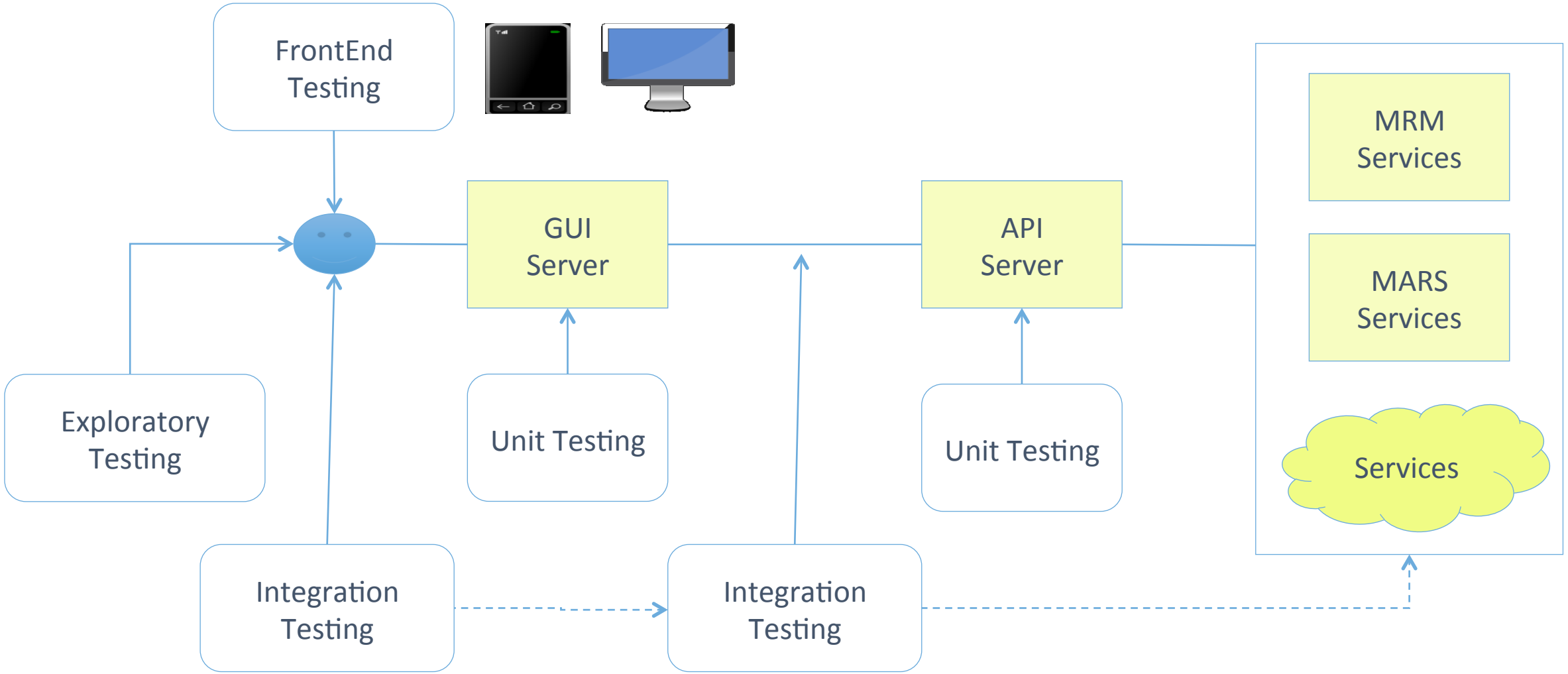
Planit
SOFTWARE TESTING

Web Services/ API Testing

Understand the utility and necessity of testing Web Services interfaces.

