Tutorial

Comparing JUnit, TCL/TK and TTCN-3

uOttawa

L'Université canadienne
Canada's university

by Bernard Stepien, Liam Peyton,
Pulei Xiong, Pierre Seguin
School of Information Technology and Engineering

Université d'Ottawa | University of Ottawa

www.uOttawa.ca

What does testing consist of?

1. Test specification
   - Specify test data
   - Specify test behavior as sequences of events
   - Specify test outcome (pass/fail)
2. Perform the test
   - Manage communication with SUT
   - Invoke test cases
   - Code or decode messages (telecom)
3. Analyzing test results
   - Details to understand results (expected vs actual values)
   - Tracing of test events
   - Produce reports

2

Purpose of testing tools and frameworks

- Help designing tests
- Reduce the coding effort for test execution
- Reduce the coding effort for test results presentation and analysis

3

Categories of testing tools

- Generic tools and frameworks
- Targeted tools and frameworks (for specific applications)
  - Web testing
  - Specific telecom protocols (SIP, SS7, 3GPP)
- Frameworks that address only part of the testing problem.
  - httpUnit: handles only the communication management and codec of web applications.

4

Testing challenges

- Systems Under Test (SUTs) are composed of a variety of components, each of them can have one or many of the following characteristics:
  - Written in a different implementation language
  - Run on different platforms
  - Run on different locations
  - Use different communication protocols

5

Test implementation choices

- **Choices:**
  - General purpose programming language
  - Specialized programming language
- **Our study:**
  - TCL is a general purpose language
  - TTCN-3 is a specialized language
  - JUnit is a mix of general purpose (Java) and specialized language (Framework classes)

6

Examples of specific testing tools for Web application testing

Other projects that do similar things:
httpunit
webunit, http://sourceforge.net/projects/webunit/
in Python
opensta, http://opensta.org/
Windows only
RoboWeb, http://sourceforge.net/projects/roboweb/
in **perl**, with proxy. no html parsing, just regex asserts
logitest, http://sourceforge.net/projects/logitest
uses java swing browser, **tests are in xml**
Many others are listed at http://www.softwareqatest.com/qatweb1.html#LOAD

7

Tools and Frameworks under study

- **JUnit**
  - A kind of generic testing tool for Java applications only.
- **TCL/TK**
  - An easy language that was not designed originally for testing but is now a de facto standard for testing.
- **TTCN-3**
  - A specialized standard testing language with specialized testing tools

8

Concepts comparison summary

concept	JUnit	TCL/TK	TTCN-3
Test cases definition	Yes	no	yes
typing	Yes (classes)	no	yes
Invoke test cases	Yes (implied)	yes	yes
codec	No, but other frameworks available	Powerful regular expression feature	Yes (API and implementation languages)
Test results details	Yes (some)	no	yes
Display test verdict	yes	no	yes
Tracing of test events	No (only failures)	no	yes

9

A comparison of claims

- Collected from web sites or documentations
- Main arguments:
  - Coding effort
  - Learning effort
  - Cost
  - Integration with other languages
  - purpose

10

JUnit benefits

sources: IBM, Clarkware

- I do not have to write my own framework.
- It is open source, so it is free.
- Other developers in the open-source community use it, so I can find a lot of examples.
- It allows me to separate test code from product code.
- It is easy to integrate into my build process.
- JUnit tests are developer tests.
- JUnit tests are written in Java

11

JUnit is written in Java

http://clarkware.com/articles/JUnitPrimer.html

- The tests become an extension to the overall software and code can be refactored from the tests into the software under test.
- The Java compiler helps the testing process by performing static syntax checking of the unit tests and ensuring that the software interface contracts are being obeyed
- Developer's write and own the JUnit tests

12

TCL benefits

http://www.tcl.tk/about/features.html

- **Rapid development**
- **Graphical user interfaces**
- **Cross-platform applications**
- **Easy to learn**
- **Mature but Evolving**
- **Extend, Embed and Integrate**
- **Deployment**
- **Testing**
- **Network-aware applications**
- **The Tcl community**
- **It's free!**

13

TCL Rapid development

- Tcl gets the job done faster: 5-10x faster especially if the application involves:
  - GUIs
  - string-handling
  - integration.
- Once an application is built in Tcl, it can also be evolved rapidly to meet changing needs

14

Use of TCL in Testing

- **Tcl is an ideal language to use for automated hardware and software testing.**
- **it may well be the dominant language used for this purpose.**
- **Tcl can easily connect to testing hardware or internal APIs of an application, invoke test functions, check the results, and report errors.**
- **Tcl's interpreted implementation allows tests to be created rapidly**
- **tests can be saved as Tcl script files to reuse for regression testing.**
- **Tcl allows you to connect directly to lower-level APIs within the application, which provides much more precise and complete testing.**

15

TTCN-3 benefits

source: www.ttcn-3.org

- Internationally standardized testing language
- Specifically designed for testing and certification
- A testing technology that applies to a variety of application domains and types of testing
- Offers potential for reducing training and test maintenance costs significantly

