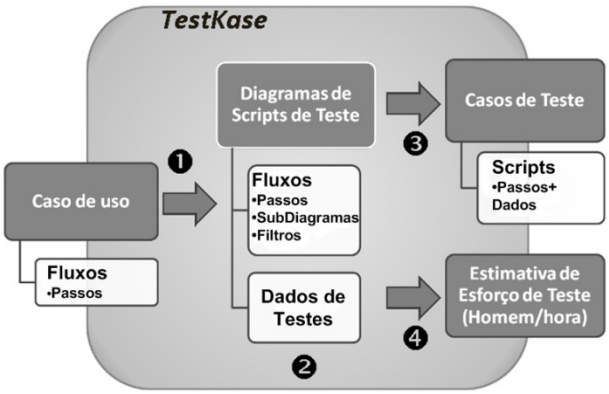

Figura 1. Processo de geração de casos de testes a partir do TSD

O restante deste estudo está organizado em seis seções. A seção 2 aborda o referencial teórico referente a testes baseado em modelo e a testes baseados em casos de uso. A terceira seção traz o Diagrama TSD e um algoritmo de geração de scripts de testes a partir do TSD. A quarta seção apresenta a experiência de uso do TSD no Serpro,

empresa de tecnologia da informação do governo federal. A quinta seção registra trabalhos relacionados, e a sexta e última reúne as principais conclusões do estudo.

2. Teste Baseado em Caso de Uso

Testes de software sempre utilizam modelos para o seu projeto ou a sua execução. Nos casos em que o modelo não é explícito, é implicitamente utilizado por projetistas para gerar casos de testes.

Testes baseados em modelo utilizam um modelo explícito, que representa os requisitos ou especificações da aplicação sob teste [Dalal, 1999; Hessel, 2007] e um algoritmo de navegação que percorre o modelo a partir de um critério de cobertura, gerando os casos de testes. Cobertura é a medida de completude de um conjunto de testes [Benjamin, 1999].

Dentre os modelos de testes mais utilizados, estão aqueles baseados em máquinas de estados finitos e os baseados em diagramas UML [Neto, 2008].

Casos de Uso é uma das formas mais utilizadas para capturar requisitos funcionais de um sistema [Fröhlich, 2000]. Um caso de uso modela funções esperadas do sistema e seu ambiente, podendo ser utilizado como fonte de informações para testes de sistemas [Rational, 2010].

Um caso de uso inclui, afora outros elementos, um conjunto de fluxos de eventos [Bittner, 2002], compreendendo um fluxo principal e suas possíveis variações, representadas pelos fluxos alternativos ou de exceção. Um fluxo de eventos é uma seqüência de passos que descreve o que um sistema deve fazer baseado no comportamento do usuário [Leffingwell, 2003].

Cenários são seqüências de passos [Welzer, 2002] que descrevem caminhos específicos através dos fluxos de eventos [Zielczynski , 2010].

Um teste baseado em cenários verifica quanto o sistema desenvolvido atende a essa situação [Kaner, 2003].

Diversos estudos têm sido propostos no sentido de derivar casos de testes a partir de casos de uso [Nogueira, 2007; Rocha, 2008]. A IBM propôs uma técnica de teste denominada Teste Baseado em Casos de Uso (UCBT, do inglês *Used Case Based Test*) [Abdurazik, 2000]. A implementação dessa técnica se dá em duas etapas. A primeira consiste em identificar os cenários, enquanto a segunda consiste em aplicar dados de entrada para cada cenário encontrado e associá-los a um resultado esperado.

Casos de testes baseados em caso de uso podem utilizar um critério para seleção dos dados de entrada. O diagrama proposto neste artigo seleciona de maneira sistemática os dados de entrada, utilizando um método denominado Particionamento de Classes de Equivalência. Esse método preconiza a divisão do domínio de entrada do programa em classes de dados. Cada classe, válida ou inválida, representa um conjunto de elementos que acarretam um mesmo comportamento do sistema, independentemente de qual dos elementos da classe tenha sido escolhido como entrada. Por isso, subentende-se que qualquer um dos elementos de uma classe de equivalência pode ser utilizado para testar o comportamento do sistema para todos os elementos da classe [Myers, 2004].

Na próxima seção são apresentados os elementos do diagrama TSD, sua forma de representação de dados e um algoritmo para geração de scripts de testes a partir de um TSD

3. Test Script Diagram (TSD)

A presente seção apresenta o TSD e propõe uma forma de representação dos dados de testes que possibilita associá-los às variáveis presentes nesse diagrama. Apresenta também um algoritmo para geração de scripts de testes a partir de um TSD modelado.

O TSD consiste em (1) uma representação gráfica de fluxos interligados de um caso de uso que podem ser associados aos seus respectivos dados de testes e (2) uma representação para modelar os dados associados aos fluxos.

O diagrama possui um conjunto inicial de elementos que podem ser extraídos a partir de uma especificação de casos de uso. Outros elementos e a associação de dados de testes são indicados pelo projetista do TSD com vistas a complementar as informações necessárias para geração de scripts de testes.

3.1. Elementos do TSD

O TSD possui elementos dos tipos Passo, Fluxo, Seta do Fluxo, Filtro, Subdiagrama e Loop. Alem desses elementos, o TSD possui um nome que o identifica.

A representação gráfica de cada um desses seis elementos inclui um atributo textual. Cada um desses atributos textuais é descrito por meio de uma gramática em formato BNF (Tabela 1).

