Advanced Object-Oriented Design

Lecture 13

Unit Testing the Web

Bartosz Walter

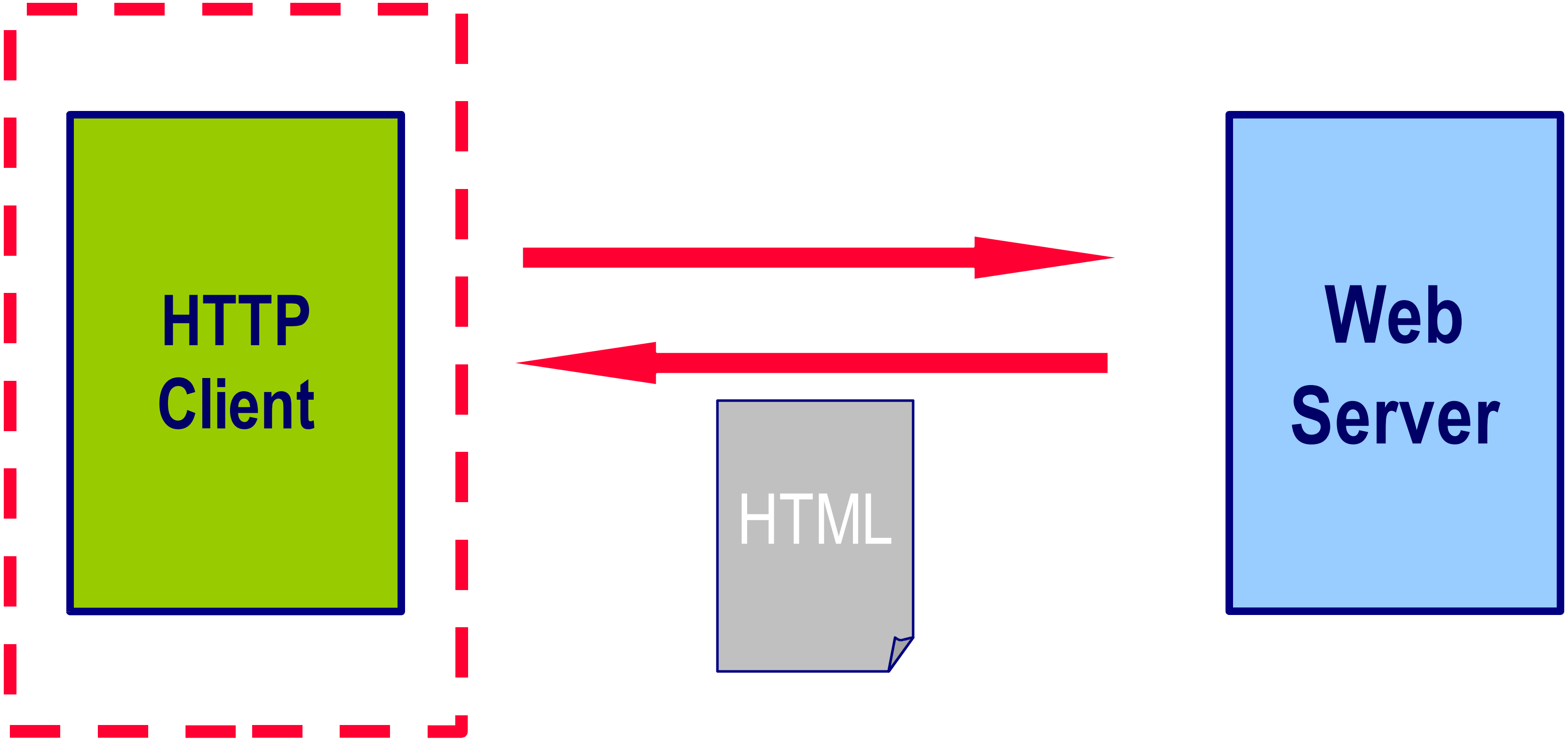
<Bartek.Walter@man.poznan.pl>

Testing the Web

- **Functional testing: HttpUnit**
- **In-container testing: Jakarta Cactus**
- **Code logic testing: Mock Objects**