16

How is TTCN-3 different

source: www.ttcn-3.org

- Rich type system including native list types and support for subtyping
- Embodies powerful built-in matching mechanism
- Snapshot semantics, i.e., well defined handling of port and timeout queues during their access
- Concept of verdicts and a verdict resolution mechanism
- Support for specification of concurrent test behaviour
- Support for timers
- Allows test configuration at run-time
- Tests focus only on implementation to be tested
- Not tied to a particular application or its interface(s)
- Not tied to any specific test execution environment, compiler or operation system
- TTCN-3 as such is not executable and requires a compiler/interpreter, adapter as well as codec implementations

17

Comparing the three languages

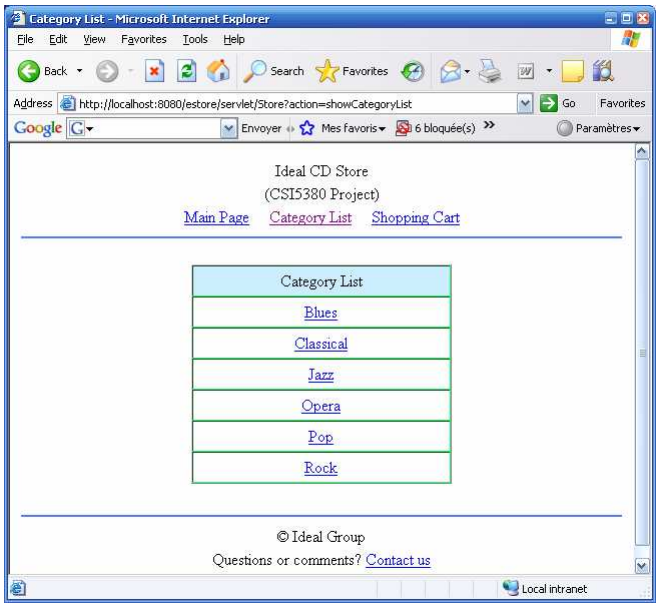
18

Test specification

- A testing example:
  - Testing a simple web page content
- Three implementations of the same case study:
  - JUnit
  - TCL
  - TTCN-3
- The three implementations have intentionally and naively been coded in specific programming styles to illustrate later some important differences among them.

19

Example web page



20

JUnit source of inspiration
single values testing

- JUnit in conjunction with other frameworks such as HtmlUnit is presented in litterature as simple, easy to use and understand

example taken from the HtmlUnit Documentation:

```
public void testHtmlUnitHomePage() throws Exception {
    final WebClient webClient = new WebClient();
    final URL url = new URL("http://htmlunit.sourceforge.net");
    final HtmlPage page = (HtmlPage)webClient.getPage(url);
    assertEquals( "htmlunit - Welcome to HtmlUnit", page.getTitleText() );
}
```

21

Testing a web page JUnit
multiple values testing

```
public void testCategories() {
   List urlList = new LinkedList();
   String[] theLinkNames = {"Main Page", "Category List", "Shopping Cart",
           "Blues", "Classical", "Jazz", "Opera", "Pop", "Rock", "Contact us"};

   final WebClient webClient = new WebClient();
   assertNotNull(webClient);
   try {
      final URL url = new URL("file:categories_list.html");
      final HtmlPage theCurrentPage = (HtmlPage)webClient.getPage(url);

      assertNotNull(theCurrentPage);
      assertTrue(theCurrentPage.getWebResponse().getStatusCode() == 200);
      assertTrue(theCurrentPage.getTitleText().equals("Category List"));

      int textPosition = theCurrentPage.asText().indexOf("Ideal CD Store");
      assertTrue(textPosition >= 0);

      List theLinks = theCurrentPage.getAnchors();

      int n = theLinks.size();

      assertEquals(n, 10);

      for(int i=0; i<n; i++) {
         HtmlAnchor theAnchor = (HtmlAnchor) theLinks.get(i);
         assertEquals(theAnchor.asText(), theLinkNames[i]);
         urlList.add(theAnchor.getHrefAttribute());
      }
   }  catch( Exception e) {...}
}
```

22

Testing a web page in TCL

using regular expression feature

```
package require http 1.0

proc testCategoriesPage {} {
        puts “testing categories page test"

        set categoriesPage [http_get http://localhost:8080/estore/servlet/Store?action=showCategoryList -query]
        set categoriesPageData     [ http_data $categoriesPage ]
        set pageStatus [ http_status $categoriesPage ]

        puts $categoriesPageData

        puts "----------------------------------------------------"

        if { $pageStatus != "ok" } {
                puts "page status not ok - set verdict to fail"
                return
        }

        set textFound [ regexp "<html>.* <title>.*Category List.*</title>.*Ideal CD Store.*CSI5380 Project.*<.*\
href=.*>.*Main Page.*<.*href=.*>.*Category List.*<.*href=.*>.*Shopping Cart.*<.*href=.*>.*Blues.*<.*href=.*>.*Classical.*\
<.*href=.*>.*Jazz.*<.*href=.*>.*Opera.*<.*href=.*>.*Pop.*<.*href=.*>.*Rock.*" $categoriesPageData ]

        if { $textFound == 1 } {
                puts "categories page has matched the expectation - verdict pass"
        } else {
                puts "categories page has NOT matched the expectation - verdict fail"
        }
}

testCategoriesPage
```