Tabela 1. Gramática BNF que descreve os atributos textuais dos elementos do TSD

Atributo	Descrição
Expressão Descritiva do Passo	<EXPRESSÃO DESCRITIVA> ::= <VARIÁVEL> | <PALAVRA> | <VARIÁVEL><ESPAÇO BRANCO><EXPRESSÃO DESCRITIVA> | <PALAVRA><ESPAÇO BRANCO><EXPRESSÀO DESCRITIVA >
Conjunto de Classes de Equivalência do Filtro	<CONJUNTO CLASSES EQUIVALENCIA> ::= {<LISTA CLASSES >}
Variável	<VARIÁVEL> ::= %<IDENTIFICADOR_DA_VARIAVEL>
Identificador de uma Variável	<IDENTIFICADOR_DA_VARIÁVEL> ::= <PALAVRA> ‘,’ .<IDENTIFICADOR_DA_VARIAVEL> | <PALAVRA>
Lista de Classes de Equivalência	<LISTA CLASSES >::= <CLASSE > | <CLASSE>,<LISTA CLASSE>
Classe de Equivalência	<CLASSE> ::= <PALAVRA>
Nome do Diagrama	<NOME DO DIAGRAMA> ::= <PALAVRA>
Nome do Loop	<NOME DO LOOP> ::= <PALAVRA>
Espaço em Branco	<ESPAÇO BRANCO> ::= ‘ ’
Palavra	<PALAVRA> ::= <CARACTER> | <CARACTER><PALAVRA> |<PALAVRA>
Caractere	<CARACTERE> ::= A | ... | Z | a | ... | z|0| ... |9|-|_| %%| $ | &

3.1.1 Elemento Passo

O Passo descreve uma ação ou verificação de um caso de teste. Pode ser o ponto de partida ou o ponto de chegada de um elemento do tipo Loop. O elemento possui uma expressão descritiva que, em seu conteúdo, pode incluir variáveis utilizadas para associar os dados de testes aos scripts de testes a serem gerados.

É representado por um retângulo de linha contínua contendo a Expressão Descritiva do Passo. Seu conteúdo não pode ser vazio, devendo compreender palavras e variáveis. A Figura 2 mostra um exemplo da representação de dois passos. No primeiro passo a expressão descritiva contém a variável %nome.

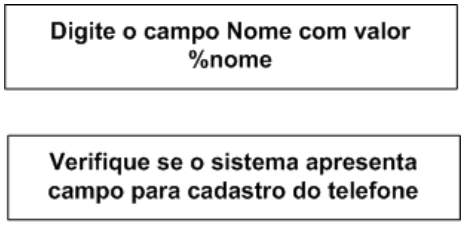

Figura 2. Exemplo de uso dos Passos de um TSD

O nome de uma variável, que não pode conter espaços, é iniciado pelo caractere "%". Quando for necessário expressar esse caractere dentro de uma expressão descritiva do passo, utiliza-se uma sequência de dois caracteres: "%%". No caso de variável composta, o identificador será composto por várias palavras ligadas por pontos, como, por exemplo, "%aluno.disciplina1.nota1".

3.1.2. Elemento Fluxo

O elemento Fluxo é formado por um conjunto de elementos dos tipos Passo, Subdiagrama, Filtro e Loop, sendo delimitado por um retângulo pontilhado. O elemento Fluxo possui o mesmo objetivo dos fluxos de um caso de uso. Dentro de um TSD, os fluxos são ordenados de acordo com a sua disposição seguindo a ordem da esquerda para direita. A Figura 3 mostra o exemplo de dois fluxos.

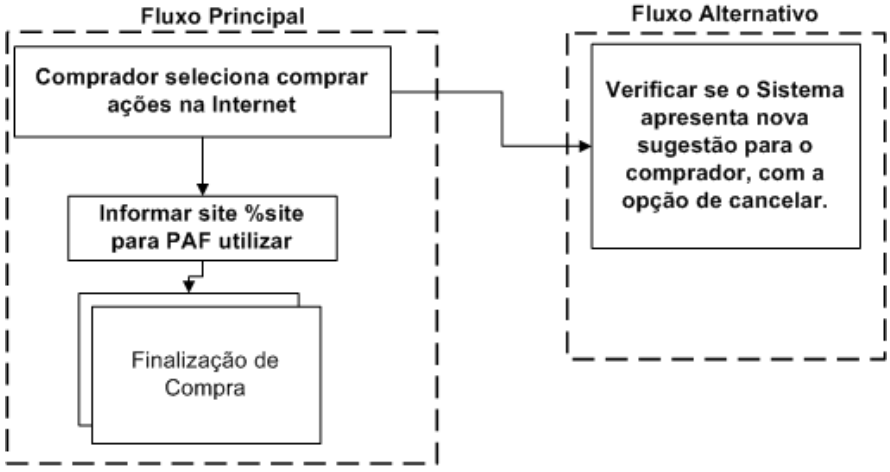

Figura 3. Exemplo de uso dos elementos Fluxo, Seta do Fluxo e Subdiagrama

3.1.3. Elemento Seta do Fluxo

Elemento do diagrama representado por uma seta de linha contínua, a Seta do Fluxo que indica a ordem das ações e verificações, representadas pelos Passos do TSD em um script de teste. A Seta do Fluxo, em um mesmo fluxo ou em fluxos distintos, pode ligar:

1. um passo a outro;
2. um passo a um subdiagrama;
3. um subdiagrama a um passo;
4. um subdiagrama a um subdiagrama;
5. um passo a um filtro;
6. um filtro a um passo.

