Sperasoft
Your game dev partner

Introducing soapUI http://www.soapui.org

SOAP and REST services: main QA aspects

- Compliance to protocols' standards
- Functional testing
  - API functions tests with supported parameters range
  - Negative tests
- Security testing
- Load and Performance testing
- Usability testing
- Documentation and Logging

Most of these types can be tested with soapUI.

- It supports SOAP, REST and regular Web services via HTTP protocol
- It has a multi-OS test-runner that can be integrated into a build server

Main elements

Available elements of a soapUI project

- **Web Service Description Language (.wsdl) file**
- A default config element for Simple Object Access Protocol (SOAP) services
- **Web Application Description Language (.wadl) file**
- A default config element for REpresentation State Transfer (REST) services
- **REST Service**
- A config element of a REST service, created manually
- **Mock Service**
- A config element of a Stub Service that can emulate several operations (see below)
- **Test Suite**
- An element containing Test Cases and Web Test Cases (see below)
- Can contain Setup and TearDown scripts

What types of Test Cases does soapUI support?

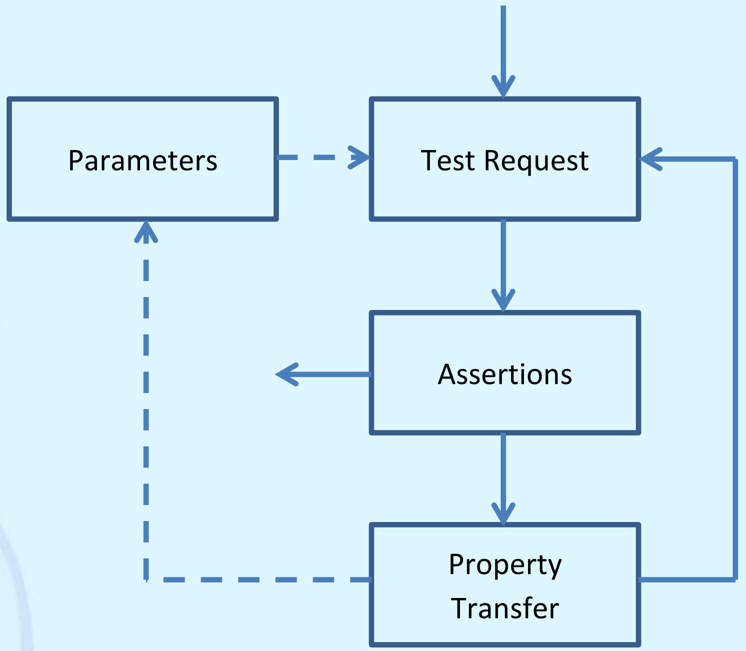
- **Test Case**
  - A set of requests to any service/server
  - Includes test steps, load tests and security tests
  - Can contain Setup and TearDown scripts
- **Web Test Case**
  - A set of requests to a web server with support of HTTP recording
  - Includes test steps, load tests and security tests
  - Can contain Setup and TearDown scripts

What types of Test Steps does soapUI support?

- **Test Request** – a request to a SOAP service
- **REST Test Request** – a request to a REST service
- **HTTP Test Request** – a request to a HTTP server
- **JDBC Request** – a query to a Database
- **Property Transfer** – a special step allowing to transfer parameters between other Test Steps
- **Groovy Script** – a script that can do any action
- **Delay** – a pause
- **Conditional Goto** – goes to a given step if an XPath expression applied to the previous step returns true; otherwise goes to the next step
- **Security Test** – a test request with specific parameters and assertions
- **Load Test** – a set of test requests with specific statistics
- **Etc…**

Main testing cycle

How are most of test cases written?



Parameters

- Three-level hierarchy: Project level, Test Suite level, Test Case level
- Accessible from Property Transfer elements, from Groovy Scripts and from any place as expressions ${#Level#Name}

What elements are in Test Requests?