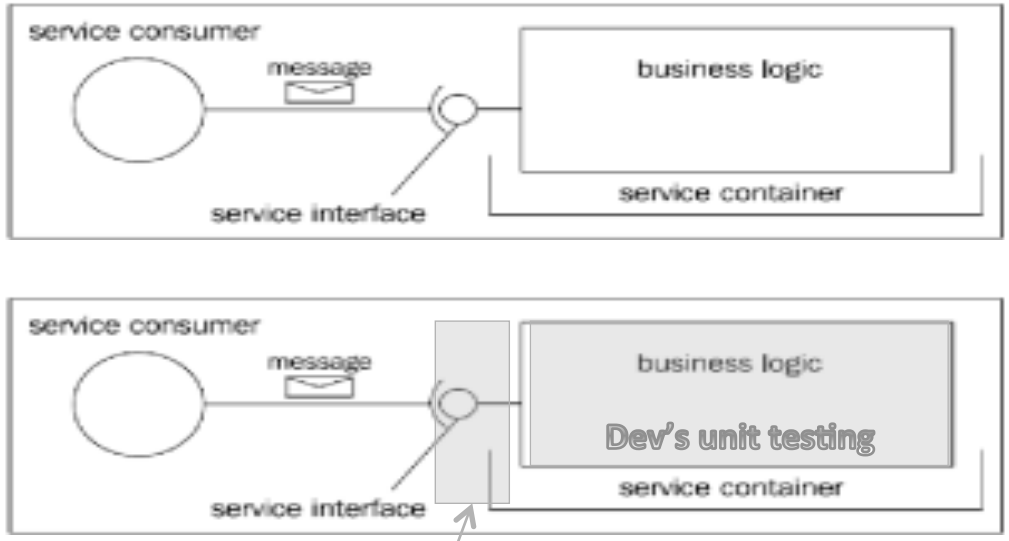
www.planittesting.com

Types of testing performed



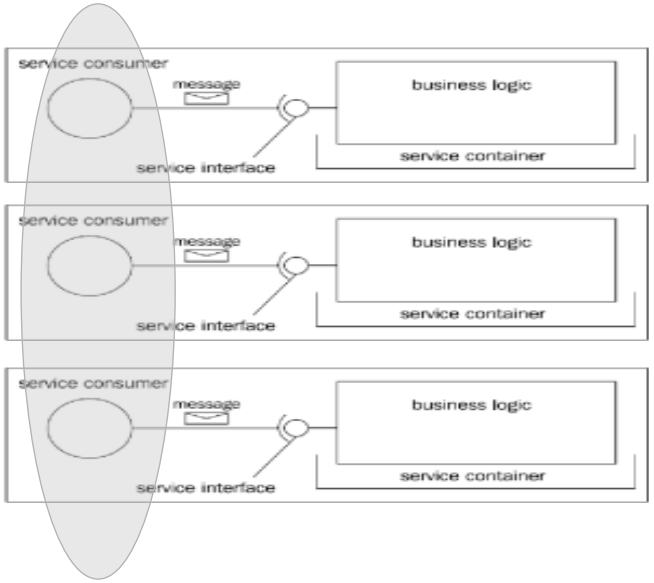
Intro to testing WS interfaces

UNIT testing



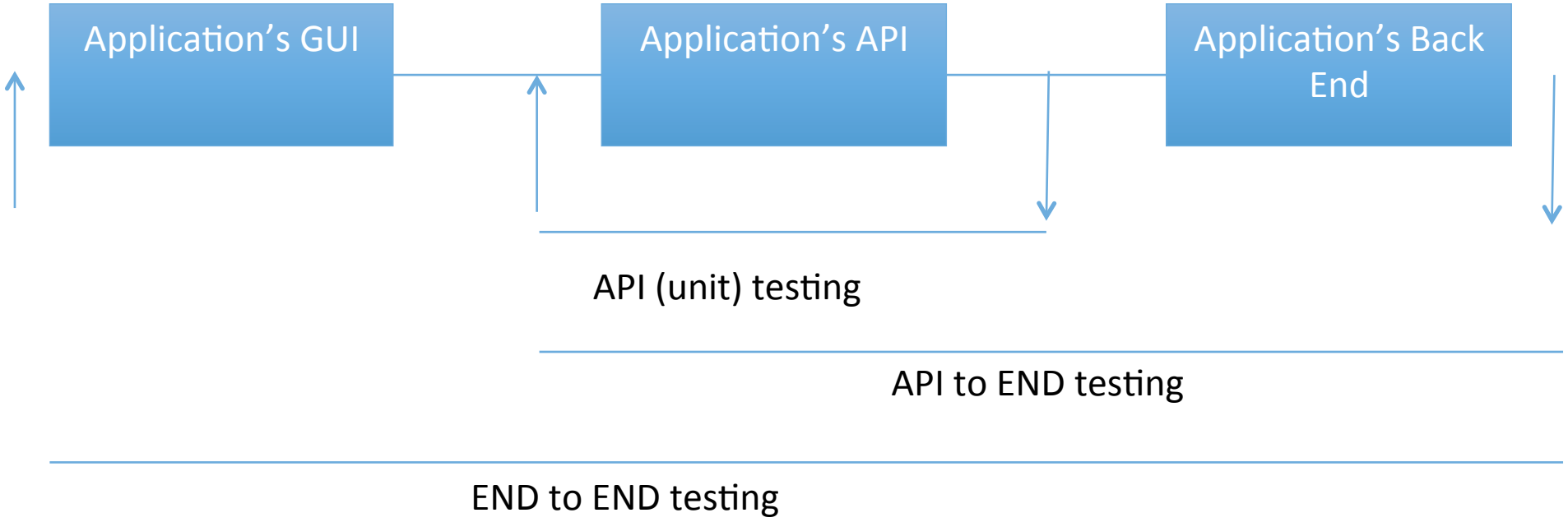
Who tests this?