3.1.4. Elemento Filtro

O Filtro seleciona entre os dados de testes aqueles pertencentes a uma das classes de equivalência a ele associadas. Conforme pode ser observado na Figura 4, o Filtro é representado por um triângulo com um dos vértices no sentido do fluxo e uma expressão descritiva que enumera as suas classes de equivalência. A sintaxe para essa expressão descritiva é apresentada no atributo "Conjunto de Classes de Equivalência do Filtro" da Tabela 1.

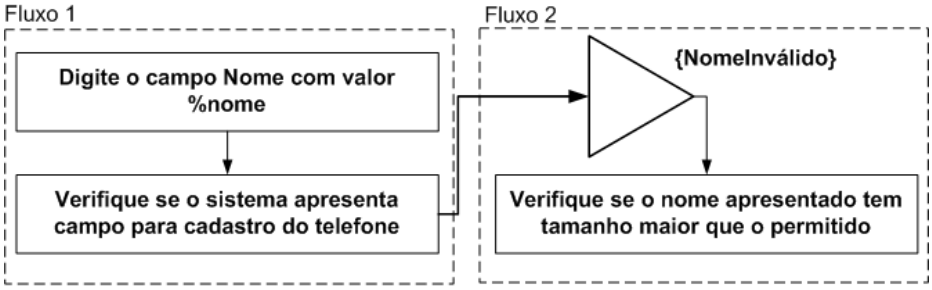

Figura 4. Exemplo de uso de um Filtro

A Figura 4 ilustra a utilização de um Filtro associado a uma única classe de equivalência, denominada "NomeInválido". Nesse caso, o Filtro foi incluído para que apenas dados pertencentes à classe "NomeInválido" sejam aplicados aos cenários que possuírem os Passos do segundo Fluxo. O Filtro é utilizado para restringir a aplicação dos dados, a partir do ponto onde ele é incluído.

3.1.5. Elemento Subdiagrama

Representado por dois retângulos sobrepostos, o Subdiagrama é o elemento do TSD que encapsula um outro diagrama. Possui uma expressão descritiva que indica o nome do diagrama referenciado. Tem por objetivo reutilizar diagramas previamente criados e reduzir a complexidade dos diagramas.

3.1.6. Elemento Loop

O elemento Loop é representado por uma seta pontilhada interligando um Passo do Fluxo a outro Passo que o antecede, criando uma estrutura de repetição. Possui um nome que o identifica conforme a gramática que o representa. A quantidade de repetição dos elementos do tipo Loop é governada pelos dados de testes representados na Seção 3.2.

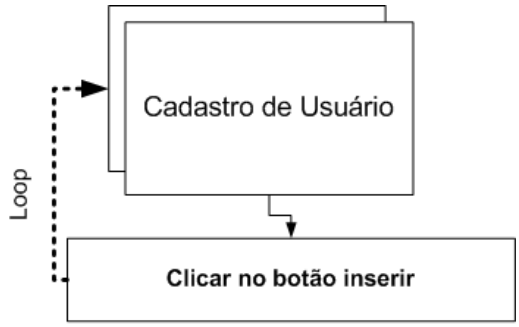

Figura 5. Exemplo de uso do elemento Loop

3.2 Forma de Representação dos Dados de Testes no TSD

Os dados de testes são associados a um TSD utilizando as variáveis presentes na expressão descritiva de cada Passo. Cada variável é substituída pelos seus respectivos valores, gerando um Passo em um script de teste.

O modelo de representação das variáveis é descrito a partir de um esquema XML (Arquivo XSD). O esquema se inicia com um elemento denominado Dados de Testes. Cada elemento Dados de Testes é formado por um conjunto de variáveis, que podem ser simples ou compostas.

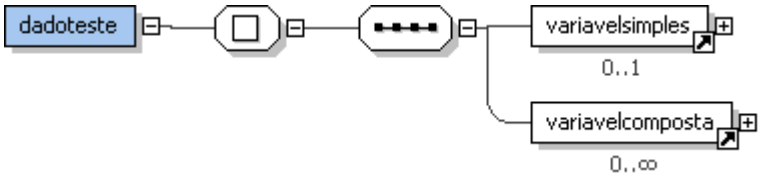

Figura 6. Diagrama lógico do elemento Dados de Testes

Somente Variáveis Simples podem ser referenciadas nos Passos do diagrama. Cada Variável Simples possui um nome e pode ter um ou mais elementos do tipo Classe de Equivalência e um elemento do tipo Valor. O elemento Classe de Equivalência possui um nome que indica a classe de equivalência de um ou mais valores utilizados. O elemento Valor é utilizado para substituir a referência da variável presente na expressão descritiva do diagrama.

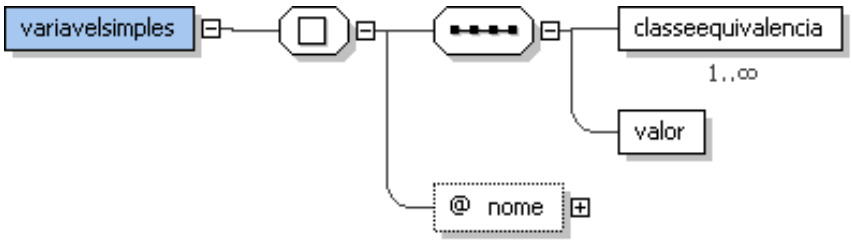

Figura 7. Diagrama Lógico do elemento Variável Simples

Uma Variável Composta possui um nome, um ou mais elementos do tipo Loop e um ou mais elementos dos tipos Variável Simples ou Variável Composta. A Variável Composta possibilita representar um agregado heterogêneo de elementos e pode ser utilizada para associar os valores de entrada aos valores de saída.