- **Resource/Method** (for SOAP/REST requests) or **EndPoint** (for Web request)
- **A list of pre-defined parameters with values:**
- *Template* parameters – <endpoint>/<path>/val1/val2
- *Query* parameters – <endpoint>/<path>?par1=val1&par2=val2
- *Matrix* parameters – <endpoint>/<path>;par1=val1,val2
- *Header* parameters – par1: val1
- **Accept Header**
- **Content-Type Header** (for requests with content)
- **Additional Headers and Assertions (see below)**
- **Etc…**
- **Response** – a result of a request, which can be presented in XML, JSON, HTML or Raw format

What are main types of assertions?

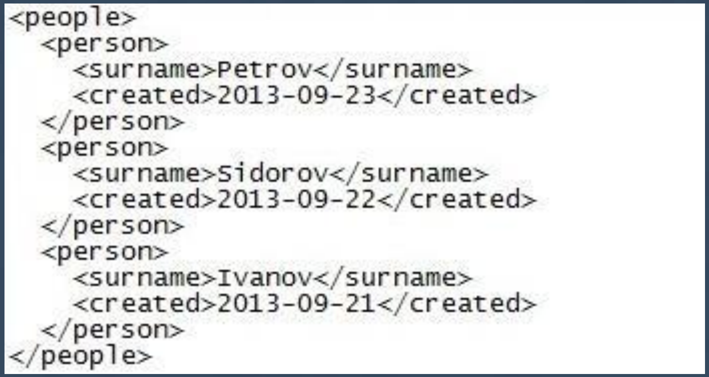
- **Assertions**

- *Contains / Not Contains* – checks if a response contains / does not contain a given fragment. Allows regular expressions
- *XPath Match* – checks if a part of a response, obtained using XPath query, equals to a given fragment. Allows wildcards
- *XQuery Match* – checks if a part of a response, obtained using XQuery expression, equals to a given fragment. Allows wildcards
- *Valid HTTP Status Codes / Invalid HTTP Status Codes* – allows to specify a list of valid / invalid response codes
- *Script Assertion* – allows to check any response element using a groovy script
- Etc…

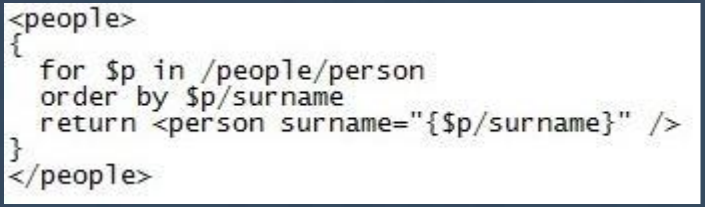
Main testing cycle

XQuery assertion?

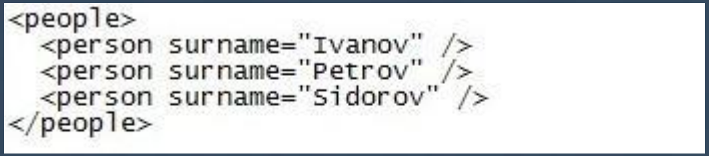
- Supports XPath and XML insertions
- Can convert nodes to attributes and vice versa
- Can return a part of xml tree
- Allows sorting
- Has a recurrent structure



```
<people>
  <person>
    <surname>Petrov</surname>
    <created>2013-09-23</created>
  </person>
  <person>
    <surname>Sidorov</surname>
    <created>2013-09-22</created>
  </person>
  <person>
    <surname>Ivanov</surname>
    <created>2013-09-21</created>
  </person>
</people>
```



```
<people>
{
  for $p in /people/person
  order by $p/surname
  return <person surname="{$p/surname}" />
}
</people>
```



```
<people>
  <person surname="Ivanov" />
  <person surname="Petrov" />
  <person surname="Sidorov" />
</people>
```

How to transfer properties?