23

TTCN-3 approach

- Define data types
- Create templates
- Use transparent and hidden TTCN-3 matching mechanism

24

Testing a web page in TTCN-3

Type declarations:

```
type record linkType {
   charstring text,
   charstring URL
}

type set of linkType linkSet;

type record WebResponseType {
   integer statusCode,
   charstring title,
   charstring content,
   linkSet links
}
```

```
type port web_port_type  message {
                out charstring;
                in WebResponseType
}

type component MTC {
   port web_port_type web_port;
}

type component SystemType {
   port web system_web_port[num_of_ptcs];
}
```

```
template linkSet categoriesPageLinks := {
   {"Main Page", ?}, {"Category List", ?},
   {"Shopping Cart", ?},  {"Contact us", ?}
   {"Blues", ?},  {"Jazz", ?}, {"Classical", ?},
   {"Opera", ?}, {"Pop", ?}, {"Rock", ?}
}
```

```
template WebResponseType categoriesPageResponse := {
   statusCode := 200,
   title := " Category List ",
   content := pattern "*Ideal CD Store*(CSI5380 Project)*",
   links := categoriesPageLinks
}
```

Testing a web page in TTCN-3

test behavior specification

```
testcase categoriesPageTest() runs on MTCType system SystemType {
   timer responseTimer;
   …
   web_port.send("http://137.122.88.254:8080/eStore/categoriesPage");
   responseTimer.start(5.0);
   alt {
         [] web_port.receive(categoriesPageResponse) {
             responseTimer.stop;
             setverdict(pass)
         }
         [] web_port.receive {
             responseTimer.stop;
             setverdict(fail)
         }
        [] responseTimer.timeout {
             setverdict(inconc)
        }
};
```

```
control { execute(categoriesPageTest()) }
```

26

The TTCN-3 adaptation layer

(an excerpt of 200 lines of code)

```
public class WebTesting_TestAdapter extends TestAdapter
                                    implements TriCommunicationSA, TriPlatformPA, TciEncoding {
  …
  public TriStatus triSend(TriComponentId componentId, TriPortId tsiPortId, TriAddress address,
                                                                          TriMessage sendMessage) {

    Byte [] mesg = sendMessage.getEncodedMessage();
    String String theUrlStr = new String(mesg);

     if(tsiPortId.getPortName().equals("systemUserWebPort")) {

       final WebClient webClient = new WebClient();

       try {
            final URL url = new URL(theUrlStr);

            theCurrentPage = (HtmlPage) webClient.getPage(url);

            TriMessageImpl rcvMessage = new TriMessageImpl(theCurrentPage.asText().getBytes());

            myCte.triEnqueueMsg(tsiPortId, new TriAddressImpl( new byte[] {}),  componentId, rcvMessage);
       } catch ( … ) { … }
  }
…
}
```


The TTCN-3 codec

(an excerpt of 300 lines of code)

```
public class WebAndService_Codec extends AbstractBaseCodec implements TciCDProvided {

  public Value decode(TriMessage message, Type type) {
    if(type.getTypeClass() == TciTypeClass.RECORD) {
      String theRecordName = type.getName();

      if(theRecordName.equals("WebResponseType")) {

        try {
             RecordValue cv = Decode_WebResponseType(type.newInstance(), msg, i_con);
            return (Value) cv;
        } catch(Exception iox) {… }
    }
    return null;
  }

 private RecordValue Decode_WebResponseType(Value value2feed, byte [] msg, int i_con) throws Exception {
    Type type = value2feed.getType();
    RecordValue theWebPageValue = null;
    IntegerValue theStatusValue = null;
    CharstringValue theTitleValue = null;
    int theStatus = theAdapterInstance.theCurrentPage.getWebResponse().getStatusCode();
    theStatusValue.setInt(theStatus);

    String theTitle = theAdapterInstance.theCurrentPage.getTitleText();
    theTitleValue.setString(theTitle);
    ….
    return theWebPageValue;
}
```

Web page testing example statistics

- JUnit: **43** lines
- TCL/TK: **30** lines
- TTCN-3:
  - Abstract test suite: 63 lines
  - Adaptation layer: 200 lines
  - Codec: 300 lines
  - Total lines: **563** lines

29

Statistics ?