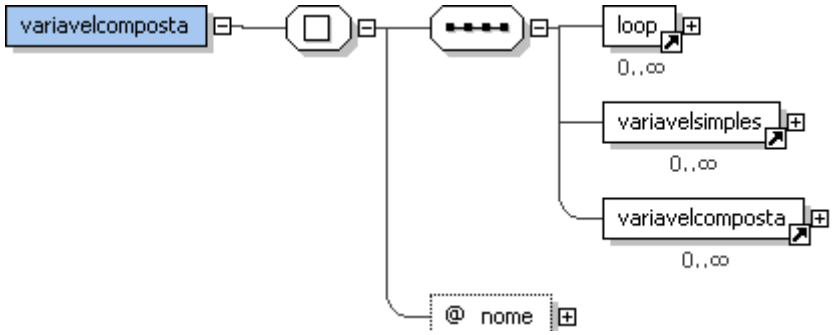

Figura 8. Diagrama lógico do elemento Variável Composta

O elemento do tipo Loop possui um nome e representa uma sequência de iterações. Cada elemento Iteração possui um conjunto de variáveis. O número de elementos Iteração presentes no Loop indica quantas vezes um elemento Passo deve ser repetido em um script de teste.

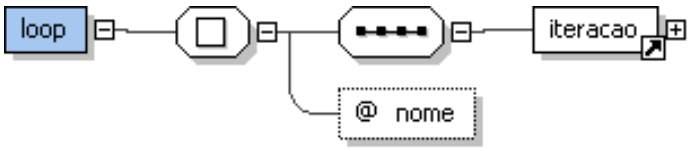

Figura 9. Diagrama lógico do esquema de definição dos dados de testes

De forma a exemplificar os dados de testes pertencentes a um Loop, a Listagem 2 apresenta a lista dos dados de entrada e saídas esperadas de um sistema que escolhe o maior entre vários números fornecidos.

Listagem 2. Representação de dados para o sistema que escolhe o maior número entre diversos números fornecidos

```
<?xml version="1.0" encoding="UTF-8"?>
<dadoteste xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="file: TSD.xsd">
   <variavelcomposta nome="sistemapegamaior">
      <loop nome="">
         <iteracao>
            <variavelsimples>
               <classeequivalencia>numerovalido</classeequivalencia>
               <valor>2</valor>
            </variavelsimples>
         </iteracao>
         <iteracao>
            <variavelsimples>
               <classeequivalencia>numerovalido</classeequivalencia>
               <valor>7</valor>
            </variavelsimples>
         </iteracao>
      </loop>
   </variavelcomposta>
   <variavelsimples nome="saidaesperada">
      <classeequivalencia>numerovalido</classeequivalencia>
      <valor>O maior número possui valor 7</valor>
   </variavelsimples>
</dadoteste>
```

3.3. Exemplo de Uso do TSD

Esta seção apresenta um exemplo de uso do TSD aplicado a um sistema que registra pedidos de compra. A Figura 10 ilustra os dados modelados para um pedido válido (classe de equivalência "C1EqPedidoValido") criado na data "12/02/2010" para o cliente "João". O pedido contém dois itens, Manteiga e Leite, cujos valores são, respectivamente, R$ 5,00 e R$ 2,00. O valor total esperado do pedido é R$ 7,00.

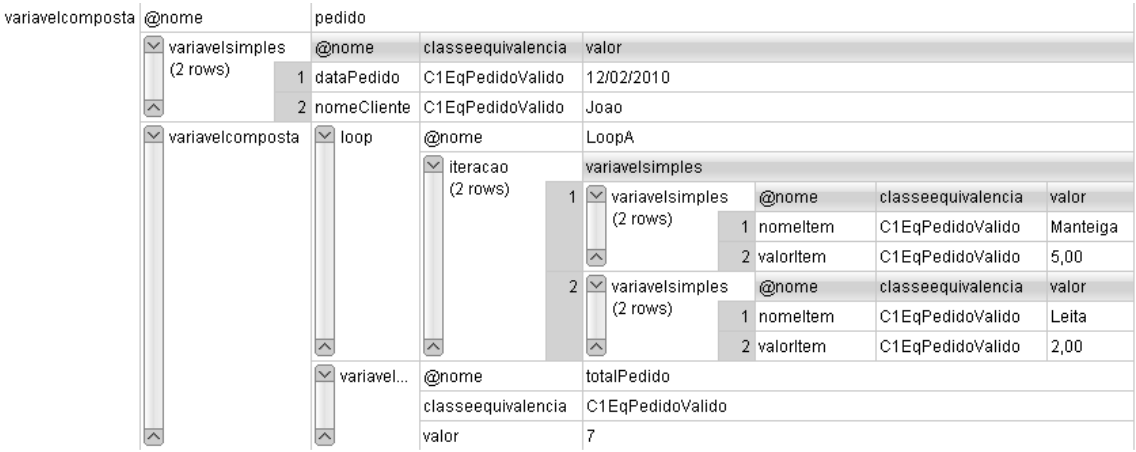

Figura 10. Visão em grade (XML) dos dados de um pedido

A Figura 11 exibe o Fluxo Principal e apenas um dentre os diversos fluxos alternativos possíveis. No Fluxo Principal, foi aplicado o filtro "C1EqPedidoValido", correspondente à classe de equivalência de pedidos válidos. No Fluxo Alternativo, foi aplicado o filtro "ClEqVlItemInvalido", correspondente à classe de equivalência de valores inválidos. Os dados da Figura 10 são respectivamente capturados pelas variáveis do Fluxo Principal, gerando o script exibido na Figura 11.

