Natural Language Requirement Specification for Web Service Testing

Harry M. Sneed & Chris Verhoef

for

WSE-2013

Eindhoven
Goals of a Web Service Test

- Analyse Service Requirements to obtain Test Cases
- Analyse Service Interface (WSDL)
- Generate Service Test Scripts
- Invoke Service Operations
- Generate Service Requests
- Validate Service Results
- Evaluate Service Quality

The overall Goal is to select the best possible service for the task at hand
Structure of a Service Requirement Specification (SRS)

- Business Goals
- Business Requirements
  - Functional
  - Non-Functional
  - Data
- Use Cases
- Business Interfaces
- Business Objects
- Business Data
- Business Rules
- Business Views

Requirements model should be complete, accurate, and consistent.
Use Cases fulfill Requirements
implement Business Rules
process Objects
invoke Operations
send Requests
receive Responses

Use Cases:
- References to Business Goals
- References to Requirements
- References to Business Rules
- References to Business Objects
- References to Service Interfaces
- References to other Interfaces
- Reference to Actors
- Trigger
- Pre-Conditions
- Post-Conditions
- Main Path with Steps
- Alternate Path with Steps
- Exceptions

Test Cases:
- Test Case 1
  - Test Operation A
    - Generate Request A
    - Validate Response A
  - Test Operation C
    - Generate Request C
    - Validate Response C
  - Test Operation B
    - Generate Request B
    - Validate Response B
- Test Case 2
  - Test Operation A
  - Generate Request A
  - Validate Response A
Role of Use Cases in Connecting Test Cases to the Service

Web Service

- Use Case
  - Test Case
  - Test Case
  - Test Case

- Use Case
  - Test Case
  - Test Case
  - Test Case

- Use Case
  - Test Case
  - Test Case
  - Test Case
From Requirements to Web Services via Use Cases

Requirements
Functional & Qualitative

Business Rules
Global & local

Business Objects

Constraints

Service Interface

Use Case Specification
Trigger / Pre- & Post Conditions / Paths / Steps / Functions

Test case = Sequence of Requests

Web Service

Operation
Input Msg
Algorithm
Output Msg

Operation
Input Msg
Algorithm
Output Msg

Operation
Input Msg
Algorithm
Output Msg

N Test Cases per Use Case

Request

Sub Service

Data base

Sub Service

Test Cases per Use Case

From Requirements to Web Services via Use Cases

Sneed-05
WSDL Definition is copied over into the Requirement Document.

Tester fills sample Test data into the Table.

Data Structure + Test data + Test conditions = Test cases.
## Service Interface Definition Table

<table>
<thead>
<tr>
<th>FUNC-04:</th>
<th>QueryAvailableProductsString</th>
<th>struc</th>
</tr>
</thead>
<tbody>
<tr>
<td>INPUT-04:</td>
<td>QueryAvailableProductsString3Request</td>
<td>struc</td>
</tr>
<tr>
<td>PARM:</td>
<td>CustNo</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>ArtType</td>
<td>string</td>
</tr>
<tr>
<td>OUTPUT-04:</td>
<td>QueryAvailableProductsString3Response</td>
<td>struc</td>
</tr>
<tr>
<td>RESULT:</td>
<td>return</td>
<td>array</td>
</tr>
<tr>
<td>RESULT:</td>
<td>item[1]</td>
<td>string</td>
</tr>
<tr>
<td>RESULT:</td>
<td>item[2]</td>
<td>string</td>
</tr>
<tr>
<td>FUNC-05:</td>
<td>BuySomething</td>
<td>struc</td>
</tr>
<tr>
<td>INPUT-05:</td>
<td>BuySomething4Request</td>
<td>struc</td>
</tr>
<tr>
<td>PARM:</td>
<td>AnOrder</td>
<td>struc</td>
</tr>
<tr>
<td>PARM:</td>
<td>CustNo</td>
<td>string</td>
</tr>
<tr>
<td>PARM:</td>
<td>OrderNo</td>
<td>int</td>
</tr>
<tr>
<td>PARM:</td>
<td>DeliveryMode</td>
<td>string</td>
</tr>
<tr>
<td>PARM:</td>
<td>PaymentMode</td>
<td>string</td>
</tr>
<tr>
<td>PARM:</td>
<td>Orders</td>
<td>array</td>
</tr>
<tr>
<td>PARM:</td>
<td>item[1]</td>
<td>struc</td>
</tr>
<tr>
<td>PARM:</td>
<td>OrderArtNo[1]</td>
<td>string</td>
</tr>
<tr>
<td>PARM:</td>
<td>OrderArtType[1]</td>
<td>string</td>
</tr>
<tr>
<td>PARM:</td>
<td>OrderArtName[1]</td>
<td>string</td>
</tr>
<tr>
<td>PARM:</td>
<td>OrderArtAmount[1]</td>
<td>int</td>
</tr>
<tr>
<td>PARM:</td>
<td>item[2]</td>
<td>struc</td>
</tr>
<tr>
<td>PARM:</td>
<td>OrderArtNo[2]</td>
<td>string</td>
</tr>
<tr>
<td>PARM:</td>
<td>OrderArtType[2]</td>
<td>string</td>
</tr>
<tr>
<td>PARM:</td>
<td>OrderArtName[2]</td>
<td>string</td>
</tr>
</tbody>
</table>
## Specification of a Use Case