- Are those statistics fair?
- The answer is no!
- Reasons:
  - Numbers change rapidly when the number of web pages to test increases:
    - JUnit has a fixed coding effort (import statements) and a large proportion of variable coding effort.
    - TCL code is proportional to the number of web pages tested.
    - TTCN-3 has a large fixed coding effort part and limited variable coding effort part
  - The TCL solution has no GUI part

30

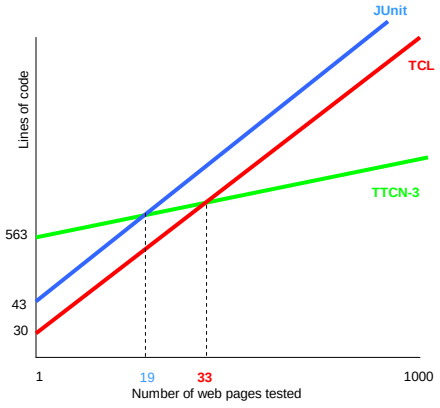
TTCN-3 coding effort structure

- **Fixed coding effort**:
  - Type definitions
  - Behavior definitions (if parametrized)
  - Test adapter
  - Codec
- **Variable coding effort**:
  - Templates definitions
  - Control part

31

TTCN-3 coding effort comparison

Fixed coding effort:	
type definitions:	26 lines
Behavior definitions:	20 lines
module/control	4 lines
Test adapter:	200 lines
Codec:	300 lines
Total fixed part:	**550 lines**
Variable coding effort:	
Templates definitions:	12 lines
Control part:	1 line
Total variable part:	**13 lines**
Total for one page:	**563 lines**



32

Important remark about fixed and variable parts of code

- All three approaches can be decomposed into fixed and variable code parts in a similar way with similar coding effort savings.
- However, the main difference between TTCN-3 and JUnit or TCL is that in TTCN-3 there is a **model** that **forces** the tester to decompose the problem that way.

33

Additional important comparison elements

- The previous example was simple and could be misleading.
- More complex problems reveal more prominent differences:
  - Matching mechanism
  - Structuring concepts
  - Tool results inspection features
  - Complex behaviors specification
  - typing

34

Matching mechanism comparison

35

Matching mechanism JUnit

- Only four categories of matching
  - Expression evaluation to boolean
    - assertTrue(<expression>)
    - assertFalse(<expression>)
  - Checking for nullity
    - assertNull(object)
    - assertNotNull(object)
  - Equality between simple types
    - assertEquals(String, String)
    - …
    - assertEquals(int, int)
  - Equality between objects
    - assertSame(Object, Object)
    - assertNotSame(Object, Object)

36

Matching mechanism TCL

- Regular expressions can become overly complex and unmanageable when there are:
  - Alternate values (“href” | “HREF”)
  - Alternate sequence patterns (see music categories list example. The list of categories could be in different order)

37

Matching mechanism TTCN-3

- As easy as a data assignment to a data structure.
- Is more than a data assignment since for each field various kinds of matching rules can be specified.
- The TTCN-3 template concept allows maximum re-use capabilities, thus a natural structuring mechanism.

38

TTCN-3 matching mechanism advantages

- Transparent matching of complex types
- Transparent matching of lists and sets
- Alternative values matching
- Ranges matching

39

Complex data structures matching

JUnit: 27 lines

```
public void testListEquality(){
   LinkedList list_1 = new LinkedList();
   LinkedList list_2 = new LinkedList();

   list_1.add("john");
   list_1.add("mary");

   list_2.add("john");
   list_2.add("mary");

   int n = list_1.size();

   for(int i=0; i < n; i++) {
      Object obj1 = list_1.get(i);

      if(obj1 instanceof java.lang.String) {
         Object obj2 = list_2.get(i);
         if(obj2  instanceofjava.lang.String) {
            String value_1 = (String) obj1;
            String value_2 = (String) obj2;
            assertEquals(value_1, value_2);
         }
         else fail();
      }
      else
         fail();
   }
}
```

TTCN-3: 13 lines

```
type set of charstring myListType;

template myListType list_1 := {
          "john", "mary"
}
template myListType list_2 := {
          "john", "mary"
}

If(match(list_1, list_2)) {
   setverdict(pass)
}
else { setverdict(fail) }
```

40

Structuring concepts comparisons

- Structuring enables code re-usability
- Structuring improves clarity, thus makes code easier to modify or maintain

41

Structuring potentials

- Factor out re-usable code
- Separate code by functionality

42

JUnit structuring concepts

- Test cases that are Java methods preceded by the word ‘test’.
- Preamble and postamble that are always executed to set a system in a testing state and restore the system to its initial state whether the test passed or failed.
- In JUnit/Java there is no separation between elements of tests and the infrastructure that extracts data elements.
- Factor out re-usable code in plain Java methods