END-to-END testing



The same service consumer uses different services in a controlled flow

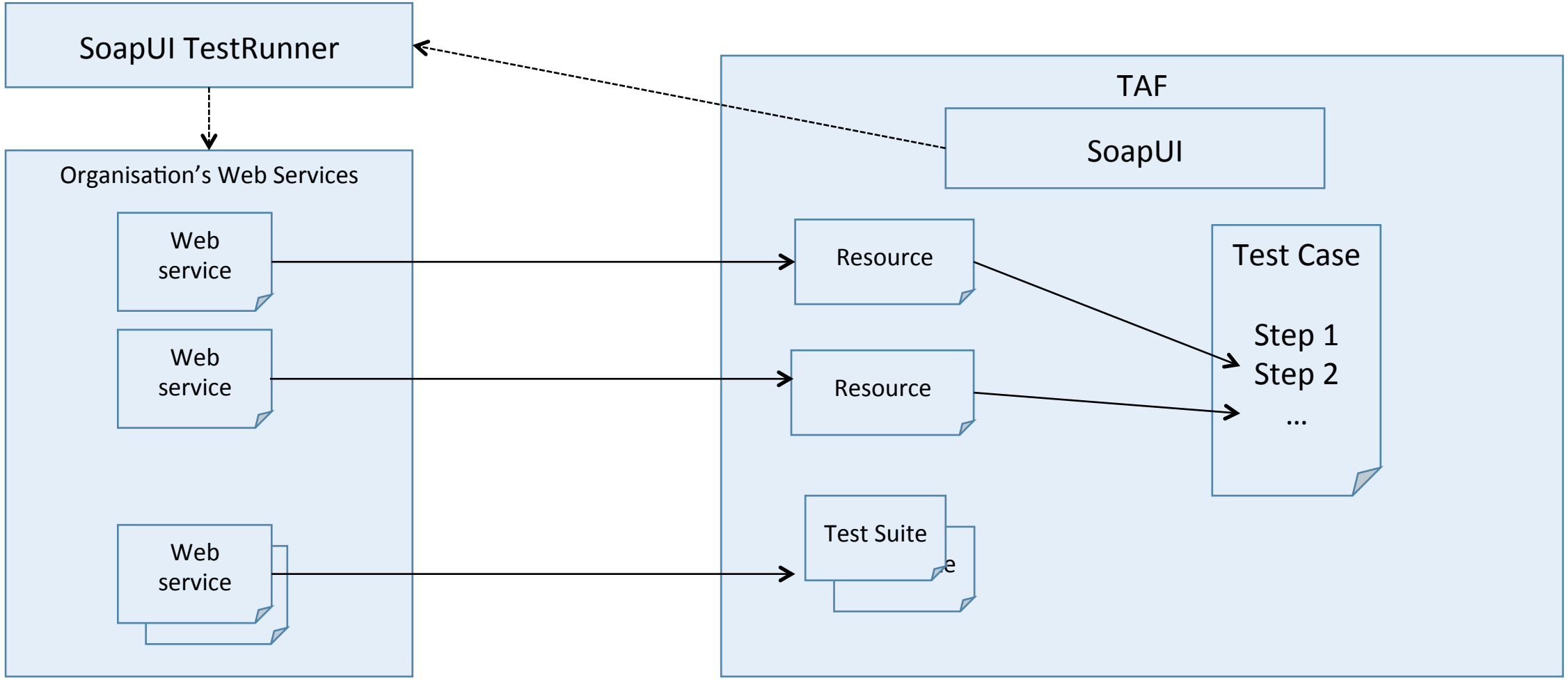
Some reasons to perform API testing



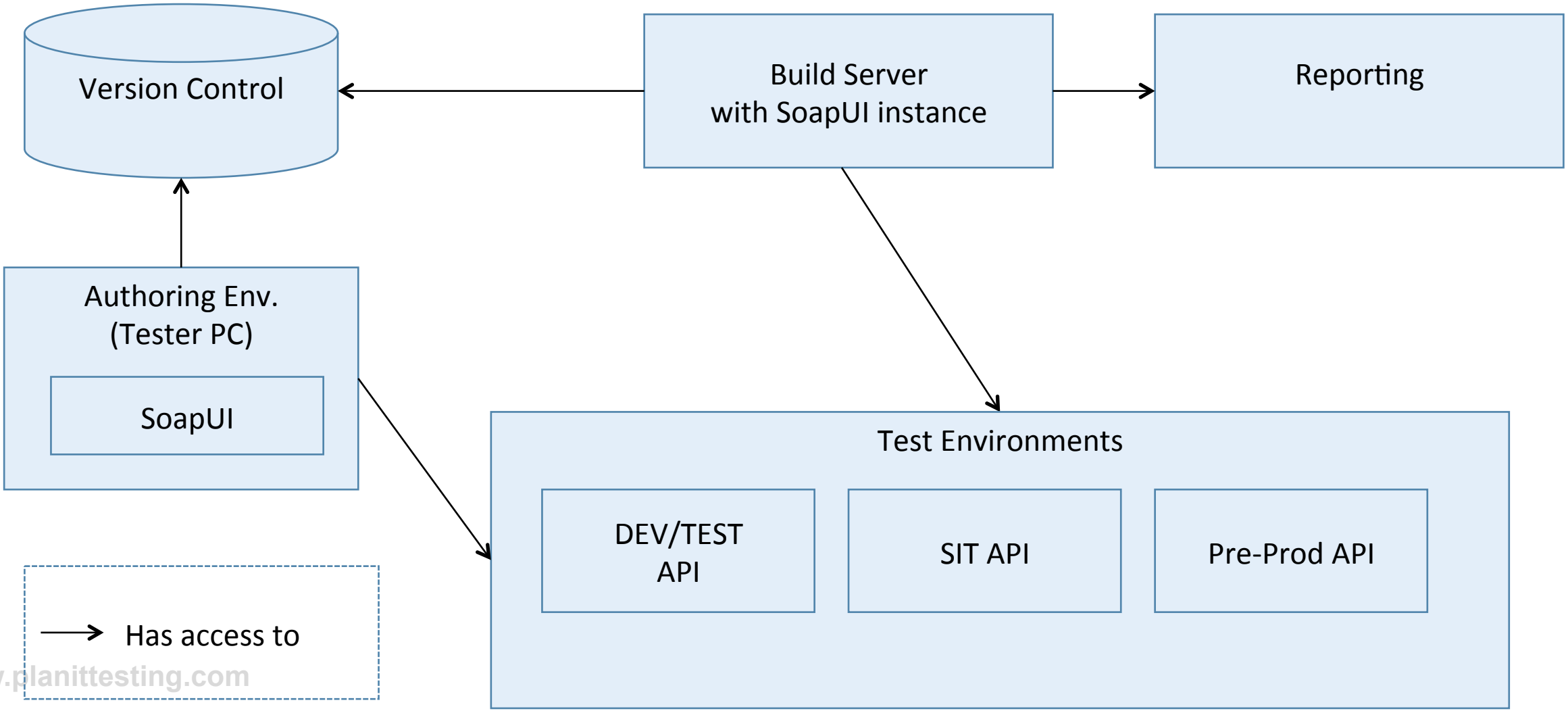
Web Service/API testing

1. Understand the ~~WSDL file~~ contract of the service
2. Determine the operations that particular web service provides
3. Determine the XML or JSON request format which we need to send
4. Determine the response XML or JSON format
5. Using a tool or writing code to send request and validate the response

Anatomy of automated API testing



Architecture for API Testing



All you need is REST

Soap vs Rest

- SOAP
  - Posts a message to a URL
  - Uses XML messages
  - Actions defined by a name in the WSDL
  - Only one URL for the service

$\supset$

- REST
  - Can GET resources, POST messages, PUT resources, DELETE resources
  - Can use XML, Json messages
  - Actions defined by the URL and the protocol – several URLs

SOAP request to SOAP service

SoapUI 5.0.0

Projects
- REST Project 1
  - http://www.webservicex.com
    - Globalweather.asmx [/glob
      - Globalweather.asmx
        - Request 1
- Sample1
  - GlobalWeatherSoap
    - GetCitiesByCountry
      - Request 1
    - GetWeather
  - GlobalWeatherSoap12
- Test

Request Properties

Property	Value
Name	Request 1
Description	
Message Size	331

Request 1

Method	Endpoint	Resource	Parameters
POST	http://www.webservicex.com	/globalweather.asmx	

Request 1

http://www.webservicex.com/globalweather.asmx

```
<soapenv:Envelope xmlns:soapenv="http
   <soapenv:Header/>