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>Billing</td>
</tr>
<tr>
<td>fulfills</td>
<td>*Func-Req-06</td>
</tr>
<tr>
<td>Invokes</td>
<td>*FUNC-06.</td>
</tr>
<tr>
<td>Receives</td>
<td>*INPUT-06.</td>
</tr>
<tr>
<td>Sends</td>
<td>*OUTPUT-06, *REPORT-03.</td>
</tr>
<tr>
<td>Trigger</td>
<td>GUI-Menu.</td>
</tr>
<tr>
<td>Actor</td>
<td>Accountant</td>
</tr>
<tr>
<td>PreConditions</td>
<td>Billing item file must be available.</td>
</tr>
<tr>
<td>PostConditions</td>
<td>Bills are printed.</td>
</tr>
</tbody>
</table>
| Main Path | 1) Accountant starts Billing job.  
            2) Service sorts Bills by Customer.  
            3) Service collects billing items for each customer.  
            4) Service fetches the customer data.  
            5) Service sums up the prices.  
            6) Service computes the VAT.  
            7) Service prints out a bill for each customer. |
| Alternate Paths | None |
| Exceptions | Service finds no billing items.  
            Service finds no customer data for the billing items.  
            Service finds no VAT Table. |
| Response Time | 3000 MS |
| Inherits  | Standard-Printing Use Case |
| Uses      | Batch-Process |
| Extends   | Customer Order processing |
| Comment   | Bill recipient is the customer. |
### Functional Requirements

**FREQ-02 ArticleSelection**
The customer should have the possibility to view all the articles of a selected type with their names and prices.
The articles should be ordered alphabetically.

**FREQ-03 OrderProcessing**
The customer should have the possibility of selecting one or more articles to be ordered. He needs only to select an article and give in the amount. The service will confirm his identity and check his credibility. The credibility rate should be over 2. If they are both ok, the processing may continue, otherwise it should be terminated with a message to the user.
If an ordered article is available on stock and in sufficient quantity, the order is to be accepted. If the article is not on stock the order is to be rejected. If the quantity on stock is too low, a back order is to be created.

### Non-Functional Requirements

**QREQ-01 ResponseTime**
The response time for a customer query should be <= 2 seconds. The response time for a customer order should be <= 3 seconds. The response time for the background jobs should be <= 10 seconds.

**QREQ-03 Availability**
The system should be available 24 hours a day, 7 days a week, for at least 95% of the time.
Use Case Specification:
Label: processCustomerOrder
Requirements: FREQ-01, FREQ-02, FREQ-03.
Rules: BR-01, BR-02, BR-03, BR-04, BR-05, BR-06.
Functions: FUNC-01, FUNC-02, FUNC-03, FUNC-04.
Inputs: INPUT-01, INPUT-02, INPUT-03, INPUT-04.
Outputs: OUTPUT-01, OUTPUT-02, OUTPUT-03, OUTPUT-04.
Objects: BO-01, BO-02, BO-05, BO-06, BO-07, BO-09, BO-11, BO-13.
Trigger: Menu_Selection
Actor: Customer
Frequency: Continuous
ResponseTime: 3 Sec
PreConditions: Ordered article must be on stock. Customer must be authorized to purchase articles. Customer must be credible.
PostConditions: Article amount is reduced by the order and Dispatch order exits for each fulfilled order item and Billing item exits for each fulfilled order item or back order exits for not fulfilled order item. A supply order item exits if the article amount falls below the minimum order amount required.
MainPath:
1) GetArticleTypes Customer requests article types.
   Service returns a list of article types.
2) ArticleQuery Customer selects an article type.
   Service returns a list of articles and prices.
3) CustomerOrder Customer orders an article.
   Service checks customer credit.
   Service checks if article on stock.
   Service checks if amount on stock is sufficient.
   If not sufficient, service creates a back order.
   If sufficient, service subtracts amount ordered from amount on stock, creates a dispatch order item and creates a billing item.
   If amount on stock falls below the minimum amount, the service creates a resupply order.
Generating a Service Test Case Table

FUNC-02: QueryAvailableProducts.
INPUT-02: QueryAvailableProducts2Request.
The article query request should contain:
# CustNo = “009999”.
# ArtType = “BOOK”.
OUTPUT-02: QueryAvailableProducts2Response.
The article query response should contain:
# item[1].
# ResponseArtNo[1] = “004711”.
# ResponseArtType[1] = “BOOK”.
# ResponseArtName[1] = “MeinKampf”.
# ResponseArtPrice[1] = “40.50”.