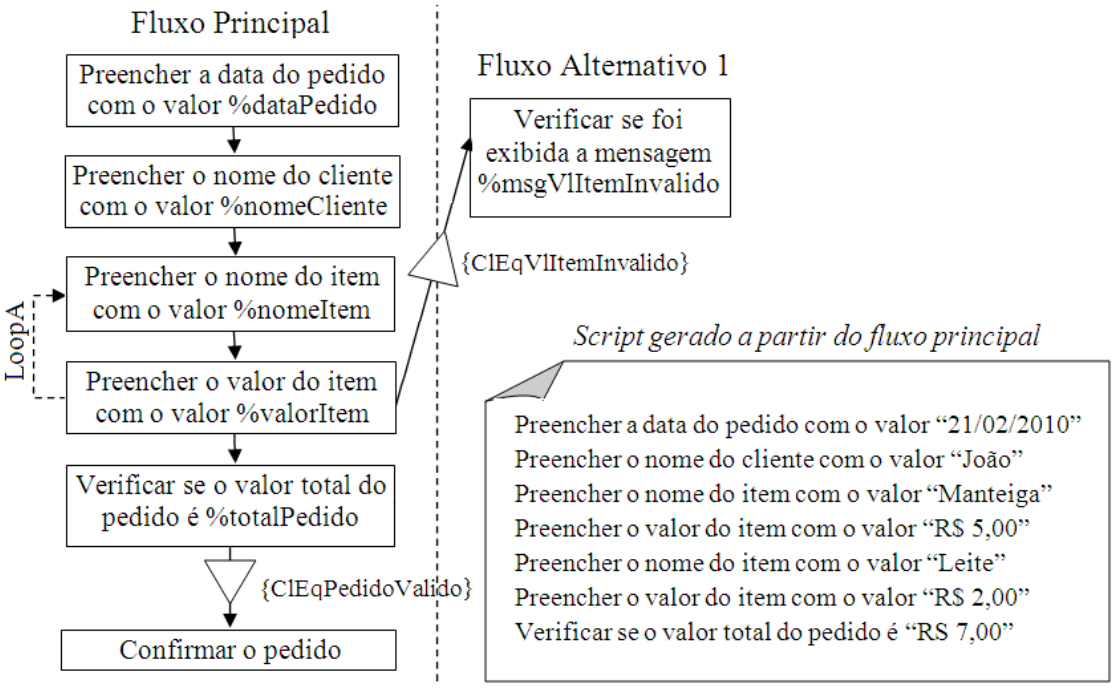

Figura 11. Exemplo de Uso do TSD em um sistema de cadastro.

3.4. Geração de Scripts a Partir do TSD

A geração dos scripts de testes a partir do TSD utiliza um algoritmo recursivo. O algoritmo deriva os casos de testes em três etapas (Figura 12):

- **Substituição dos Subdiagramas**: A primeira etapa consiste em substituir todos os subdiagramas no TSD pelos elementos do subdiagrama referenciado. Na Figura 12, o subdiagrama 3 é substituído pelos passos 3.1 e 3.2. Após esta etapa, os cenários derivados possuem apenas elementos dos tipos Passo, Seta do Fluxo, Filtro e Loop.
- **Derivação de Cenários**: A segunda etapa consiste em derivar todos os cenários necessários para cobrir todos os passos do diagrama. Cobrir todos os elementos do diagrama significa gerar casos de testes que, em seu conjunto, contenham pelo menos uma vez cada passo do diagrama. Na Figura 12 são obtidos dois cenários: Cenário 1 [Filtro 1, Loop 1 e Passos 2,3.1, 3.2 e 4] e Cenário 2 [Filtro 1 e Passos 2,5 e 6].
- **Instanciação dos Dados de Testes**: A terceira etapa consiste em aplicar todos os dados de testes conforme representado na Seção 3.2 obtendo os scripts de testes. Cada variável é substituída pelos seus respectivos valores, indicados na representação dos dados de testes. Caso haja um filtro, pertencente ao cenário, os valores cujas classes de equivalência de suas variáveis não pertencem à lista de classes do filtro serão descartados. Descartar os valores significa que não serão gerados casos de testes com os valores para o cenário que contém um filtro e as classes de equivalência dos valores não pertencem à lista de classes indicada no filtro. Na Figura12 a variável "v" é substituída pelos seus respectivos valores: A , B e C, representados na Listagem 3, gerando dois scripts de testes.