43

JUnit - Using Java methods for structuring

- Usual factoring out of re-usable code into methods
- These methods are now part of the fixed code

```
public void testCategories() {
   List urlList = new LinkedList();
   String[] theLinkNames = {"Main Page", "Category List", "Shopping Cart",
         "Blues", "Classical", "Jazz", "Opera", "Pop", "Rock", "Contact us"};

   final HtmlPage theCurrentPage = GetWebPage(“file:categories_list.html)”;

   assertNotNull(theCurrentPage);
   assertTrue(theCurrentPage.getWebResponse().getStatusCode() == 200);
   assertTrue(theCurrentPage.getTitleText().equals("Category List"));

   verifyLinks(theLinkNames);
}
```

44

Structuring by functionality

- Three functionalities in the following code:
  - Data extraction functionality
  - Data matching functionality
  - Data storing functionality

```
verifyLinks(List theLinkNames) {
    for(int i=0; i<n; i++) {
        HtmlAnchor theAnchor = (HtmlAnchor) theLinks.get(i);   <-- Extraction functionality
        assertEquals(theAnchor.asText(), theLinkNames[i]);     <-- Matching functionality
        urlList.add(theAnchor.getHrefAttribute());             <-- Storing functionality
    }
}
```

45

JUnit

proportion of non-matching functionality statements

```
public void testCategories() {
  List urlList = new LinkedList();
  String[] theLinkNames = {"Main Page", "Category List", "Shopping Cart",
        "Blues", "Classical", "Jazz", "Opera", "Pop", "Rock", "Contact us"};

  final WebClient webClient = new WebClient();
  assertNotNull(webClient);
  try {
    final URL url = new URL("file:categories_list.html");
    final HtmlPage theCurrentPage = (HtmlPage)webClient.getPage(url);

    assertNotNull(theCurrentPage);
    assertTrue(theCurrentPage.getWebResponse().getStatusCode() == 200);
    assertTrue(theCurrentPage.getTitleText().equals("Category List"));

    List theLinks = theCurrentPage.getAnchors();

    int n = theLinks.size();

    assertEquals(n, 10);

    for(int i=0; i<n; i++) {
      HtmlAnchor theAnchor = (HtmlAnchor) theLinks.get(i);
      assertEquals(theAnchor.asText(), theLinkNames[i]);
      urlList.add(theAnchor.getHrefAttribute());
    }
  }
  catch( Exception e) {System.out.println("exception "+e.getMessage());fail(); }
}
```

46

TCL structuring concepts

- Procedures only.
- No concept of test cases.
- Structuring comparable to Java.
- How to structure regular expressions?

47

TTCN-3 structuring concepts

- within the Abstract layer:
  - Test cases
  - Functions
  - Templates
- Separation of concerns:
  - Between abstract and adaptation layer
  - Between behavior and conditions governing behavior in the abstract layer

48

Testcase parametrization

```
testcase webSystemTest(charstring theURL,
                       WebResponseType theResponsePage)
                                runs on MTCType system SystemType {
  timer responseTimer;
  …
  web_port.send(theURL);
  responseTimer.start(10.0);
  alt {
       [] web_port.receive(theResponsePage) {
          responseTimer.stop;
          setverdict(pass)
       }
       [] web_port.receive {
          responseTimer.stop;
          setverdict(fail)
       }
      [] responseTimer.timeout {
          setverdict(inconc)
      }
};
```

Benefits of the TTCN-3 template concept

- TTCN-3 templates are a cross-breed between **data structures** and **functions**, this allows:
  - Static re-use which implies great structuring qualities.
  - Parametrization which allows dynamic re-use of templates.
  - Because TTCN-3 templates are strongly typed, this forces the tester to specify a value or rule.

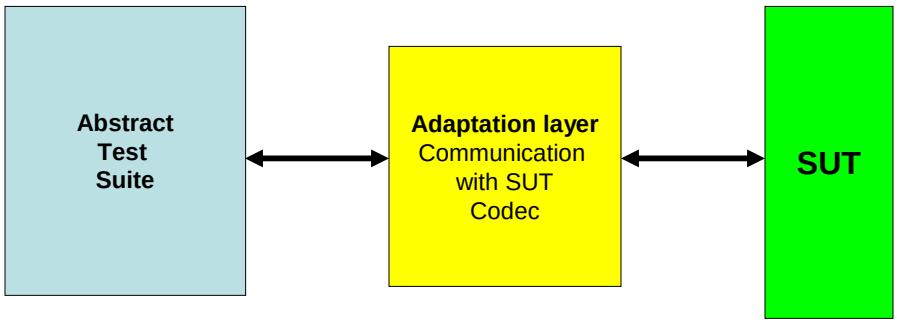
50

Benefits of TTCN-3 separation of concern

- Separation between abstract and concrete layers:
  - Improve clarity of the test behavior
  - Re-usability
  - Re-writability
- Separation between behavior and conditions governing behavior
- Improve clarity of the test behavior
- Provide overview qualities