Testing the Web

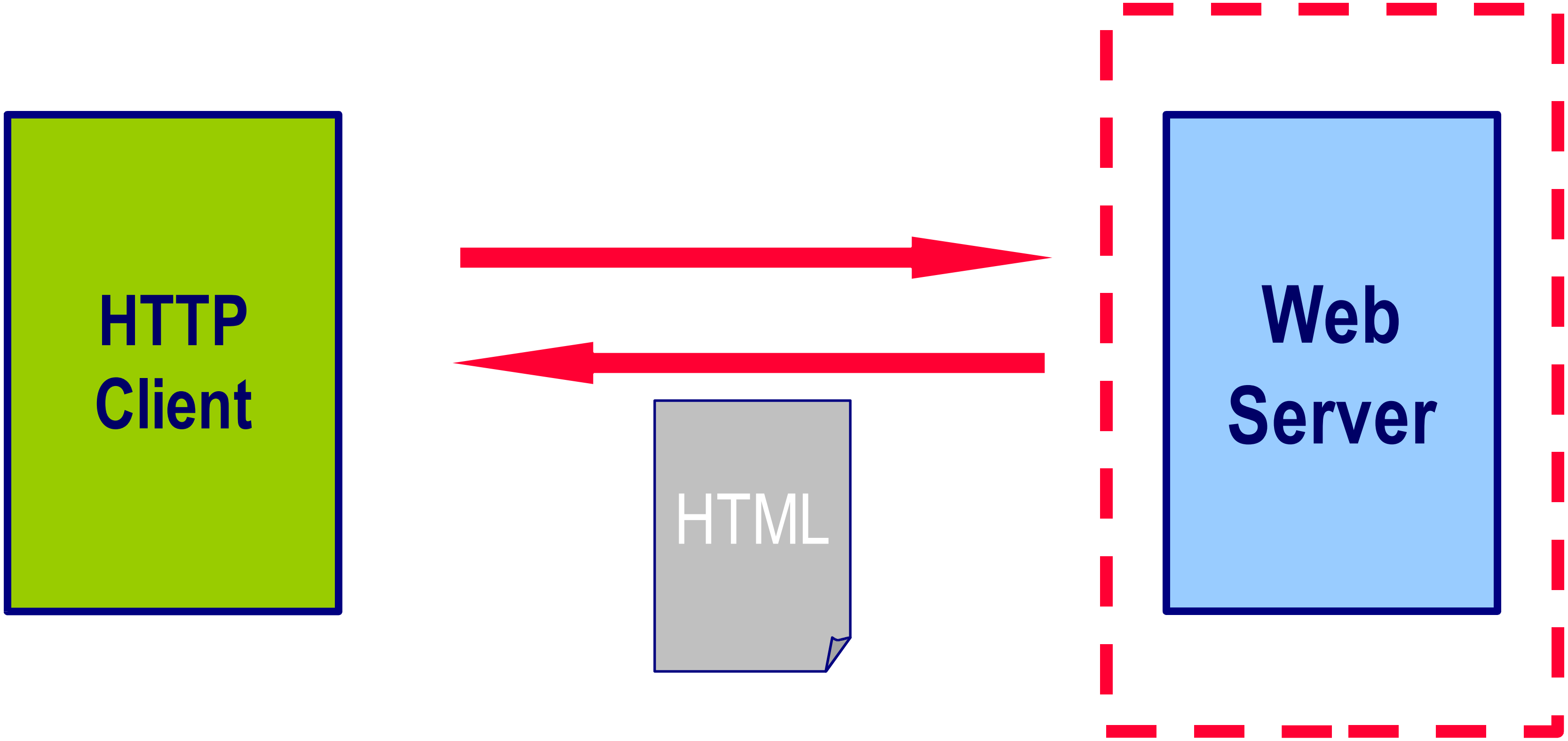
HttpUnit



Client can access the HTTP interface only.