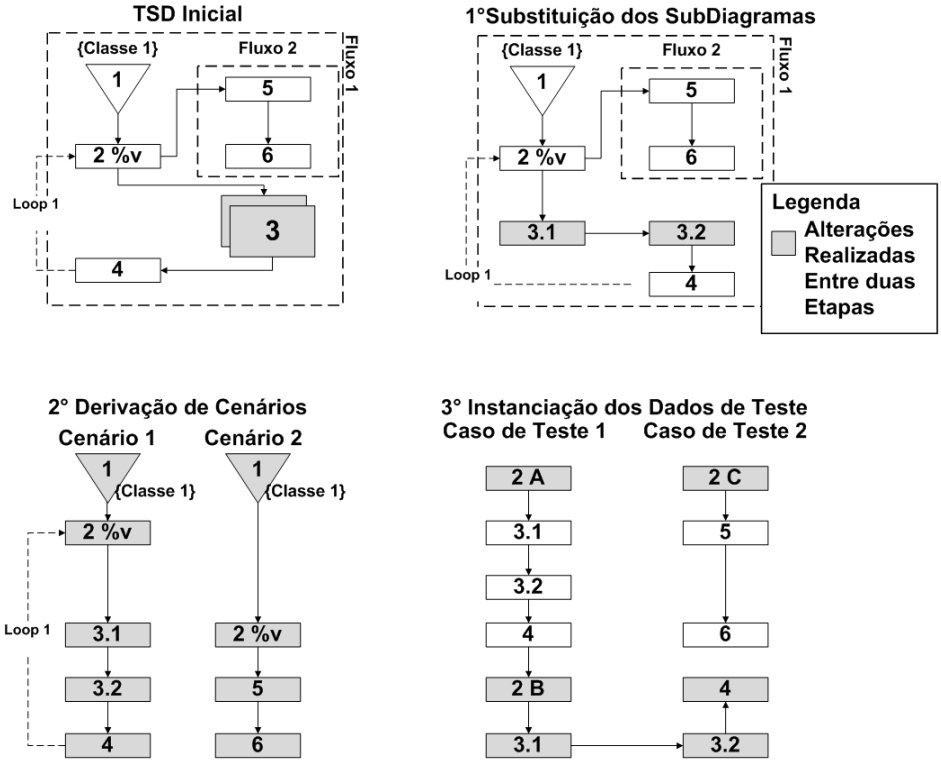

Figura 12. Etapas de derivação de scripts de testes a partir do TSD

Listagem 3. Representação de dados para scripts de testes que testam o TSD apresentado na Figura

```
<?xml version="1.0" encoding="UTF-8"?>
<dadoteste xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="file: TSD.xsd">
   <variavelcomposta nome="tela">
      <loop nome="Loop1">
         <iteracao>
            <variavelsimples nome="v">
               <classeequivalencia>Classe 1</classeequivalencia>
               <valor>A</valor>
            </variavelsimples>
         </iteracao>
         <iteracao>
            <variavelsimples nome="v">
               <classeequivalencia>Classe 1</classeequivalencia>
               <valor>B</valor>
            </variavelsimples>
         </iteracao>
      </loop>
      <variavelsimples nome="v">
         <classeequivalencia>Classe 1</classeequivalencia>
         <valor>C</valor>
      </variavelsimples>
   </variavelcomposta>
</dadoteste>
```

4. Experiência de Uso

O TSD foi utilizado em 66 projetos do Serviço Federal de Processamento de Dados (Serpro), empresa pública responsável pelo desenvolvimento dos sistemas do governo federal brasileiro. Atualmente há 24 projetistas de testes utilizando o TSD em quatro equipes de testes independentes, dispersas em quatro cidades. A ferramenta TestKase automatiza a geração dos scripts de testes modelados pelo TSD. A Figura 13 exibe uma tela de edição do TestKase.

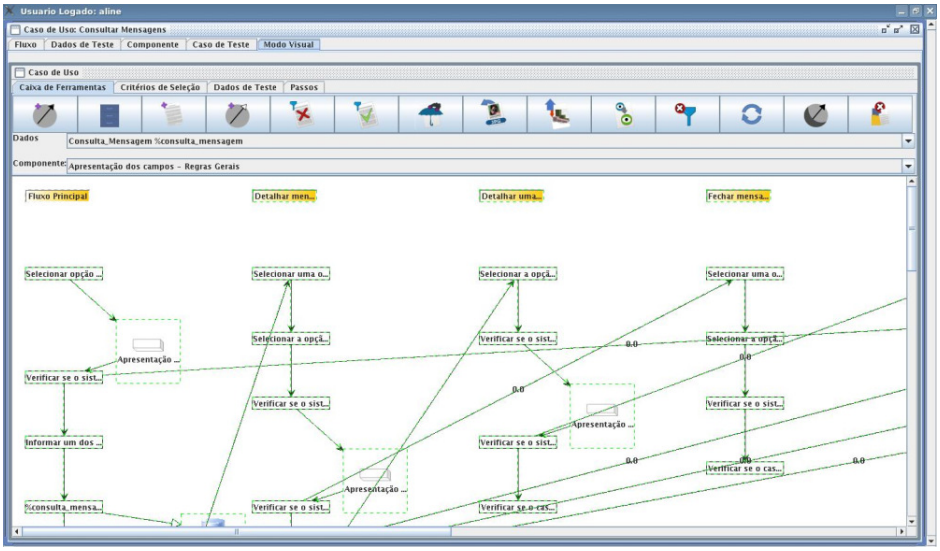

Figura 13. Tela de edição do TSD pela ferramenta TestKase

Foram realizadas uma avaliação quantitativa, com base num estudo de caso comparativo e uma avaliação qualitativa do uso da ferramenta.