- **Property Transfer**
- Can transfer fragments of a test request object to pre-created parameters (in its hierarchy) or directly to another request
- Can use XPath or XQuery when transferring, or transfer the whole response
- Can transfer text content of a node or an XML tree
- Supports JSON responses as well as XML ones
- Using **Groovy Scripts** for transferring properties
- Can transfer wider set of values
- Can transfer to any pre-created parameter

Security tests

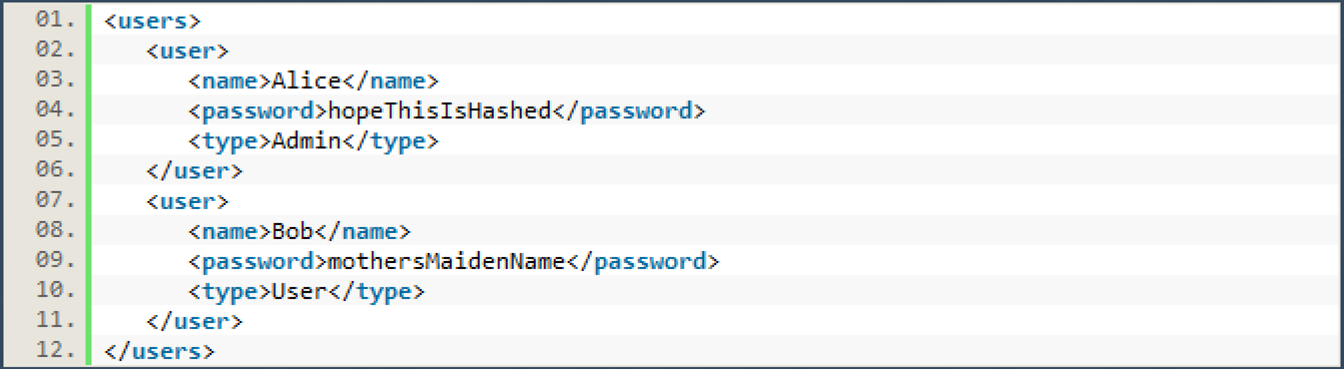
What attacks are you able to simulate?

SQL Injection : tries to exploit bad database integration coding.

statement = "SELECT * FROM `users` WHERE `name` = '" + userName + "';“

userName = ' or '1'='1

XPath Injection : tries to exploit bad XML processing inside your target service



```
<users>
    <user>
        <name>Alice</name>
        <password>hopeThisIsHashed</password>
        <type>Admin</type>
    </user>
    <user>
        <name>Bob</name>
        <password>mothersMaidenName</password>
        <type>User</type>
    </user>
</users>
```

String xpathQuery = "//user[name/text()='" + request.get("username") + "' And password/text()='" + request.get("password") + "']";

userName = lol' or 1=1 or 'a'='a

Boundary Scan/Ivalid types : tries to exploit bad handling of values that are outside of defined ranges or of different type, e.g.:

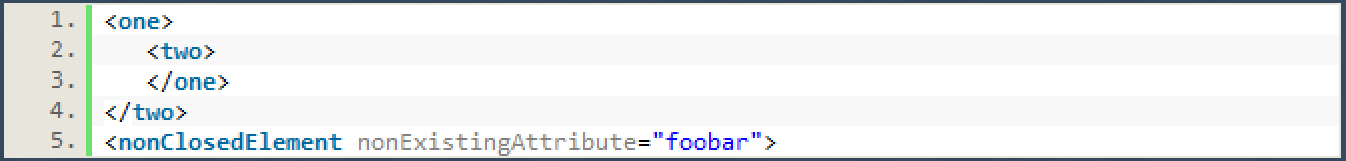
xsd:min, xsd:max, xsd:length, xsd:minInclusive, xsd:maxInclusive, xsd:minExclusive, xsd:maxExclusive, xsd:totalDigits, xsd:fractionDigits

Continuation on the next page…

Security tests

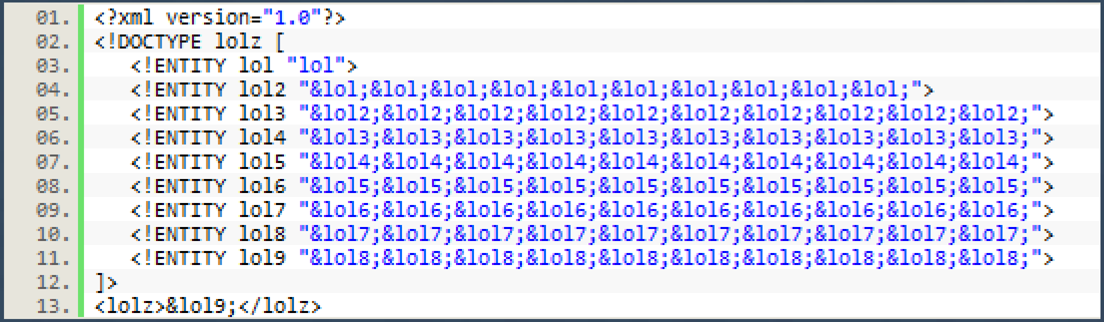
What attacks are you able to simulate?