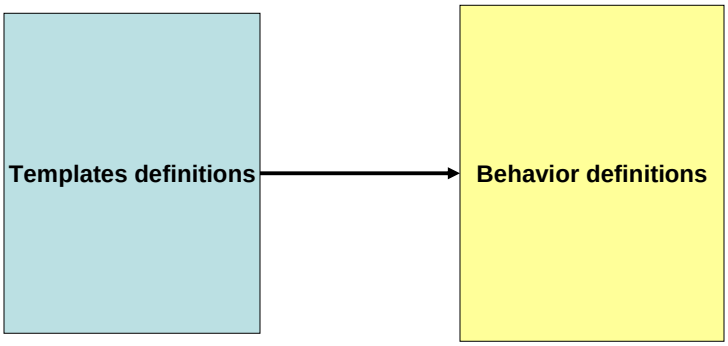
51

Separation of concerns between abstract and concrete layers



52

Separation between behavior and conditions governing behavior



53

Separation of concern

post mortem example

- A company spent two person/years to develop a test suite for a web application using JUnit and httpUnit.
- The test suite was hard to maintain due to the intensive use of httpUnit methods buried deep in the code.
- A number of items could not be tested because httpUnit did not provide appropriate features for that purpose.
- Converting to more appropriate htmlUnit would have required massive changes (80% of the code consisted in invocations to httpUnit methods).
- The test suite was merely scrapped and thus never used.

54

Remarks on separation of concerns in JUnit

- Nothing could prevent a tester to implement the concept of separation of concerns in a general purpose language like Java
- The only difference with JUnit/Java is that with TTCN-3, the tester is forced to do so and thus has no other choice than to be more efficient.
- TTCN-3 inherently provides a model for efficiency.

55

Conclusions on structuring

- In JUnit/Java and TCL, structuring is at the discretion of the test implementer.
- In TTCN-3, structuring is an integral part of the model. There is no way to avoid it.

56

Tool results inspection features comparison

57

JUnit tool features

Failure traces

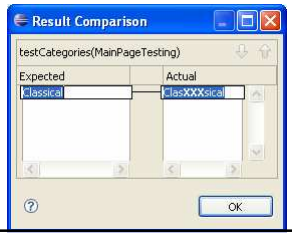
junit.framework.ComparisonFailure: **expected:<...> but was:<...XXX>**
at junit.framework.Assert.assertEquals(Assert.java:81)
at junit.framework.Assert.assertEquals(Assert.java:87)
at MainPageTesting.matchWebPage(**MainPageTesting.java:127**)
at MainPageTesting.testCategoriesPage(**MainPageTesting.java:101**)
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)
at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)
at java.lang.reflect.Method.invoke(Method.java:585)
at junit.framework.TestCase.runTest(TestCase.java:154)
at junit.framework.TestCase.runBare(TestCase.java:127)
at junit.framework.TestResult$1.protect(TestResult.java:106)
at junit.framework.TestResult.runProtected(TestResult.java:124)
at junit.framework.TestResult.run(TestResult.java:109)
at junit.framework.TestCase.run(TestCase.java:118)
at junit.framework.TestSuite.runTest(TestSuite.java:208)
at junit.framework.TestSuite.run(TestSuite.java:203)
at org.eclipse.jdt.internal.junit.runner.junit3.JUnit3TestReference.run(JUnit3TestReference.java:128)
at org.eclipse.jdt.internal.junit.runner.TestExecution.run(TestExecution.java:38)
at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.runTests(RemoteTestRunner.java:460)
at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.runTests(RemoteTestRunner.java:673)
at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.run(RemoteTestRunner.java:386)
at org.eclipse.jdt.internal.junit.runner.RemoteTestRunner.main(RemoteTestRunner.java:196)

58

JUnit mismatch display feature

- JUnit shows only what did not match
- It shows only the first mismatch
- Usable only for assertEquals() assertions
- Does not help for assertTrue() assertions

assertEquals(X, “ClaXXXsical”




```
public void testAssertTrue(){

    int X = 10;
    assertTrue(X == 5);
}
```

In the above, JUnit does not display
The value of the variable X

59

TCL/TK results analysis features

- Basically there are none
- However, because of TK, it is easy to create a custom GUI to display results and improve results analysis
- With TK, GUIs for displaying results can be considered as very flexible. Other tools have only fixed features that a user can not modify.