   <soapenv:Body>
      <web:GetCitiesByCountry>
         <!--Optional:-->
         <web:CountryName>peru</web:C
      </web:GetCitiesByCountry>
   </soapenv:Body>
</soapenv:Envelope>
```

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsi="htt
   <soap:Body>
      <GetCitiesByCountryResponse xmlns="http://www.webserviceX.NET">
         <GetCitiesByCountryResult><![CDATA[<NewDataSet>
 <Table>
   <Country>Peru</Country>
   <City>Atalaya</City>
 </Table>
 <Table>
   <Country>Peru</Country>
   <City>Pucallpa</City>
 </Table>
 <Table>
   <Country>Peru</Country>
   <City>Chimbote</City>
 </Table>
 <Table>
   <Country>Peru</Country>
   <City>Tingo Maria</City>
 </Table>
 <Table>
   <Country>Peru</Country>
   <City>Chiclayo</City>
 </Table>
 <Table>
   <Country>Peru</Country>
   <City>Ayacucho</City>
 </Table>
 <Table>
   <Country>Peru</Country>
   <City>Andahuayla</City>
 </Table>
 <Table>
   <Country>Peru</Country>
   <City>Anta Huaraz</City>
 </Table>
 <Table>
   <Country>Peru</Country>
```

... Hea... Attach... ... W... JMS ... JMS Pr...

Headers (10) Attachments (0) SSL Info WSS (0) JMS (0)

REST request to SOAP service

SoapUI 5.0.0

Projects
- REST Project 1
  - http://www.webservicex.com
    - Globalweather.asmx [/glob
      - Globalweather.asmx
        - Request 1
- Sample1
  - GlobalWeatherSoap
    - GetCitiesByCountry
      - Request 1
    - GetWeather
  - GlobalWeatherSoap12
- Test

Request 1

Method: POST
Endpoint: http://www.webservicex.com
Resource: /globalweather.asmx
Parameters:

Name | Value | Style | Level

Required:
Type:

Media Type: text/xml