Malformed XML : tries to exploit bad handling of invalid XML on your server or in your service



```
<one>
    <two>
    </one>
</two>
<nonClosedElement nonExistingAttribute="foobar">
```

XML Bomb : tries to exploit bad handling of malicious XML request (be careful)



```
<?xml version="1.0"?>
<!DOCTYPE lolz [
   <!ENTITY lol "lol">
   <!ENTITY lol2 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
   <!ENTITY lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
   <!ENTITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
   <!ENTITY lol5 "&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
   <!ENTITY lol6 "&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
   <!ENTITY lol7 "&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
   <!ENTITY lol8 "&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
   <!ENTITY lol9 "&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">
]>
<lolz>&lol9;</lolz>
```

Malicious Attachment : tries to exploit bad handling of attached files

- Corrupted or very large files intended to make the server to crash.
- Files containing code that is harmful for the server or server to execute/parse, i.e. a virus targeted at the server.

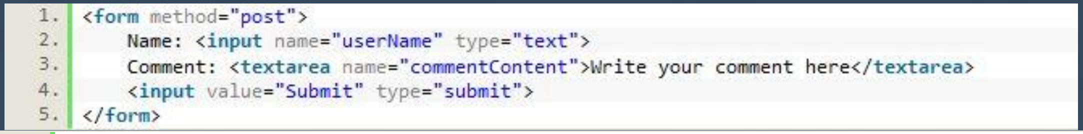
The Malicious Attachment Security Scan allows generation of corrupt files as well as attachment of user-selected files.

Continuation on the next page…

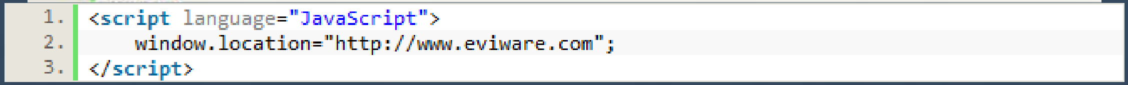
Security tests

What attacks are you able to simulate?

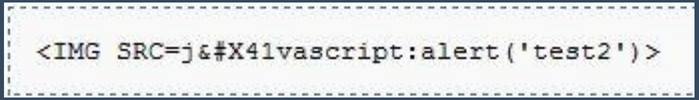
Cross Site Scripting (XSS): tries to find cross-site scripting vulnerabilities



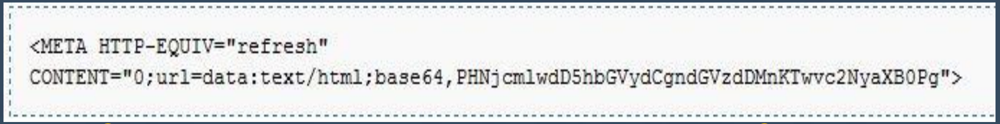
```
<form method="post">
    Name: <input name="userName" type="text">
    Comment: <textarea name="commentContent">Write your comment here</textarea>
    <input value="Submit" type="submit">
</form>
```



```
<script language="JavaScript">
    window.location="http://www.eviware.com";
</script>
```



```
<IMG SRC=j&#X41vascript:alert('test2')>
```