O estudo de caso comparativo consistiu da implementação de uma build com e sem a utilização do TSD. Inicialmente a buid foi testada sem o TSD e em seguida conseguiu-se realizar novos testes aproveitando a disponibilidade da ferramenta. A equipe de testes foi composta por 6 projetistas e 1 testador. Foram observados quatro indicadores: o número de casos de teste por fluxo de caso de uso; a quantidade de casos de teste gerados; o esforço de implementação (medido em homens/hora) para o total de casos de teste; e o esforço de implementação por caso de teste. Todos os casos de testes projetados foram executados. A Tabela 2 sumariza os resultados do estudo comparativo.

Tabela 2 – Resultados do estudo de caso comparativo

Indicador	Sem o TSD	Com o TSD	Δ	Δ%
Casos de Teste/Fluxo	1,5	2,8	1,3	87
Esforço do Total dos Casos de Teste(Homens/hora)	60	36	-24	-40
Quantidade de Casos de Teste Gerados	45	80	35	78
Esforço (Homens/hora) / Caso de Teste	1,33	0,45	-0,88	-0,66

Observa-se que o Número de Casos de Teste por Fluxo de Caso de Uso saltou de 1,5 para 2,8, registrando acréscimo de 87%. Além disso, sem a utilização do TSD, os projetistas se limitaram a gerar 45 casos de teste, enquanto que, utilizando o TSD, geraram 80 casos de teste. Ambos os indicadores apontam uma maior cobertura das situações de erro na aplicação ao utilizar o TSD.

Apesar do aumento no número de casos de teste, o esforço para gerá-los experimentou uma redução considerável. Sem utilizar o TSD, o esforço foi de 60 homens/hora, enquanto que, utilizando o TSD, esse esforço decresceu para 36 homens/hora. Ou seja, o esforço necessário foi reduzido quase à metade.

Quando comparamos o esforço de implementação por caso de teste, o resultado é mais visível. O indicador decresceu de 1,33 homem/hora para 0,45 homem/hora por caso de teste, reduzindo-se o esforço ao utilizar o TSD para um terço do esforço sem sua utilização.

O número de erros encontrados durante o experimento, com ou sem o TSD, poderia ser um indicativo da qualidade do diagrama. Entretanto, este indicador seria fortemente influenciado pela qualidade do modelo. Um bom modelo identificaria um conjunto maior de erros, enquanto um modelo inapropriado identificaria um número inferior de erros. Vale ressaltar que a utilização de quaisquer modelos tende a gerar os casos de teste de forma mais sistemática que a sua não utilização.

Desta forma, uma limitação do estudo comparativo é que ele evidencia que com o TSD são gerados mais casos de testes em um menor tempo, entretanto não se pode concluir que a mera utilização do TSD implique na identificação de mais erros, pois isto dependeria da qualidade do modelo construído com o TSD.

A avaliação qualitativa foi realizada a partir de acompanhamento e de formulários com anotações da observação enviados a equipes de testes que utilizaram a ferramenta. Os projetistas que utilizaram o TSD afirmaram que a organização dos dados

de testes em classes de equivalência combinada com a utilização de filtros, que também deixam explícitas as classes de equivalência, possibilitou melhorias consideráveis na qualidade do conjunto de casos de testes obtidos. Um segundo ponto ressaltado foi que o fato de os dados estarem separados do modelo auxiliaria a sua reutilização. Além disso, com relação ao TestKase, foi considerada positiva a funcionalidade de exportação de dados para outras ferramentas.

5. Trabalhos Relacionados

Registram-se diversas iniciativas e sistemas para criação de casos e scripts de testes baseados em um ou mais diagramas UML, combinados entre si ou combinados com outros formalismos (principalmente com máquinas de estados):

- Abdurazik e Offutt [Abdurazik, 2000] propõem utilizar diagramas UML para geração de casos de testes. Seu trabalho utiliza diagramas de colaboração para geração dos casos de testes para testes de sistema.
- A Agedis Modelling Language (AML) baseia-se em UML, e inclui diagramas de classe, objeto, diagrama de estados e arquivos XML, em conjunto com uma linguagem de ações denominada IF. Os diagramas de classes descrevem o relacionamento entre classes. Os diagramas de estados descrevem o comportamento de cada classe e também são utilizados para diretivas de geração de teste [Hartman, 2003].
- A ferramenta Conformiq Qtronic utiliza modelagem na forma de máquina de estados. Os casos de testes são gerados em forma de diagramas de sequência com chamadas a funções [Conformiq, 2010].
- Briand e Labiche desenvolveram um método denominado Totem para testes funcionais de sistemas. O método deriva casos de testes a partir dos diagramas de atividade e iteração [Rocha, 2008].
- A ferramenta AGTCG gera um conjunto de casos de testes de forma aleatória a partir do diagrama de atividades da UML [Urbano, 2006].

O principal diferencial do diagrama TSD em relação às propostas apresentadas consiste em associar os dados de testes, organizados por classe de equivalência, aos passos oriundos de uma especificação de caso de uso.

O TSD possibilita o reuso de passos por meio da utilização de subdiagramas e possibilita o reuso de dados pela separação do modelo de testes da representação dos dados de testes. As ferramentas TaRGeT e TDE/UML apresentam propostas semelhantes às do TSD [Nogueria, 2007; Vanzin, 2006]. O diagrama TSD se diferencia no sentido de incluir uma representação em XML dos dados de testes e associar a representação ao diagrama, gerando scripts de testes que incluem os dados de testes.