```
<soapenv:Envelope xmlns:soapenv="
   <soapenv:Header/>
   <soapenv:Body>
      <web:GetCitiesByCountry>
         <!--Optional:-->
         <web:CountryName>peru</w
      </web:GetCitiesByCountry>
   </soapenv:Body>
</soapenv:Envelope>
```

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsi="http://www.w3.o
  <soap:Body>
    <GetCitiesByCountryResponse xmlns="http://www.webserviceX.NET">
      <GetCitiesByCountryResult><![CDATA[<NewDataSet>
 <Table>
   <Country>Peru</Country>
   <City>Atalaya</City>
 </Table>
 <Table>
   <Country>Peru</Country>
   <City>Pucallpa</City>
 </Table>
 <Table>
   <Country>Peru</Country>
   <City>Chimbote</City>
 </Table>
 <Table>
   <Country>Peru</Country>
   <City>Tingo Maria</City>
 </Table>
 <Table>
   <Country>Peru</Country>
   <City>Chiclayo</City>
 </Table>
 <Table>
   <Country>Peru</Country>
   <City>Ayacucho</City>
 </Table>
 <Table>
   <Country>Peru</Country>
   <City>Andahuayla</City>
 </Table>
 <Table>
   <Country>Peru</Country>
   <City>Anta Huaraz</City>
 </Table>
 <Table>
   <Country>Peru</Country>
   <City>Lima-Callao / Aerop. Internacional Jorgechavez</City>
```

Headers (10) | Attachments (0) | SSL Info | Representations (2) | Schema (conflicts) | JMS (0)

Intro to (Rest projects in) SoapUI

REST Projects

- Service – which is basically the server's URL (it can include a base path) e.g. http://www.webservicex.com

- Resource – represented by the path needed to name the listener (code/page/…) / globalweather.asmx
  - A resource has parameters – query, template, plain, header

- Methods – GET, POST, PUT, DELETE
  - A method has parameters – query, plain, header

- Request – The actual message that is sent
  - It inherits parameters from its Method and its Resource
  - It contains some JSON or XML message

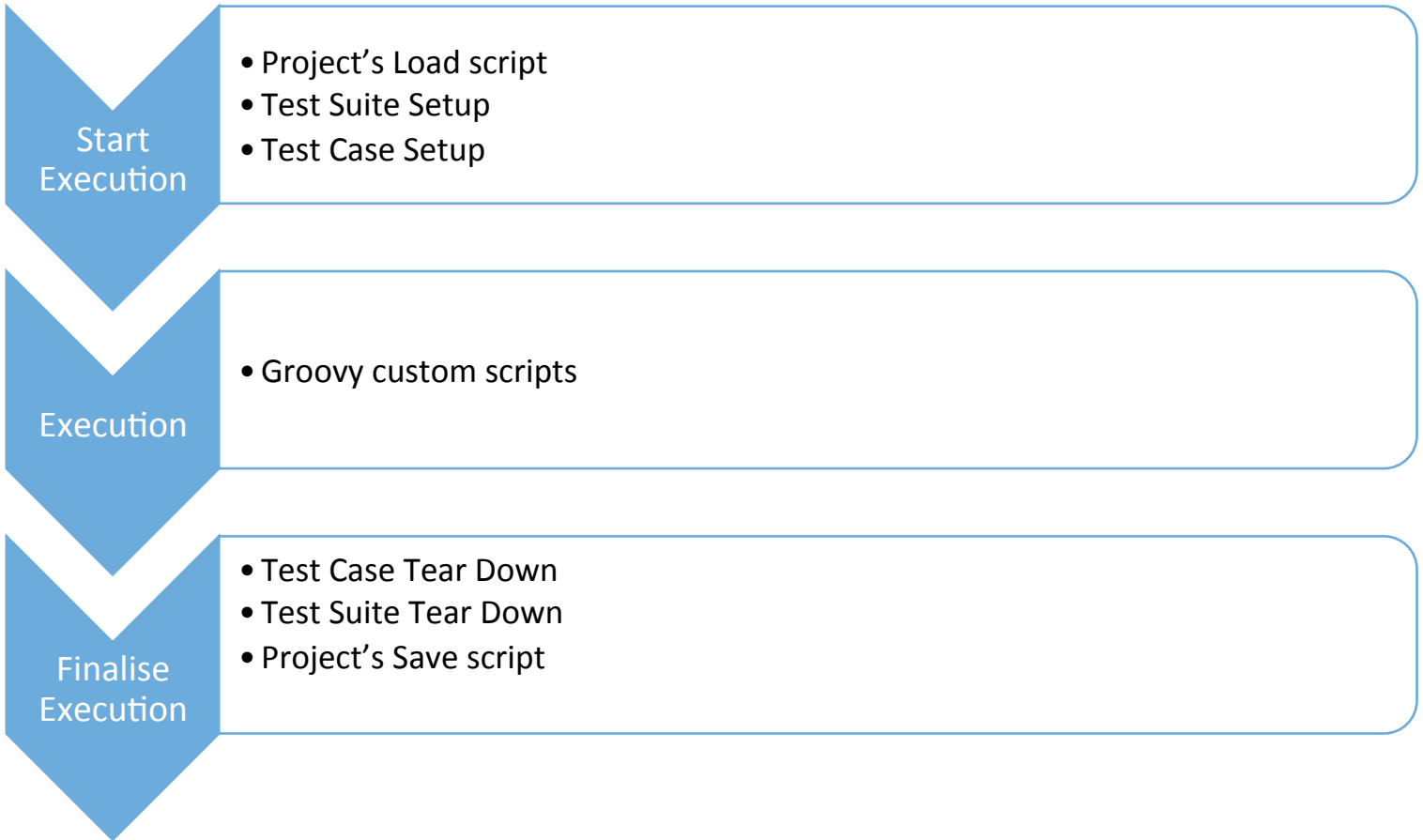
Problems with SoapUI

- Memory usage
  - Projects are saved in a single XML file
    - Size of projects – size of requests messages, size of response messages
  - Changes in request messages do not propagate
    - If you create a test suite with 100 test cases from a single SOAP operation or REST resource/method and the XML message format changes you will have (somehow) to change 100 XML messages
  - Data driven execution