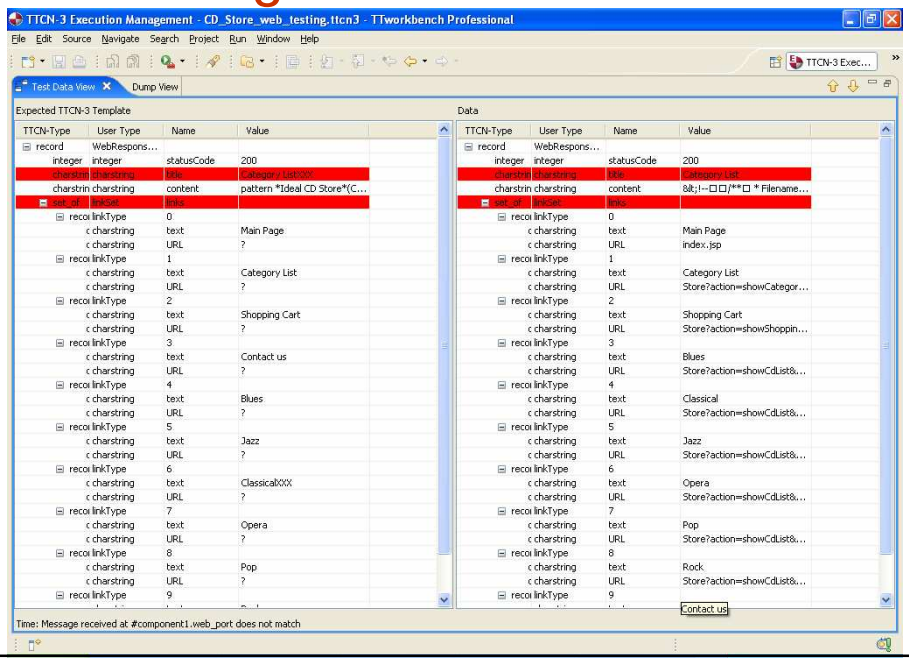
60

TTCN-3 tools features

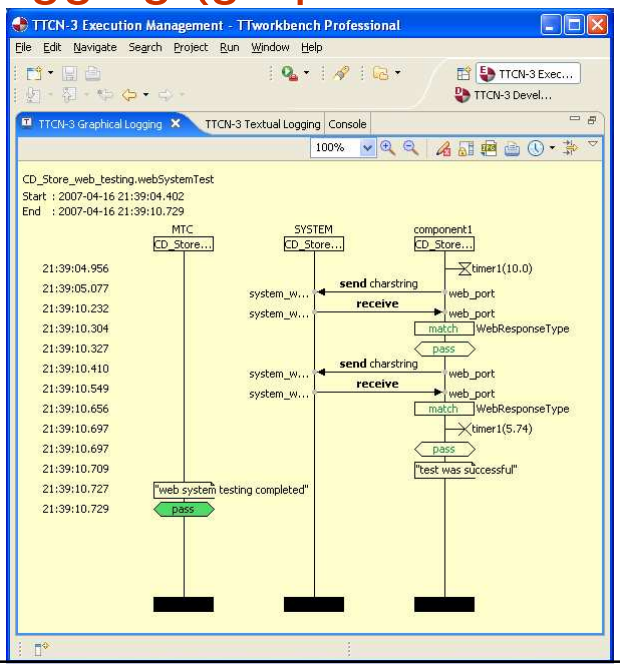
- **Matching mechanism overview**: in case of mismatch, the values of **all** the field that caused the mismatch can be viewed along with the correct values for other fields.
- **Logging**: each event gets logged and thus the sequence of events can be thoroughly inspected. Thus tracing without the need of a classical debugger.
- **Event traceability**: Logs are not limited to display failures, they show successful events too. This improves traceability.

61

Matching mechanism overview



Logging (graphical/textual)

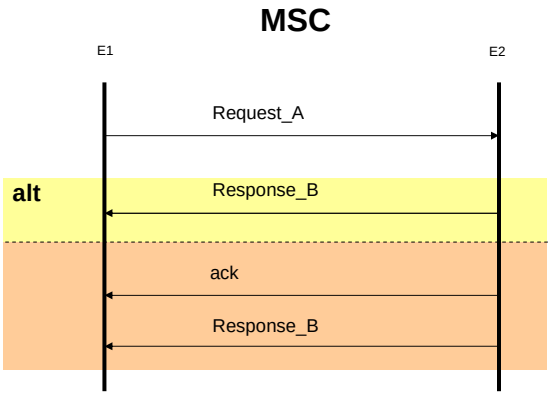


63

Complex behaviors specification comparison

64

Handling alternative behavior



65

Representating alternative behavior in JUnit

- Protocols testing often handles alternate behaviors.
- JUnit can not handle alternatives with the assert… methods.
- The assert needs to be replaced by traditional if-then-else constructs.
- Traditional if-then-else constructs can become rapidly hard to read when the number of elements to compare increases

66

Handling alternatives in JUnit

Wrong specification

// send the request A

1: assertEquals(resp, “response_B”);

2: assertEquals(resp,”ack”);

3: assertEquals(resp,”response_B”);

- If line 1 fails, line 2 and 3 become unreachable

Solution:

```
If(resp == “response_B”) {  } // silent move
else If(resp == “ack”) {
            If(resp == “response_B”) { } // …
           else fail();
}
else fail();
```

67

Handling alternatives in TCL

- Same characteristics as JUnit/Java except potentially with fewer key words.

68

Representing alternate behavior in TTCN-3

- TTCN-3 has a natural alternate construct.
- Each alternative is tried until one matches
- Verdicts are set according to test purposes
- The combination of the TTCN-3 alt construct and template concept naturally eliminates the complexity of the behavior representation when the elements being compared are complex themselves.
- TTCN-3 nesting of alternatives constitute a natural tree representation.