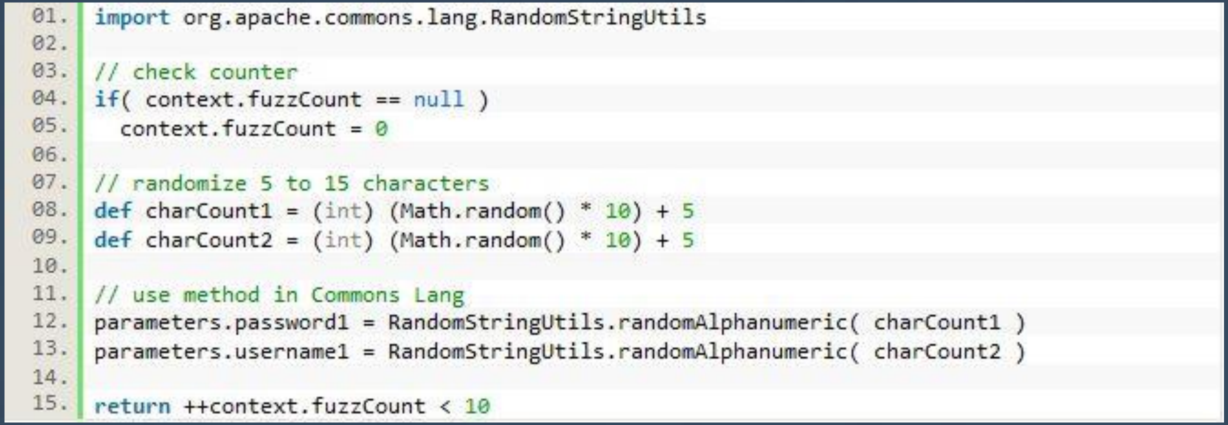


```
<META HTTP-EQUIV="refresh"
CONTENT="0;url=data:text/html;base64,PHNjcmlwdD5hbGVydCgndGVzdDMnKTwvc2NyaXB0Pg">
```

Custom Script : allows you to use a script for generating custom parameter fuzzing values

- The Custom Scan follows the basic model of the other parameter-based Security Scans but requires you to specify a script (*Groovy*, *Javascript* or *Java*) that will provide the values to send for each permutation, giving you maximum flexibility with how you can provoke your target services.

e.g.: ***fuzzling*** test



```
import org.apache.commons.lang.RandomStringUtils

// check counter
if( context.fuzzCount == null )
  context.fuzzCount = 0

// randomize 5 to 15 characters
def charCount1 = (int) (Math.random() * 10) + 5
def charCount2 = (int) (Math.random() * 10) + 5

// use method in Commons Lang
parameters.password1 = RandomStringUtils.randomAlphanumeric( charCount1 )
parameters.username1 = RandomStringUtils.randomAlphanumeric( charCount2 )

return ++context.fuzzCount < 10
```

Performance tests

What are performance tests aiming at?

Validation of:
- speed
- scalability
- stability characteristics

By means of assessing:
- response times
- throughput
- resource-utilization levels

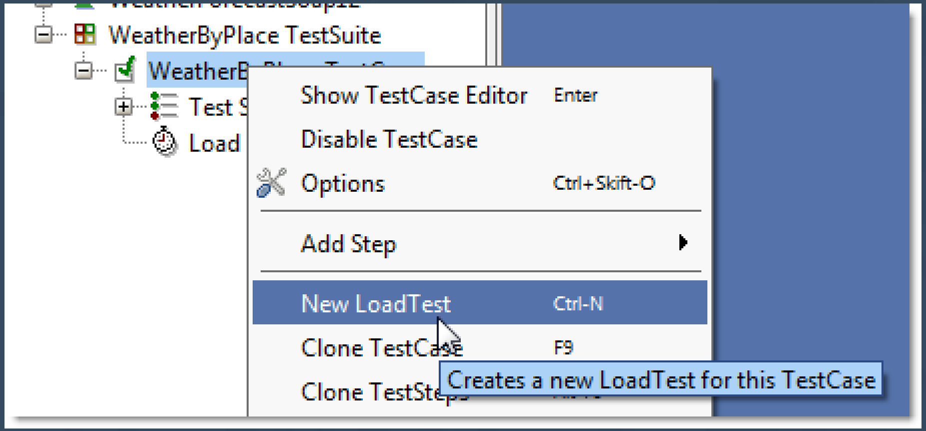
- Key types of performance tests

Term	**Purpose**
Performance test	To determine or validate speed, scalability, and/or stability.
Load test	To verify application behavior under normal and peak load conditions.
Stress test	To determine or validate an application's behavior when it is pushed beyond normal or peak load conditions.
Capacity test	To determine how many users and/or transactions a given system will support and still meet performance goals.