6. Conclusão

O presente artigo apresentou o diagrama TSD, bem como os resultados obtidos com a sua utilização. Os sistemas utilizados no estudo de caso empregaram a ferramenta TestKase, desenvolvida para auxiliar a elaboração de scripts de testes a partir do diagrama TSD.

O TSD é um modelo gerado de forma semiautomática, a partir dos casos de uso. Após a geração de uma versão inicial do diagrama, o TestKase possibilita editá-lo

graficamente, de forma a criar Subdiagramas, agrupando um conjuntos de passos. Além disso, possibilita associar dados de testes a cada passo. Os dados de testes são organizados segundo classes de equivalência.

Inicialmente, o TSD foi desenvolvido academicamente, tendo seu projeto sido adotado pelo Serpro, que passou a desenvolvê-lo como um projeto interno, em parceria com a academia.

A experiência de utilização em mais de 66 projetos indica que, em média, quase dobrou o número de casos de testes utilizados em cada fluxo, o que denota um expressivo aumento na qualidade dos testes. Por outro lado, diminuiu em 40% o esforço despendido para realizar os testes de um mesmo conjunto de casos de uso.

Entre as oportunidades de melhoria da solução apresentada, merece destaque o uso do TSD para realização de estimativas de planejamento, execução e análise de testes baseadas no número de passos e de fluxos. A ferramenta TestKase já possui um módulo que calcula o esforço de testes baseado no TSD.

Recomenda-se a realização de estudos objetivando comparar o esforço real obtido com o esforço estimado, de forma a se comprovar a efetividade da estimativa.

7. Referências

Heumann, J. (2001) "Is a Use Case a Test Case?", Proc. of the International Conference on Practical Software Testing Techniques, Minnesota.

Apfelbaum, L., Doyle, J. (1997) "Model Based Testing". Software Quality Week Conference. Maio.

Gronau, Ilan , et al. (1998) "Architeture for Automated Software Testing". IBM Research Laboratory in Haifa. Technical Report .

Neto, A., Subramanyan, R., Vieira, M. , Travassos, G. e Shull, F. (2008) "Improving Evidence about Software Technologies: A Look at Model-Based Testing". IEEE Software, Volume 25, Paginás 10-13. Maio.

Benjamin, M. et al. (1999) "A Feasibility Study in Formal Coverage Driven Test Generation". DAC.

Dalal, S., Jain, A., Karunanithi, N., Leaton, J., Lott, C., Patton, G., Horowitz, B. (1999) "Model-Based Testing in Practice". ICSE'99: Proceedings of the 21st International Conference on Software Engineering. IEEE Computer Society Press.

Hessel, Andres, Petterson, Paul. (2007) "A Global Algorithm for Model- Based Test Suite". Proceedings of Third Workshop on Model-Based Testing, Electronic Notes in Theoretical Computer Science.

Neto, Pedro de A. dos Santos. (2006) "MODEST: Um Método de Teste Baseado em Modelos". Tese de Doutorado. Universidade Federal de Minas Gerais.

Fröhlich, P., Link, J. (2000) "Automated Test Case Generation from Dynamic Models". 14th European Conference on Object-Oriented Programming.

Rational Software Corporation. Rational Unified Process®, RUP. Disponível em http://www.wthreex.com/rup/. Acessado em fev/2010

Bittner, Kurt, Spence, Ian. (2002) "Use case modeling". Addison-Wesley.

Leffingwell, D. (2003) "Managing Software Requirements: A Use Case Approach". Addison-Wesley.

Welzer, T. (2002) "Knowledge-Based Software Engineering". Proceedings of the Fifth Joint Conference on Knowledge-Based Software Engineering. I O S Press.

Zielczynski, P. Traceability from Use Cases to Test Cases. Disponível em http://www.ibm.com/developerworks/rational/library/04/r-3217/index.html?S_TACT=105AGX15&S_CMP=EDU. Acessado em fev/2010

Kaner, C. (2003) What Is a Good Test Case?. STAR East.

Myers, Glenford J. (2004) "The Art of Software Testing". New York: John Wiley & Sons.

Nogueira, S, et al. (2007) "Model Based Test Generation: A Case Study". I Brazilian Workshop on Systematic and Automated Software Testing.

Abdurazik, A e Offutt, J. (2000) "Using UML Collaboration Diagrams for Static Checking and Test Generation". Third International Conference on the Unified Modeling Language.

Rocha, Camila, Martins, Eliane. (2008) "A Method for Model Based Test Harness Generation for Component Testing". Journal of the Brazilian Computer Society.

Urbano, Maria P. (2006) "Geração Automática de Testes a Partir de Modelos UML". Universidade do Porto.

Conformic Software. The Model-Based Testing of a Protocol Stack — a TTCN-3 Integrated Approach. Disponível em http://www.conformiq.com. Acessado em fev/2010

Hartman,A. e Nagin,K (2003) “Model Driven Testing - AGEDIS architecture, interfaces, and tools”, Proceedings of the 1st European Conference on Model Driven Software Engineering, Nuremburg.

Vanzin, D., Martins, I. L., and Pereira Filho, J. B. (2006). “TDE UML Editor - A Success Development Case of a Software Extension”. In Proceedings of the IEEE international Conference on Global Software Engineering (October 16 - 19, 2006). ICGSE. IEEE Computer Society, Washington, DC, 257-258.