The file contains segments from our soapUI training class, including:

- Best practices for service testing lecture
- Request/response lecture and exercise
- TestStep (PropertyTransfer) lecture and exercise
- TestStep quiz
- Assertion (SOAP Fault) lecture and exercise
- Case study
- Refactoring exercise
- Load testing (loadUI) lecture
- Reporting (TestSuite) exercise

We offer this class in both public and private configurations. You’ll find the full syllabus here:


To register for a live, instructor-led public Webinar, visit:

http://www.wiseclouds.com/soapui_register.html

If you’d like more information, visit:

http://www.wiseclouds.com/contact.html
Seven Fundamentals of Mission-Critical Service Testing

1. Thoroughly Test Your Services
2. Test Using Large Amounts of Realistic Data
3. Make Sure Your Services Are Secure
4. Get the Most Productivity from Your Developers and Testers
5. Fully Track Your Test Results
6. Test Your Services Under Anticipated Loads
7. Make Sure You Govern Your Services

Important: while we illustrate examples with soapUI Pro, the principles themselves are vendor-neutral
Make Sure You Govern Your Services - Best Practices

- Ensure that services conform to standards
- Ensure that services conform to their schema
- Refactor your tests when contracts change
- Invest in governance technology, even if rudimentary
# Request & Response Details - Outline View

<table>
<thead>
<tr>
<th>Tab</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aut</td>
<td>Supply security details in the request</td>
</tr>
<tr>
<td>Headers</td>
<td>Set custom HTTP headers in the request; view returned HTTP headers in the response</td>
</tr>
<tr>
<td>Attachments</td>
<td>Attach files to the request; view attachments in the response</td>
</tr>
<tr>
<td>SSL Info</td>
<td>Display certificate details (if any) in the response</td>
</tr>
<tr>
<td>WS-A</td>
<td>Set WS-Addressing headers for request</td>
</tr>
<tr>
<td>WS-RM</td>
<td>Set WS-ReliableMessaging headers for request</td>
</tr>
<tr>
<td>WSS</td>
<td>Show WS-Security details (if any) in the response</td>
</tr>
<tr>
<td>JMS Headers</td>
<td>Display JMS headers in the request</td>
</tr>
<tr>
<td>JMD Property</td>
<td>Display JMS properties in the request</td>
</tr>
<tr>
<td>JMS</td>
<td>Show the response’s JMS headers and properties</td>
</tr>
<tr>
<td>XSD</td>
<td>Show the XML schema for the selected node</td>
</tr>
<tr>
<td>Table</td>
<td>Present the request and response elements in read-only tabular form</td>
</tr>
<tr>
<td>XML</td>
<td>Show the XML representation of the request and response, in read only form</td>
</tr>
<tr>
<td>Doc</td>
<td>Display documentation (if any) that’s embedded in the schema</td>
</tr>
<tr>
<td>Coverage</td>
<td>Provide a color representation of what’s been covered in the request and response messages – red means uncovered, green means covered</td>
</tr>
</tbody>
</table>
Exercise 11: Transfer Properties from a Response

- **Purpose:** Learn an alternate means of transferring properties
- **Level:** Intermediate

1. Create a new project using the following WSDL:
   
   http://SERVER/EchoSOAPBinding?WSDL

   Check the **Create Requests** and **Create TestSuite** boxes. Accept all the defaults.

2. Expand the **TestSuite** -> **Forward TestCase** entry.

3. Create a new Properties TestStep. Place it below the **Forward** TestStep.

4. Add a property called **myProperty** to the new Properties TestStep.

5. Double-click on the **Forward** TestStep, outline view.

6. Fill in the value for **Forward** Request, and run the Request.

7. Switch to the **Response** tab.

8. Click on the **Transfer To** drop-down.

9. Select the **Properties** step and property value that you added in step 5.

10. When the Transfer to Property dialog box appears, just click **OK** to accept the defaults.

11. Notice that there is now a PropertyTransfer TestStep between the TestRequest and Properties TestSteps.

12. Double-click on the **TestCase**, and click the **Run** icon.

13. Double-click on the Properties TestStep. You should see that the response from the Forward service call has been transferred there.
Select the most appropriate TestStep for each situation

1. To pause your Test Case for a specific period of time, you would use the _________ TestStep.
2. You want to launch a different Test Case in another project. You would select the _________ TestStep.
3. To send a SOAP message to a Web service, you would use the _________ TestStep.
4. To keep track of attributes for your Test Case, you can use the _________ TestStep.
5. You want to move properties from one location to another. The TestStep most appropriate for this is _________.
6. You need to branch your Test Case based on a certain criteria. The _________ TestStep is designed for this purpose.
7. To send a message to a service that has a WADL description, you would choose the _________ TestStep.
8. To communicate with a non-SOAP service that has no WADL description, the _________ TestStep is the most appropriate.
9. To send a message to a database, you would choose the _________ TestStep.
10. You want to feed your test with random data. The _________ TestStep is designed for this purpose.
11. You want to write test results to a file or database. You would choose the _________ TestStep.
12. To feed your test with stored information, you would select the _________ TestStep.
13. To fetch the next set of information from a DataSource, you need to use the _________ TestStep.
14. You want to generate a simulated service and specify a reply for that service. The _________ TestStep is the best choice.
15. You want to write your own logic and have it be included in a Test Case. You would select the _________ TestStep.