Solutions and Workarounds

- Groovy scripting
  - Project's Load and Save script
  - Test Suite's Setup and TearDown scripts
  - Test Cases's Setup and TearDown scripts
  - Groovy script Test Steps

- A custom Setup Script



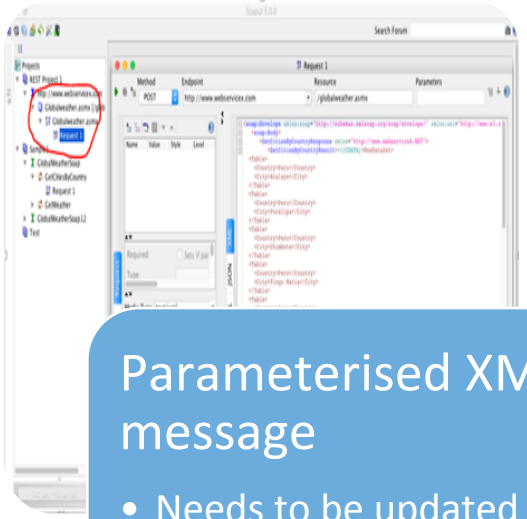
How to propagate changes to XML messages

- We need to have a template XML message
  - It needs to be parameterized

- We need a mechanism that when we execute a test case it updates the XML message (Groovy script)
  - Only the structure but keeps our parameterised data
  - Probably we don't want to manually copy and paste the Groovy code (and if we change it we want to change it automatically everywhere)

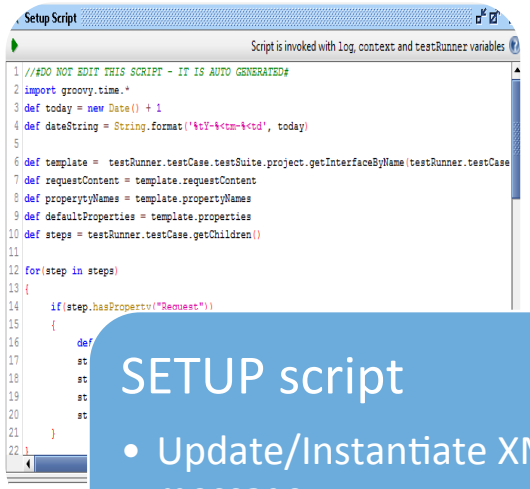
- We need a (meta)mechanism that distributes our Setup Script

How to propagate changes to XML messages



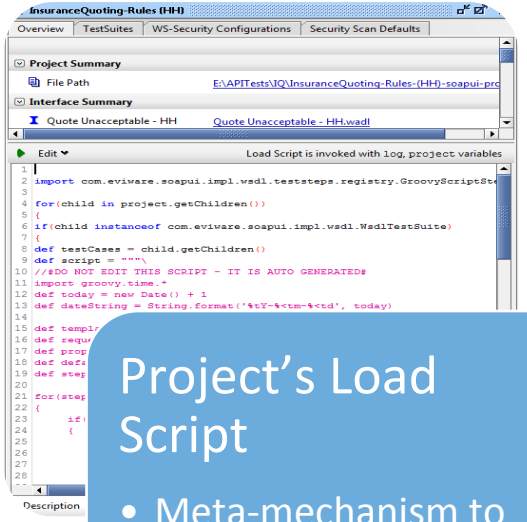
Parameterised XML message

- Needs to be updated at execution time



SETUP script

- Update/Instantiate XML message
- Updates structure
- Keeps parameterised data



Project's Load Script

- Meta-mechanism to distribute the Setup Script
- Executes when project is loaded in UI or console runner

Resources to find information for scripting

- https://www.soapui.org/apidocs/com/eviware/soapui/impl/wsdl/WsdlProject.html
- https://www.soapui.org/apidocs/com/eviware/soapui/model/testsuite/TestSuite.html