Testing the Web

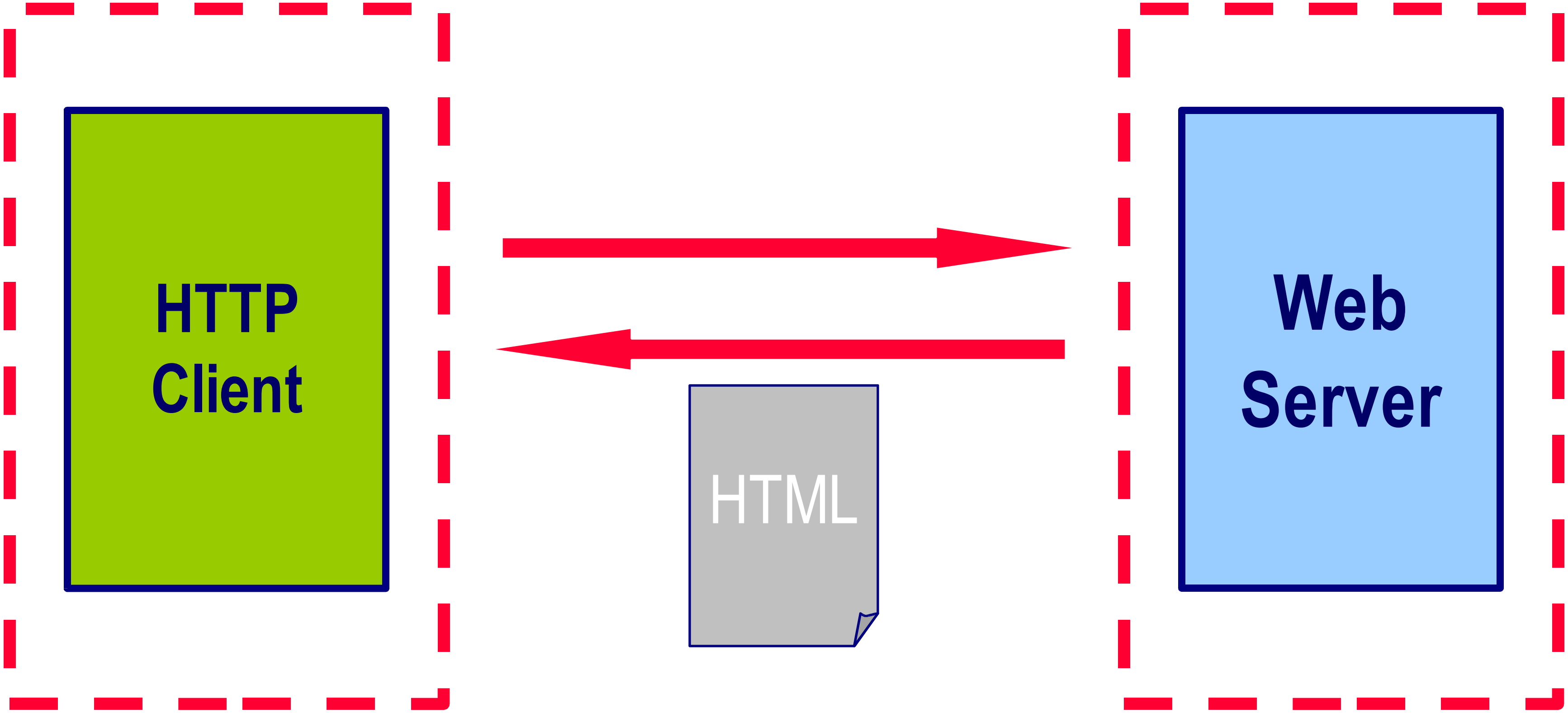
Mock Objects



Mock objects emulate the environment objects.