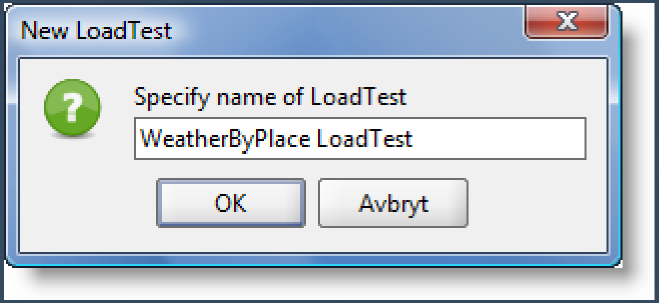
It's all about the *load model* that you choose…

Performance tests

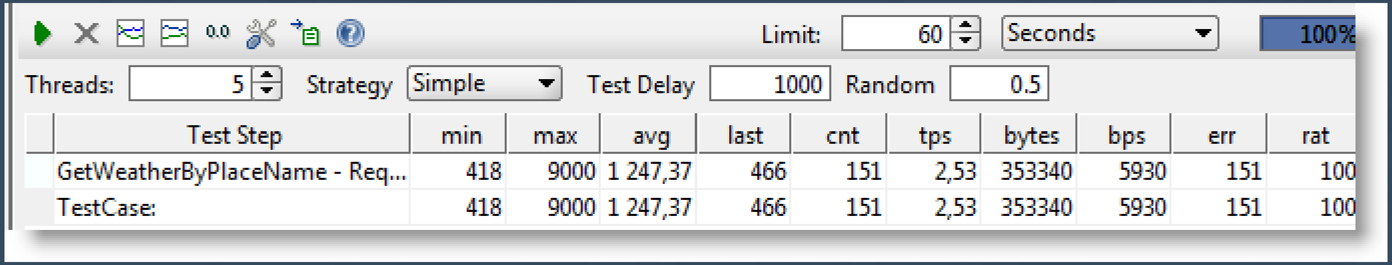
Simple performance test in soapUI



1



2



Limit: 60 Seconds 100%

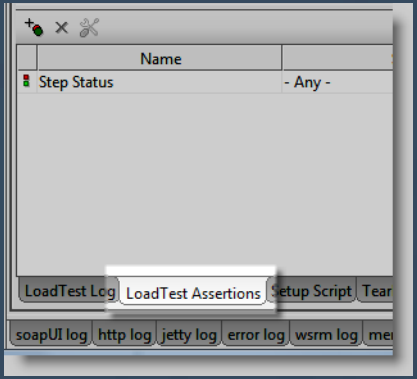
Threads: 5 Strategy: Simple Test Delay: 1000 Random: 0.5

Test Step	min	max	avg	last	cnt	tps	bytes	bps	err	rat
GetWeatherByPlaceName - Req...	418	9000	1 247,37	466	151	2,53	353340	5930	151	100
TestCase:	418	9000	1 247,37	466	151	2,53	353340	5930	151	100

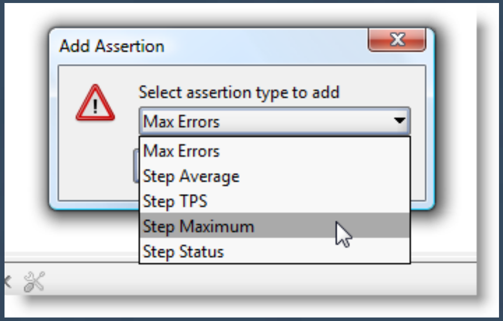
Profit!

Performance tests

And so what? Assertions!