The Project’s Load script

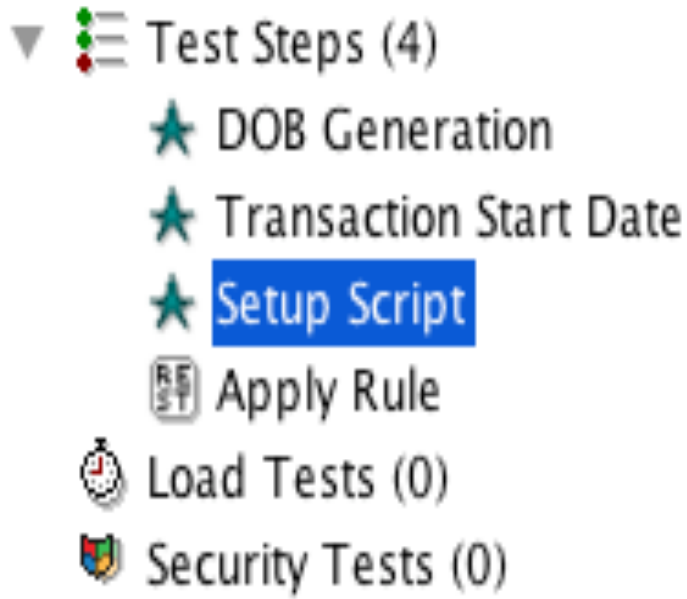
```
import com.eviware.soapui.impl.wsdl.teststeps.registry.GroovyScriptStepFactory

for(child in project.getChildren()){
  if(child instanceof com.eviware.soapui.impl.wsdl.WsdlTestSuite){
    def testCases = child.getChildren()
    def script = """\
    //#DO NOT EDIT THIS SCRIPT - IT IS AUTO GENERATED#
    …"""
   def setupTestStep = null
     for(testCase in testCases){
       try{setupTestStep = testCase.getTestStepByName('Setup Script’)}catch(ex){}

       if(setupTestStep==null){
         def numberOfSteps = testCase.getTestStepCount()
         setupTestStep = testCase.addTestStep(GroovyScriptStepFactory.GROOVY_TYPE, 'Setup Script')
         testCase.moveTestStep(numberOfSteps, -numberOfSteps)
       }
      setupTestStep.setScript(script)

    }
  }
}
```

Dynamic parameterised properties



Dynamic properties

- Dates, DOBs, age

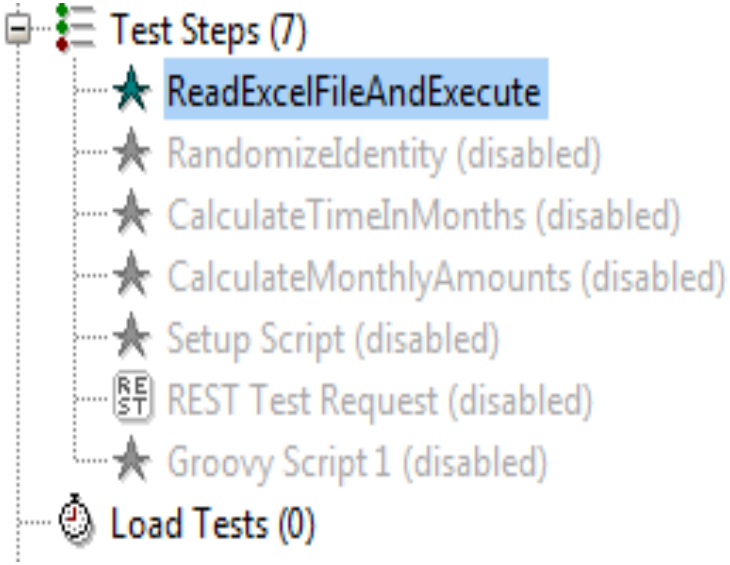
Setup script replace placeholders with values
Generated values need to be inserted in the request parameters before Setup script runs
That is why we need a custom Setup script and we do not use the TC's Setup script

Reduce size of Project file

```
def testsuites = project.testSuites
log.info testsuites.size()
for(tsentry in testsuites)
{
  ts = tsentry.getValue()
  testcases = ts.getTestCaseList()
                for(tc in testcases)
                {
                  steps = tc.getChildren()
                  for(step in steps)
                  {
                                if(step.hasProperty("Request"))
                                {
                                                step.setPropertyValue("Request", "")
                                }
                  }
                }
}
```

Data driven execution

- Feature already available in Pro version
- It can be emulated in Community version
    - Use the Apache POI libraries
    - Disable your test steps
    - Except for the loop controller



Executing a data driven loop

```
def groovyUtils =
 new com.eviware.soapui.support.GroovyUtils(context)
def myTestCase = context.testCase
def myTestSuite = context.testCase.getTestSuite()
def datafile = myTestSuite.getPropertyValue("DataFileName")
def statuses = []
def httpStatusCode = ""

ExcelReader excelReader = new ExcelReader(log:log);
List rows = excelReader.readData(datafile);
def headings = []
headings = rows.get(0);
def d = []
Iterator i = rows.iterator();
def z=0;
while( i.hasNext()){
//def z=1
//while( z<17){
        d = i.next();
        c=0;
        while(headings[c]!=null)
        {
        myTestSuite.setPropertyValue(headings[c], d[c])
        c++;
        }
        if(myTestSuite.getPropertyValue("Flag")=="Y")
        {
                testRunner.runTestStepByName( "RandomizeIdentity");
                testRunner.runTestStepByName( "CalculateTimeInMonths");
                testRunner.runTestStepByName( "CalculateMonthlyAmounts");
                testRunner.runTestStepByName( "Setup Script");
          testRunner.runTestStepByName( "REST Test Request");
          sleep(2000);

                httpResponseHeaders = context.testCase.testSteps["REST Test Request"].
                                        testRequest.response.responseHeaders

                httpStatus = httpResponseHeaders["#status#"]
                httpStatusCode = (httpStatus =~ "[1-5]\\d\\d")[0]
                statuses[z]=httpStatusCode
        }
        else
        {
                statuses[z]="Not Run"
        }
        z++;
}
```