A. Properties
B. DataSink
C. Run Test Case
D. HTTP TestRequest
E. Groovy Script
F. TestRequest
G. Property Transfer
H. JDBC TestRequest
I. Conditional Goto
J. Delay
K. REST TestRequest
L. DataGen
M. DataSource
N. DataSource Loop
O. MockResponse
SOAP Fault

- A standardized way to return error messages from a Web service
  - You’re still free to create your own types of error messages

- Has the following structure (this varies by SOAP version)

<table>
<thead>
<tr>
<th>Sub Element</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;faultcode&gt;</td>
<td>An application-provided code that identifies the error</td>
</tr>
<tr>
<td>&lt;faultstring&gt;</td>
<td>A plain text description of the error</td>
</tr>
<tr>
<td>&lt;faultactor&gt;</td>
<td>The URI of the SOAP node that generated the fault</td>
</tr>
<tr>
<td>&lt;detail&gt;</td>
<td>Application-specific error information</td>
</tr>
</tbody>
</table>
Exercise 6: Create a SOAP Fault Assertion

- **Purpose:** Check that a Response returns a SOAP fault
- **Level:** Beginner

1. Using the current project, remove all assertions by highlighting each assertion and clicking the red X found in the assertions area.

2. Add a new assertion, choosing the **SOAP Fault** value from the **Compliance, Status, and Standards** group.

3. Run your test and monitor the results. Unless you provide a **ShipmentID** of **123456789** (which triggers a SOAP Fault), this assertion will normally fail (in other words, there is no SOAP Fault error message returned).

You can use the SOAP Fault assertion to ensure that your service correctly returns a fault when it receives bad data or encounters another problem.
Case study: Design a comprehensive testing plan

In this scenario, you've just joined the staff of Pay-n-Pray Motors, a deep-discount car rental agency. The company is expanding operations to serve some of the largest US airports, including:

• New York (JFK)
• Boston (BOS)
• Chicago (ORD)
• Miami (MIA)
• Los Angeles (LAX)
• San Francisco (SFO)

Pay-n-Pray categorizes its battered, dented fleet of vehicles in the following classes:

• Subcompact
• Compact
• Intermediate
• Full-size
• SUV
• Van
• Luxury

To interact with major online reservations systems, Pay-n-Pray will be creating a series of Web services. The first Web service contains only one operation (CheckCarInventory) that reports on the immediate availability of a particular vehicle at a particular airport for a given number of days. It will accept the following inputs:

• Airport (must be in the airport list above)
• VehicleClass (must be in the vehicle class above)
• NumberOfDays (must be greater than equal to 1, and less than or equal to 30)

The service will return two results elements:

• NumberOfVehicles (either the number of available vehicles (0 means none available), or -1 to indicate an error)
• Message (either 'Success: vehicles are available' or a SOAP fault error message)

Now that we've described the background, let's look at your tasks. For each one, take a few minutes to write down (at a high level) how you would go about performing it.

1) Your first responsibility is to connect to the service and just make sure that it returns some data. What's the quickest and easiest way to do this?

2) Next, it's time to get more formal with your testing and create the proper container(s). What should you create to hold your TestSteps?
   a. What's the right sequence of these containers?
   b. What kind of TestStep should you use to communicate with this service? What would you do if it was a RESTful service instead of a SOAP service?
3) Now that you’ve got a working test, it’s time to experiment and send it some different data and see what happens. For now, you’ll send the service hard-coded data, one call at a time as follows:
   a. MIA, Subcompact, 2
   b. BOS, Luxury, 33
   c. SFO, Van, 0
   d. ORD, Intermediate, 10
   e. LHR, Full-size, 5
   f. LAX, Ferrari, 7

Looking at these examples, can you identify which ones should trigger an error from the service? What would you do to verify that an error has indeed been created?

For the valid entries, how can you test that the service is returning the proper results?

How would you structure your tests to ascertain results for both good and bad data?

4) With your tests in place, it’s now time to test for other important attributes, including:
   a. Results returned within a certain number of milliseconds
   b. Results are in a well-formed message
   c. Results match their schema

How would you modify your tests to reflect these new requirements?

5) You arrive at work one morning to learn that management wants to test 150,000 combinations of airport, vehicle, and number of days. After catching your breath, you realize that there’s a fairly straightforward way of doing this. What would you do next? How would you go about it?

6) Once your new plan is in place, management informs you that it’s now necessary to keep a permanent record of the results returned from the service. What would you do next? How would you go about it?
Exercise 2: Refactor Messages

- **Purpose:** Handle changes introduced by new, modified, deleted messages or schema
- **Level:** Intermediate

1. Create a project using the `OrderRatingService.wsdl` file provided by your instructor. Make sure the **Create Requests** and **Create TestSuite** boxes are checked, and accept all defaults.

2. Once the project has been created, right-click on the Interface, and choose the **Refactor Definition** menu option.

3. Browse to the `v2NewOrderRatingService.wsdl` file provided by your instructor. One of the operations in this file has been renamed, and there are schema differences, too. Accept the defaults on this dialog.

4. Connect the old and new operations.

5. Click **Next** to move on to the **Refactor** schema dialog.

6. Check the filter box to identify only updated schema elements.

7. Where applicable, set new values for unmatched schema elements. These new values will serve as defaults for all requests and TestRequests, but can be overridden.

8. Where applicable, discard unused (i.e. present in the older WSDL, but no longer present in the new WSDL) schema elements.

9. Click **Next** to continue.

10. Since there are no XPath expressions to update, just click **Finish** on the following dialog.

11. Explore all aspects of the project, including the Interface and TestCase. Look for examples of renaming and schema updates.
Generators
Exercise 3: Run a TestSuite Report

- Purpose: View a Jasper report with TestSuite results
- Level: Beginner

1. Double-click on the **Complaint Individual Assertions TestSuite** entry.
2. Run the TestSuite.
3. When it finishes, click on the **Create a Report for this item** icon (at the top right of the TestCase window).
4. Select **TestSuite** from the **Format** drop-down.
5. Accept all defaults.
6. View the report once it’s created.
7. Bonus: run the TestSuite several times; re-create and view the report when errors occur.