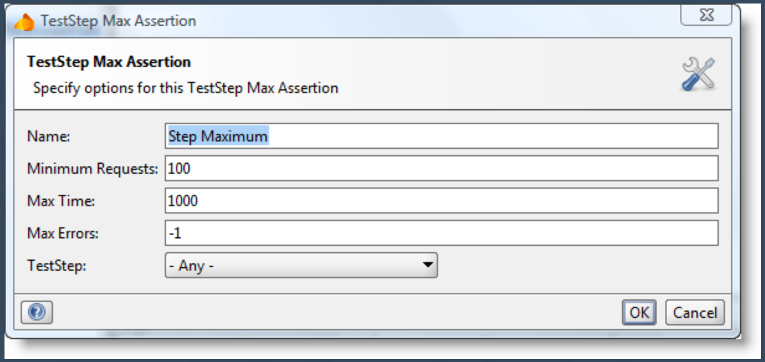
Name	
Step Status	- Any -



Add Assertion

Select assertion type to add

Max Errors



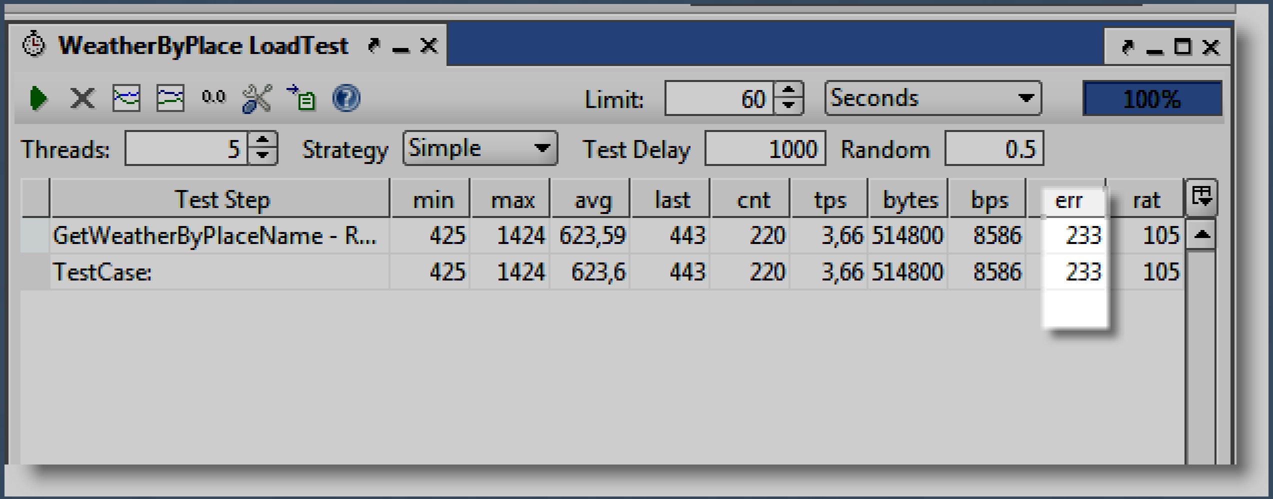
TestStep Max Assertion

TestStep Max Assertion

Specify options for this TestStep Max Assertion

Name:	Step Maximum
Minimum Requests:	100
Max Time:	1000
Max Errors:	-1
TestStep:	- Any -

We allowed a max response of one second, 1000 milliseconds. And we see that number of errors is growing since responses take much more time.