Running tests from command line

1. Open command prompt and got to your created soapui project.

2. Run command > testrunner {SoapProjectName}.xml

3. After running test you should see some text file is generated in root.

- We can run a specific TestSuite or TestCase
  testrunner –c "APITestCase" -r SampleTest-soapui-project.xml

Source: http://roadtoautomation.blogspot.com.au/2013/08/road-to-command-line-execution-of.html

Command line runner - options

usage: testrunner [options]
-v Sets password for soapui-settings.xml file
-t Sets the soapui-settings.xml file to use
-A Turns on exporting of all results using folders instead of long filenames
-D Sets system property with name=value
-G Sets global property with name=value
-I Do not stop if error occurs, ignore them
-M Creates a Test Run Log Report in XML format
-P Sets or overrides project property with name=value
-S Saves the project after running the tests
-a Turns on exporting of all results
-c Sets the testcase
-d Sets the domain
-e Sets the endpoint
-f Sets the output folder to export results to
-h Sets the host
-i Enables Swing UI for scripts
-j Sets the output to include JUnit XML reports
-m Sets the maximum number of TestStep errors to save for each testcase
-p Sets the password
-r Prints a small summary report
-s Sets the testsuite
-u Sets the username
-w Sets the WSS password type, either 'Text' or 'Digest'
-x Sets project password for decryption if project is encrypted

Executing SoapUI tests from C#/Java

```
public static bool ExecuteSoapUI(String ProjectPath, String TestSuite, String TestCase, string[] ts_properties)
    {
        string path = Directory.GetCurrentDirectory();
        string propsFile = path + "\\tsprops.properties";
        System.IO.File.WriteAllLines(propsFile,ts_properties);

        ProcessStartInfo startInfo = new ProcessStartInfo();
        startInfo.UseShellExecute = false;
        startInfo.RedirectStandardOutput = true;
        startInfo.FileName = "C:\\Program Files\\SmartBear\\SoapUI-4.6.4\\bin\\testrunner.bat";
        startInfo.Arguments = " -Dsoapui.properties."+TestSuite.Trim()+"=\ ""+propsFile+"\ " -j -t soapui-settings.xml -s \
""+TestSuite+"\ " -c \ ""+TestCase+"\ " \ ""+ProjectPath+"\ "";
        ConsoleLogger.LogInfo(startInfo.Arguments);
        using (System.Diagnostics.Process process = Process.Start(startInfo)) {
            using (StreamReader r = process.StandardOutput) {
                string result = r.ReadToEnd();
                ConsoleLogger.LogInfo(result);
            }
            if (process.ExitCode == 0)
                return true;
            else
                return false;
        }
```

Executing SoapUI from QTP

```
Const fsoForWriting = 2

datafile = "AlertsTestDataFile.txt"

Dim objFSO
Set objFSO = CreateObject("Scripting.FileSystemObject")

'Open the text file
Dim objTextStream
Set objTextStream =
objFSO.OpenTextFile(<SoapUI Project Path> & datafile,
fsoForWriting, True)

'Set parameters values in the text file
objTextStream.WriteLine "DataFileName=" &
 DataTable("DataFileName",dtLocalSheet)

'Close the file and clean up
objTextStream.Close
Set objTextStream = Nothing
Set objFSO = Nothing
```

```
Set objShell = CreateObject("Wscript.shell")

Endpoint = GetEnvironmentParameter("Endpoint")

testsuite = DataTable("TestSuite",dtLocalSheet)
testcase = DataTable("TestCase",dtLocalSheet)

Set exec = objShell.Exec("cmd /K cd <SoapUI-Project Path> & ""C:\\Program Files\\SmartBear\
\SoapUI-5.0.0\\bin\\testrunner.bat"" -Dsoapui.properties." & testsuite & "=" & datafile & " -e " &
Endpoint & " -j -s " & testsuite & " -c " & testcase & " AlertsTest-soapui-project.xml")

wait(10)

DosWindowOutput = ""
Set oStdOut = exec.StdOut

While Not oStdOut.AtEndOfStream
   sLine = oStdOut.ReadLine
   print sLine
   DosWindowOutput = DosWindowOutput + sLine
   If inStr(1,"finished",sLine) Then
                Window("DOS_Window").Close
   End If
Wend

httpRepCode = mid(DosWindowOutput, instr(DosWindowOutput, "Receiving response: HTTP/1.1") +
20, 12)

If (instr(DosWindowOutput, "HTTP/1.1 200 OK") > 1) Then
                Reporter.ReportEvent micPass,"Execute_SoapUI_Request", "Pass. " & httpRepCode
Else
                Reporter.ReportEvent micFail,"Execute_SoapUI_Request", "Fail. " & httpRepCode
End IF
```

Planit
SOFTWARE TESTING

AU: 1300 992 967
infoau@planittesting.com

NZ: 0800 752 648
infonz@planittesting.com

UK: 0203 356 2870
infoau@planittesting.com

www.planittesting.com