69

Handling alternate behavior in TTCN-3

```
myport.send(request_A);

alt {
    [] myport.receive(response_B) { setverdict(pass)}
    [] myport.receive(ack) {
        alt {
            [] myport.receive(response_B) { setverdict(pass)}
            [] myport.receive { setverdict(fail) }
        }
    }
    [] myport.receive { setverdict(fail) }
}
```

70

Benefits of the TTCN-3 tree concept

- Separation between behavior and conditions governing behavior
- Hides the complex if-then-else constructs.
- Thus, improves clarity of behavior and gives overview qualities.
- Further structuring concept: the altstep

71

Benefits of strong typing ?

72

Typing

- JUnit uses Java classes for typing
- TCL has no typing at all, thus it is very easy to mix variables for different purposes in critical decisions
- TTCN-3 is strongly typed
- TTCN-3 has operational semantics

73

Is Java strongly typed?

- Yes, but …
- However, the combination Java/JUnit is not really typed

74

Typing problems with Java mixing up method calls

```
Public void testCars() {
    …
    matchCars(theExpectedCar, theReceivedCar);
}

public static void matchEngines(Engine engine_a, Engine engine_b) {
    assertEquals(engine_a.typeOfFuel, engine_b.typeOfFuel);
    assertEquals(engine_a.numberOfPistons, engine_b.numberOfPistons);
}

public static void matchCars(Car car_a, Car car_b) {
    assertEquals(car_a.color, car_b.color);
    //matchEngines(car_a.engine, car_b.engine);
    displayEngines(car_a.engine, car_b.engine);
}

public static void displayEngines(Car car_a, Car car_b) {
    System.out.println("engines comparison");
    System.out.println("type of fuel: "+car_a.engine.typeOfFuel+" vs " + car_b.engine.typeOfFuel);
    System.out.println("number of pistons: "+car_a.engine.numberOfPistons+" vs " +
                                                    car_b.engine.numberOfPistons);
}
```

75

Lack of typing with TCL

- Both procedure invocations produce the same result.
- However the first invocation may not be desirable and may lead to wrong conclusions.
- The error is detected only at run time

```
proc myProc { x } {

    if { $x != 5 } { puts “Error x is Not equal to 5; fail" }

}

myProc  "mary had a little lamb"

myProc 1042
```

76

Strong typing benefits with TTCN-3 accidental omissions detection

A field assignment can not be omitted in a template if it is not declared as optional:

```
template Car anotherCar := {
    color := "blue"
}
```

Above, we omitted the specification for the field named “engine”.

Compile time error message:

```
11:53:27:062: [ERROR]: '{ color := "blue" }' of type 'template mapping { charstring:length(4) color }' is not
                                                                of type 'template Car'
11:53:27:062: [ERROR]: (reason) not optional: (field 'engine') '{ color := "blue" }.engine' must not be omitted
11:53:27:078: [ERROR]: compilation finished with errors
```

77

Strong typing benefits with TTCN-3 mixing up functions

You can not invoke a function to specify a template field matching value if the return value type is wrong:

```
template Car aWrongCarTemplate := {
    color := "blue",
    engine := displayMyFavoriteEngine()
}

function displayMyFavoriteEngine() {
   log("I like diesel because it is cheaper");
   log("I like 8 pistons because it looks better");
}
```

Compile time error message:

```
12:04:04:515: [ERROR]: '{ color := "blue", engine := displayMyFavoriteEngine() }' of type 'template mapping
                                { charstring:length(4) color; void engine }' is not of type 'template Car'
12:04:04:515: [ERROR]: (reason) 'displayMyFavoriteEngine()' of type 'void' would have to be of type 'record
                                { charstring typeOfFuel; integer numberOfPistons }'
12:04:04:546: [ERROR]: compilation finished with errors
```

78

Semantics

- There are no semantics either with JUnit/Java nor TCL.
- The TTCN-3 standard specifies both static and operational semantics.
- Any violation of the semantics is detected at compile time.

Example:

You can not use a timer variable in the place of a template in a receive statement.

79

Programming styles

- It all depends how we code it!
- In TTCN-3, there is a clear model that is enforced, thus any deviation from the model is immediately and automatically detected at compile time.
- Consequently, TTCN-3 is safer and thus cheaper.

80

Benefits of standardization

- Everybody uses the same specification language.
- Everybody follows the same model
- Everybody will thus understand what another party within or outside the company means and does.
- Tool vendors will provide utilities such as codecs for well known classes of problems.
- Standardization bodies will publish ready to use test suites (ETSI, …)

81

Contact information

- bernard@site.uottawa.ca
- lpeyton@site.uottawa.ca
- xiong@site.uottawa.ca
- http://www.site.uottawa.ca/~bernard/ttcn.html
- http://www.site.uottawa.ca/~lpeyton/

82