Testing the Web

Jakarta Cactus

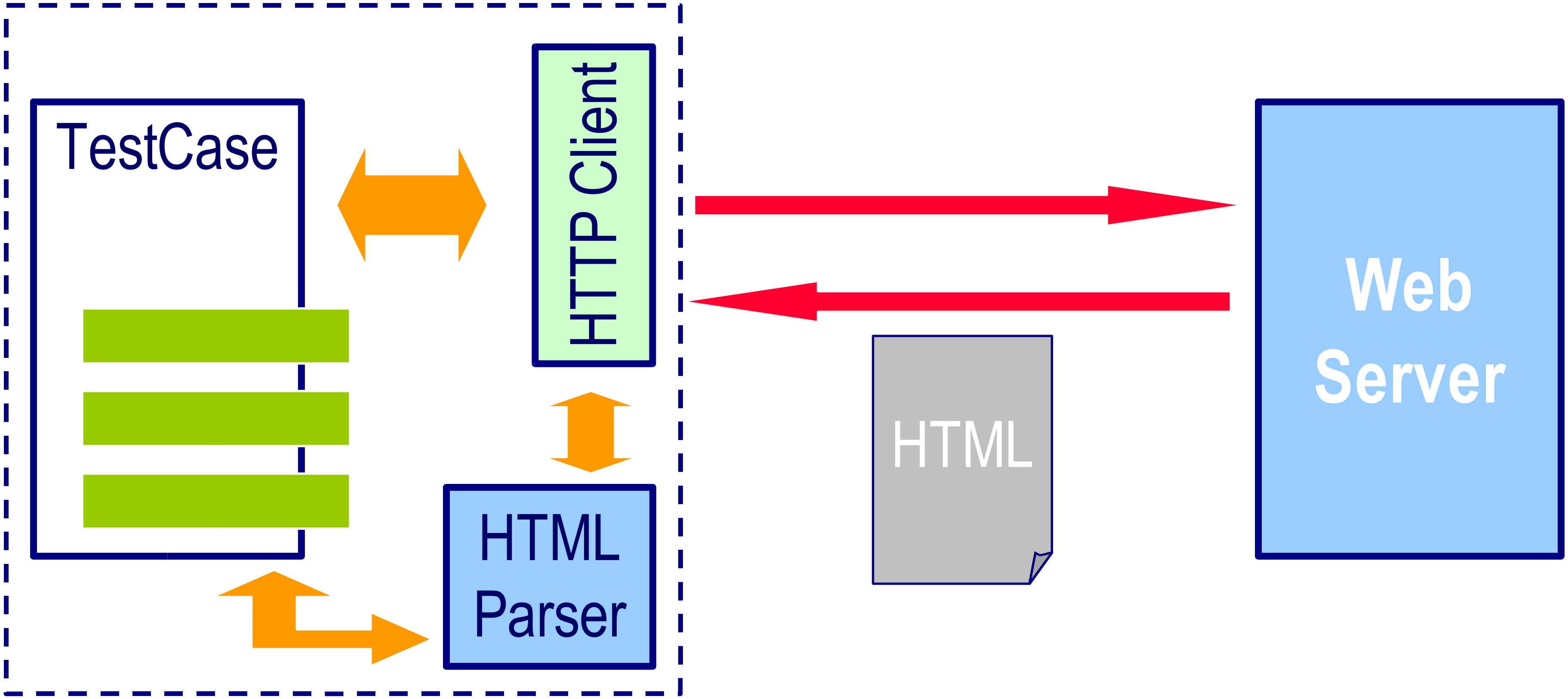


Tests are both server- and client-aware.