WeatherByPlace LoadTest

Limit: 60 Seconds 100%

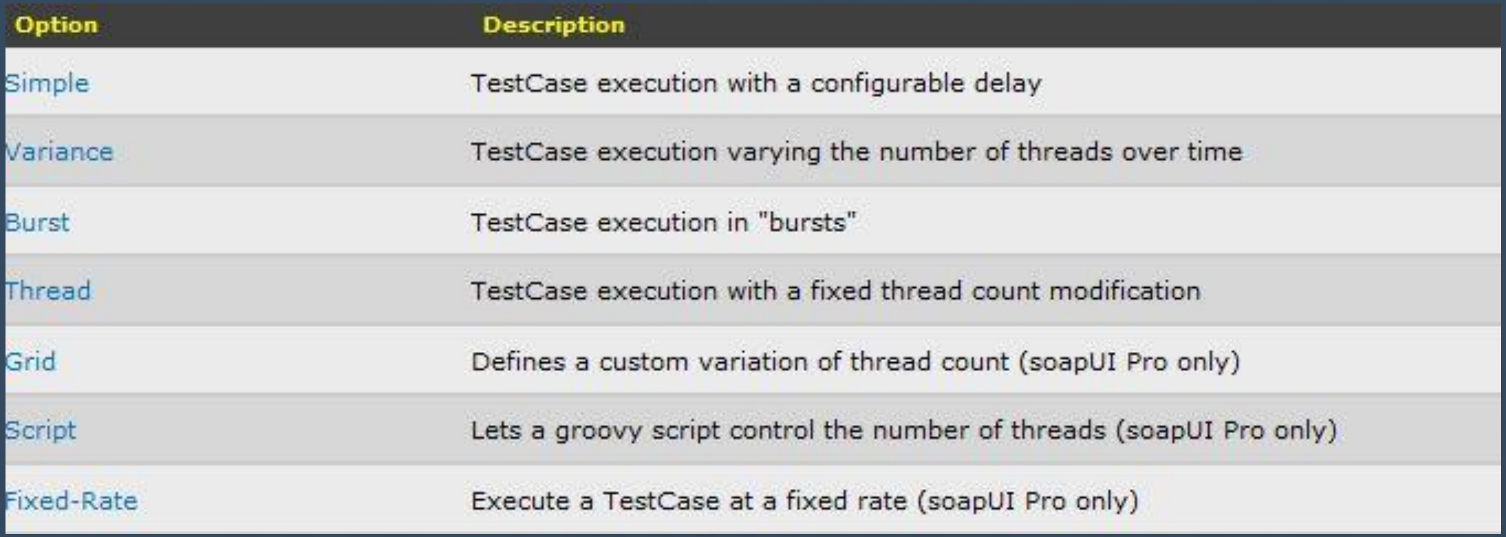
Threads: 5 Strategy Simple Test Delay 1000 Random 0.5

Test Step	min	max	avg	last	cnt	tps	bytes	bps	err	rat
GetWeatherByPlaceName - R...	425	1424	623,59	443	220	3,66	514800	8586	233	105
TestCase:	425	1424	623,6	443	220	3,66	514800	8586	233	105

Create more complicated *strategies* and *models*, take reports, it’s all in soapUI…

Performance tests

Load Strategies



Option	Description
Simple	TestCase execution with a configurable delay
Variance	TestCase execution varying the number of threads over time
Burst	TestCase execution in "bursts"
Thread	TestCase execution with a fixed thread count modification
Grid	Defines a custom variation of thread count (soapUI Pro only)
Script	Lets a groovy script control the number of threads (soapUI Pro only)
Fixed-Rate	Execute a TestCase at a fixed rate (soapUI Pro only)

Choose load strategy corresponding your load model.
More info on strategies: http://www.soapui.org/Load-Testing/strategies.html

What is a Mock Service?

According to the Cambridge Dictionary something that is *“mocked”* is:

“Not real but appearing or pretending to be exactly like something”

So we are essentially talking about something that will **not** behave as a real service, but will only **mimic** the behavior of the service.

A mock service is **not** the same as a full service **simulation**. A mock will only simulate a part, perhaps one specific interaction, of a system. While a service simulator will simulate the entire system and behave in an expected way for all calls.

Why should you mock a service?

- **The real service is not implemented**
- While serial development usually sux (slow)
- **Services out of your control:**
- Test data
- Life cycle
- Availability & Access
- Negative scenarios
- **Charged services**
- **Prototyping**
- **3rd-party Consumers**

How does soapUI help?

- **What do you need to run a mocked service?**
- A service contract (WSDL) to mock
- Specify port to run the mock on from soapUI
- Generate responses you need (positive or negative, static or dynamic)
- Launch your mock
- **What is your mock good for?**
- A MockService can simulate any number of WSDL contracts
- Built in scripting functionality (Groovy) helps simulate almost any desired behavior
- Fixed responses, random errors, dynamic results, etc.
- **How is your mock managed and hosted?**
- You may run it from soapUI tool GUI
- You may run it from command-line (Java-based multi-OS runner)
- You may deploy it to a standard servlet container as a WAR

Thanks for watching!

What have we learned?

- Types of testing soapUI can perform
- Main soapUI entities: configuration elements and test suites
- Types of test cases
- Types of test requests and their main fields
- Types of assertions
- Means for transferring properties
- Security testing with soapUI
- Load testing with soapUI
- Mocking services

Questions?