<table>
<thead>
<tr>
<th>TestCase</th>
<th>Operation</th>
<th>Parameter</th>
<th>Type</th>
<th>Ind</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Orders02</td>
<td>GetTypes</td>
<td>GetTypes1Request</td>
<td>Parameter</td>
<td></td>
<td>?</td>
</tr>
<tr>
<td></td>
<td>GetTypes</td>
<td>item</td>
<td>Return</td>
<td>1</td>
<td>MAGA</td>
</tr>
<tr>
<td></td>
<td>GetTypes</td>
<td>item</td>
<td>Return</td>
<td>2</td>
<td>NEWS</td>
</tr>
<tr>
<td></td>
<td>GetTypes</td>
<td>item</td>
<td>Return</td>
<td>3</td>
<td>BOOK</td>
</tr>
<tr>
<td>QueryAvail</td>
<td>CustNo</td>
<td></td>
<td>Parameter</td>
<td></td>
<td>0099999</td>
</tr>
<tr>
<td>QueryAvail</td>
<td>ArtType</td>
<td></td>
<td>Parameter</td>
<td></td>
<td>BOOK</td>
</tr>
<tr>
<td>QueryAvail</td>
<td>Item</td>
<td></td>
<td>Return</td>
<td>1</td>
<td>Struc</td>
</tr>
<tr>
<td>QueryAvail</td>
<td>ResponseArtNo</td>
<td></td>
<td>Return</td>
<td>1</td>
<td>04711</td>
</tr>
<tr>
<td>QueryAvail</td>
<td>ResponseArtType</td>
<td></td>
<td>Return</td>
<td>1</td>
<td>BOOK</td>
</tr>
<tr>
<td>QueryAvail</td>
<td>ResponseArtName</td>
<td></td>
<td>Return</td>
<td>1</td>
<td>MeinKampf</td>
</tr>
<tr>
<td>QueryAvail</td>
<td>ResponseArtPrice</td>
<td></td>
<td>Return</td>
<td>1</td>
<td>40.50</td>
</tr>
</tbody>
</table>
Generating a Web Service Test Script

// First Request to Frontend to order Articles
if ( operation = "GetTypes");
if ( request = "GetTypes1Request");
   assert inp.GetTypes1Request_DummyParam = "?";
endRequest ;
if ( response = "GetTypes1Response");
   assert out.$ResponseTime < "1000";
if ( object = "return" occurs = "2");
   assert out.item[1] = old.item[1];
endObject;
endResponse ;
endOperation;
if ( operation = "QueryAvailableProducts");
if ( request = "QueryAvailableProducts2Request");
   assert inp.CustNo = "009999";
   assert inp.ArtType = "BOOK";
endRequest ;
if ( response = "QueryAvailableProducts2Response");
   assert out.$ResponseTime < "1000";
if ( object = "return" occurs = "1");
   if ( object = "item[1]");
      assert out.ResponseArtNo = "4711";
      assert out.ResponseArtType = "BOOK";
      assert out.ResponseArtName = "MeinKampf";
   endObject;
<table>
<thead>
<tr>
<th>WSDL Response Validation Report</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tester: IWarehouseWebService</td>
</tr>
<tr>
<td>TestName: WSDL-Response</td>
</tr>
<tr>
<td>Date: 04.09.12</td>
</tr>
<tr>
<td>TestCase: 001</td>
</tr>
<tr>
<td>System: Orders</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Non-Matching Params</th>
<th>Non-Matching Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resp-Id: GetTypes1Response_001_return</td>
<td></td>
</tr>
<tr>
<td>Ist: item[2]</td>
<td>BOOK</td>
</tr>
<tr>
<td>Soll: Asserted_Value</td>
<td>=NEWS</td>
</tr>
</tbody>
</table>

| Ist: item[3]        | 0                  |
| Soll: Asserted_Value | =BOOK              |

| Ist: item[1]        | ________________    |
| Soll: Asserted_Value | =MAGA              |

| Ist: item[2]        | BOOK               |
| Soll: Asserted_Value | =NEWS              |

| Resp-Id: GetTypes1Response_001       | missing from the old Response |

Total Number of old Responses: 08
Number of old Responses matched to new ones: 06
Total Number of Params checked: 18
Total Number of non-Matching Params: 16
Percentage of matching Params: 11 %
Percentage of matching Responses: 80 %
Steps of the Web Service Evaluation Process

1) Prepare the SRS
2) Analyse the SRS
3) Audit the WSDL
4) Edit the Test Cases and generate the Test Scripts
5) Compile the Test Scripts
6) Generate the Service Requests
7) Start the service Test
8) Validate the Service Results
9) Evaluate the Service Quality

Figure 7: Web Service Testing Process
Summary

- A structured semi-formal Requirement Specification can be used as a basis for web service testing.
- The requirement document must follow a predefined structure and be marked with key words.
- It is more costly to create but the costs are saved later.
- The logical test cases can be automatically generated from the natural language text.
- The test cases can lead to a compileable test script.
- The biggest remaining effort is for selecting meaningful test data values which fit the purpose of the test cases.