Http Unit

- A library for the client-side unit-testing of the web server
- Tests do not rely on the server implementation.
- http://httpunit.sourceforge.net/

Unit Testing at the Client Side



WebConversation

- HTTP Client embedded into the HttpUnit
- Acts like a web browser
- Maintains the session context
- Talks to web servers sending requests and obtaining responses

```
WebConversation wc = new WebConversation();
WebRequest req = new GetMethodWebRequest(
    "http://www.meterware.com/testpage.html");
WebResponse resp = wc.getResponse( req );
```

WebConversation API

- WebResponse getResponse(WebRequest request)
- WebResponse getResponse(String urlString)
- String getHeaderField(String fieldName)
- ClientProperties getClientProperties()
- void addCookie(String name, String value)
- WebWindow getMainWindow()
- WebWindow getOpenWindow(String name)
- String[] getFrameNames()

WebRequest

- Represents the HTTP request
- It can be set up manually
- Specific subclasses handle GET, POST & PUT

```
WebConversation wc = new WebConversation();
WebRequest req = new GetMethodWebRequest(
    "http://www.meterware.com/testpage.html");
WebResponse resp = wc.getResponse( req );
```

WebRequest API

- `void setParameter(String name, String value)`
- `void setParameter(parameterName, UploadFileSpec[] files)`
- `void setImageButtonClickPosition(int x, int y)`
- `void setHeaderField(String name, String value)`
- `void selectFile(String name, File file)`
- `java.net.URL getURL()`
- `java.util.Dictionary getHeaders()`

WebResponse

- Represents the HTTP response
- Can be processed both as plain text and as DOM

```
WebConversation wc = new WebConversation();
WebRequest req = new GetMethodWebRequest(
    "http://www.meterware.com/testpage.html");
WebResponse resp = wc.getResponse( req );
```

WebResponse API

- int getContentLength()
- String getContentType()
- org.w3c.dom.Document getDOM()
- HTMLElement[] getElementsWithName(String name)
- HTMLElement getElementWithID(String id)
- WebForm getFirstMatchingForm
  (HTMLElementPredicate predicate, Object criteria)
- WebLink getFirstMatchingLink
  (HTMLElementPredicate predicate, Object criteria)
- WebTable getFirstMatchingTable
  (HTMLElementPredicate predicate, Object criteria)

WebResponse API (cont.)

- `String getHeaderField(String fieldName)`
- `WebImage[] getImages()`
- `java.io.InputStream getInputStream()`
- `int getResponseCode()`
- `String getText()`
- `String getTitle()`
- `java.net.URL getURL()`
- `boolean isHTML()`

Navigation

WebResponse

- `WebLink getLinkWith(String text)`
- `WebLink getLinkWithImageText(String text)`
- `WebLink getLinkWithName(String name)`

WebLink

- `void mouseOver()`
- `WebResponse click()`

Navigation: Example

```
WebConversation wc = new WebConversation();

WebResponse resp = wc.getResponse(url);
   // read this page

WebLink link = resp.getLinkWith("response");
   // find the link

link.click(); // follow it

WebResponse jdoc = wc.getCurrentPage();
   // retrieve the referenced page
```

source: HttpUnit Home Page

Tables

WebResponse

- `WebTable[] getTables()`
- `WebTable getTableStartingWith(String text)`
- `WebTable getTableWithID(String text)`
- `WebTable getTableWithSummary(String text)`

WebTable

- `String getCellAsText(int row, int column)`
- `int getColumnCount()`
- `int getRowCount()`

Tables: Example

```
WebTable table = resp.getTables()[0];
assertEquals("rows", 4, table.getRowCount());
assertEquals("columns", 3, table.getColumnCount());
assertEquals("links", 1, table.getTableCell(0, 2)
    .getLinks().length);
String[][] colors = resp.getTables()[1].asText();
assertEquals("Name", colors[0][0]);
assertEquals("Color", colors[0][1]);
assertEquals("gules", colors[1][0]);
assertEquals("red", colors[1][1]);
assertEquals("sable", colors[2][0]);
assertEquals( "black", colors[2][1])
```

source: HttpUnit Home Page

Forms

WebResponse

- `WebForm[] getForms()`
- `WebForm getFormWithID(String ID)`
- `WebForm getFormWithName(String name)`

WebForm

- `String getAction()`
- `Button getButtonWithID(String buttonID)`
- `String getMethod()`
- `String getParameterValue(String name)`
- `boolean isXXXParameter(String name)`
- `void setParameter(String name, String[] values)`
- `WebResponse submit(SubmitButton button)`

Forms: Example

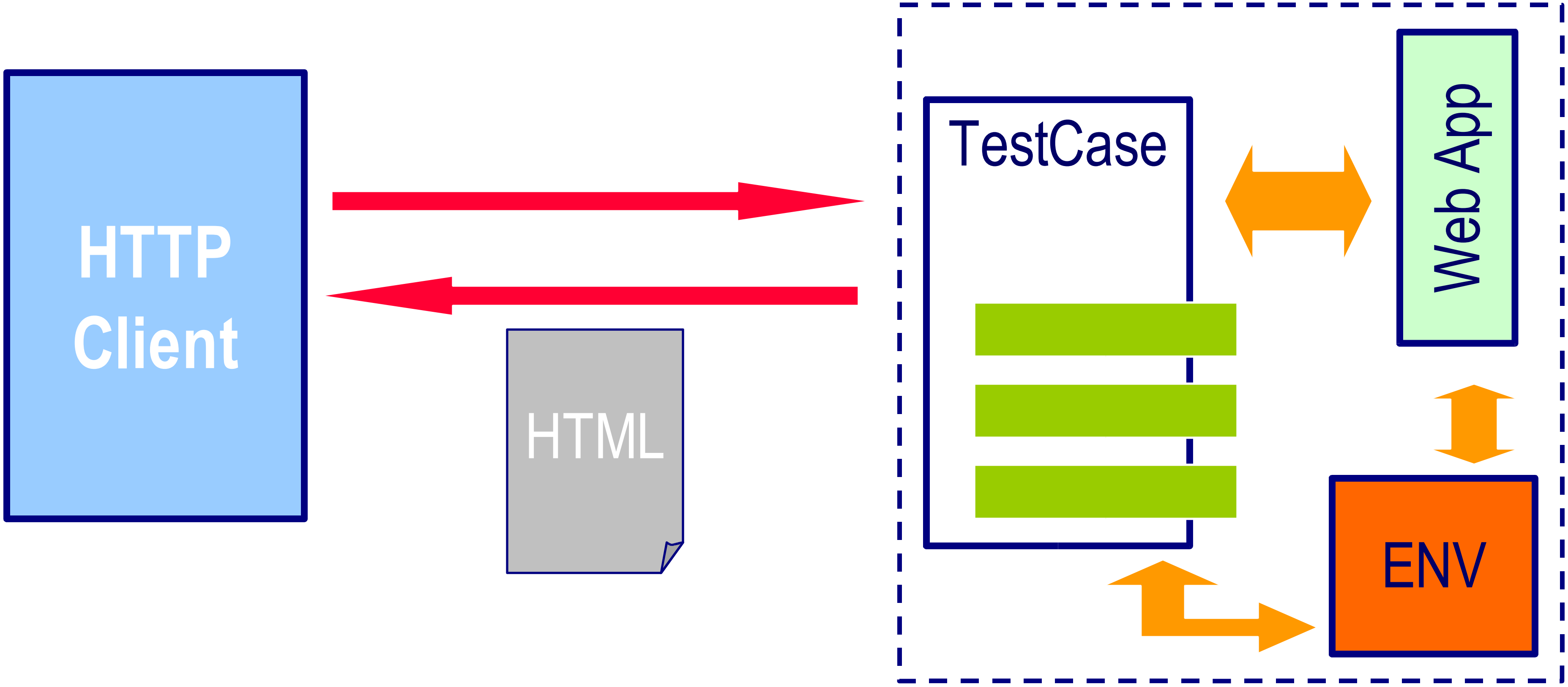
```
WebForm form = resp.getForms()[0];
    // select the first form in the page

assertEquals("La Cerentolla",
    form.getParameterValue( "Name"));

assertEquals("Chinese",
    form.getParameterValue( "Food"));

assertEquals("Manayunk",
    form.getParameterValue( "Location"));

form.setParameter("Food", "Italian");
    // select one of the permitted values for food

form.removeParameter("CreditCard");
    // clear the check box

form.submit(); // submit the form
```

source: HttpUnit Home Page

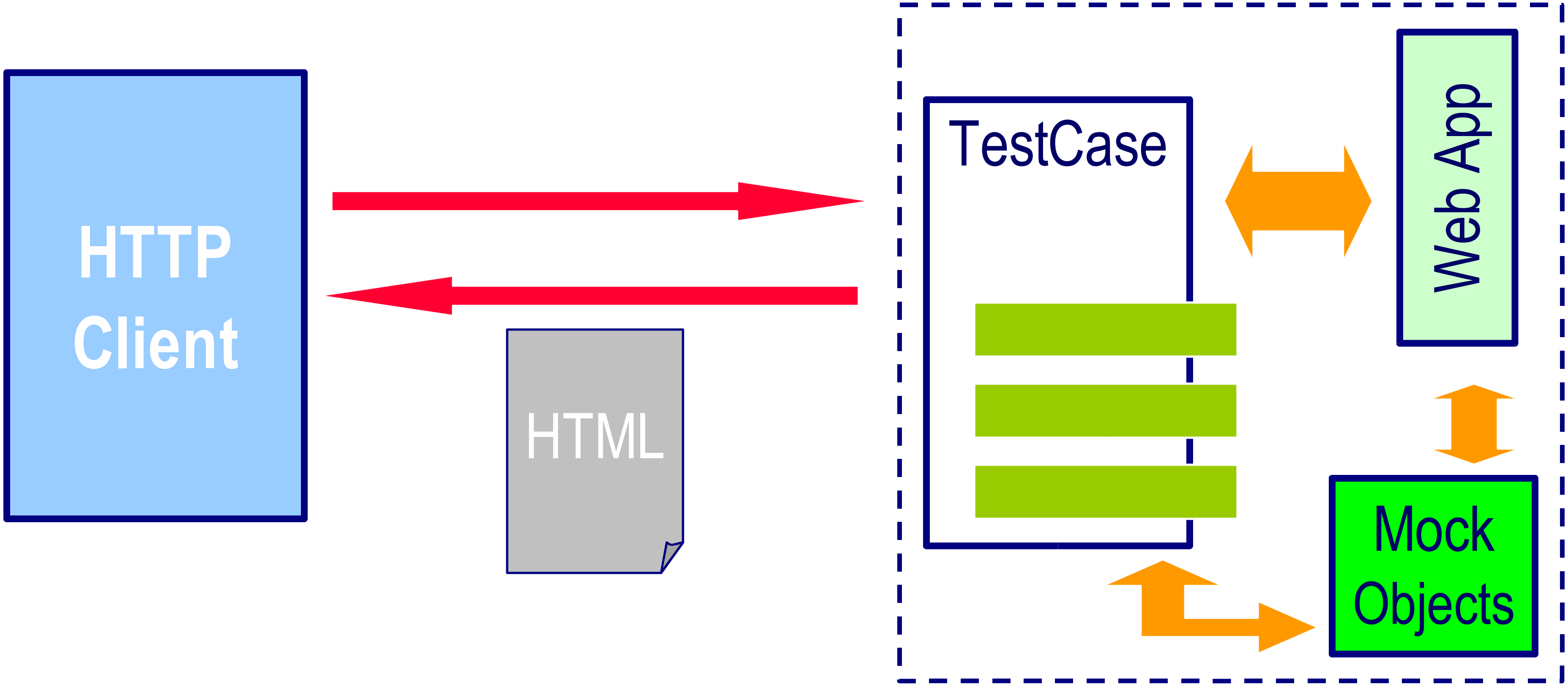
Mock Objects

- A library of objects emulating environment objects for the server-side unit-testing
- MO heavily depend on the technology used
- Ready-to-use mock implementation of several technologies
- http://mockobjects.sourceforge.net/

Unit Testing at the Server Side



Unit Testing at the Server Side



Mock Objects

A mock object is a "double agent" used to test the behaviour of other objects.

- acts as a faux implementation of an interface or class that mimics the external behaviour of a true implementation
- observes how other objects interact with its methods and compares actual behaviour with preset expectations.

source: www.mockobjects.com

Testing the mocks

- When a discrepancy occurs, a mock object can interrupt the test and report the anomaly.
- If the discrepancy cannot be noted during the test, a verification method called by the tester ensures that all expectations have been met or failures reported.

Testing process with mocks

The common style for testing with mock objects:

- Create instances of mock objects
- Set state and expectations in the mock objects
- Invoke domain code with mock objects as parameters
- Verify consistency in the mock objects

MockHttpServletRequest

- Represents the HTTP request
- It can be set up manually
- Stores both expected and actual data

```
String getContentType()

void setupGetContentType(String aContentType)

void setContentType(String contentType)

void setExpectedContentType
(String aContentType)
```

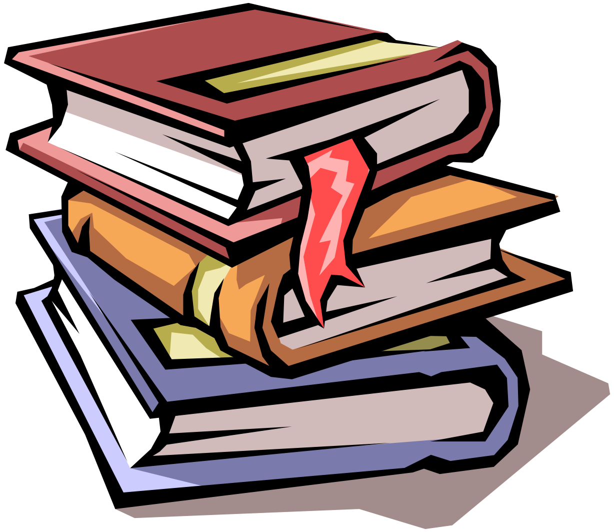
Mock Objects: Example

```
public void setUp() {
   MockHttpServletRequest myMockHttpRequest =
      new MockHttpServletRequest();
   MockHttpServletResponse myMockHttpResponse =
      new MockHttpServletResponse();
   MockServletConfig myMockServletConfig =
      new MockServletConfig();
   MyServlet myServlet = new MyServlet();
}
```

Mock Objects: Example (cont.)

```
public void testXXX() {
    myMockHttpRequest.setupAddParameter("param1", "value1");
    myMockHttpRequest.setupAddParameter("param2", "value2");
    myMockHttpRequest.setExpectedAttribute(
        "some_name_set_in_mymethod", "some value");
    myMockHttpResponse.setExpectedOutput(
        "<html><head/><body>A GET request</body></html>");

    myServlet.init(myMockServletConfig);
    myServlet.doGet(myMockHttpRequest, myMockHttpResponse);

    myMockHttpRequest.verify();
    myMockHttpResponse.verify();
}
```

Readings



1. *Endo-Testing. Unit Testing with Mock Objects*, http://www.mockobjects.com/wiki/MocksObjectsPaper?action=AttachFile&do=get&target=mockobjects.pdf
2. *HttpUnit*, http://httpunit.sf.net/
3. *Jakarta-Cactus*, http://jakarta.apache.org/catus/

Q & A