TTCN-3 in end-to-end model based testing explained on a case study

TTCN-3 User Conference 2010
8-10 June, Beijing, China

Presenter: Andrus Lehtmets (andrus.lehtmets@elvior.com)
Elvior

www.elvior.com
About Elvior

Founded in 1992
Location: Tallinn, Estonia

Test tools
► TestCast TTCN-3 test tool
► TestCast Generator
► XML–Simulator

Testing services
► TTCN-3 testing
► Model based testing
► Embedded systems testing
► Building automated test environments
<table>
<thead>
<tr>
<th></th>
<th>Agenda</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Overview of model-based testing (MBT)</td>
</tr>
<tr>
<td>2</td>
<td>Tools used in practical exercise</td>
</tr>
<tr>
<td>3</td>
<td>Example SUT (Light switch)</td>
</tr>
<tr>
<td>4</td>
<td>Workflow of MBT</td>
</tr>
<tr>
<td>5</td>
<td>System Adapter (SA) used in example</td>
</tr>
<tr>
<td>6</td>
<td>Light switch state model</td>
</tr>
<tr>
<td>7</td>
<td>Test cases generation</td>
</tr>
<tr>
<td>8</td>
<td>Execution of generated TTCN-3 test cases</td>
</tr>
<tr>
<td>9</td>
<td>Example of real industrial case study</td>
</tr>
<tr>
<td>10</td>
<td>Questions</td>
</tr>
</tbody>
</table>
Model Based Testing (MBT) – what is it?

► is software testing where
► from a **model** that describes some (usually functional) aspects of the system under test
► + **model coverage criterion**
► **test cases (scripts)** are derived automatically by some tool
TTCN-3 testing

SUT specification

Test goal

Test code/script TTCN-3

System Under Test (SUT)

TTCN-3 Test Tool
Model Based Testing and TTCN-3

- SUT specification
- Test goal
- SUT state model
- Test Generator
- Test code/script TTCN-3
- TTCN-3 Test Tool

System Under Test (SUT)
MBT – when to use?

► it is possible to formalize system behavior **OK** / not **NOK**
► functional testing **OK** / GUI testing **NOK**
► automated testing **OK** / manual testing **NOK**
► it must be possible to **control testing** by test script and SUT behavior must be **observable**

System Under Test (SUT)  
Test tool  
Test script

observable events  
controllable events
Benefits of MBT

► Writing and maintenance of test scripts is a time and effort consuming task.

► **Better tests.** Easier and cheaper to generate sufficient amount of test scripts to achieve a good enough test coverage.

► **Lower costs.** Work effort for test suite maintenance will reduce significantly.
  
  ► Instead of maintaining huge amount of test scripts the test engineer should maintain a SUT model only.
  
  ► If there are changes in the behaviour of the SUT then it is rather easy to update the model correspondingly and re-generate all test scripts once again.
Classical expectations to MBT

1. Through formalization discloses ambiguity in specifications and helps validation of specifications.
2. Better test coverage.
3. Cost effective in maintenance phase.
Test environment in MBT (Elvior approach)
Tools – Poseidon for UML (3rd party tool)

- Used for creating SUT model
- Transition language
- Subset of TTCN-3
Tools – TestCast Generator (Elvior test generator)

► Used for generating tests (TTCN-3 scripts)
► uses SUT model in XMI format (created by Poseidon)
► Runs on Eclipse platform
Tools – **TestCast** (Elvior TTCN-3 test tool)

- Used for executing TTCN-3 tests
- Uses test scripts generated by **TestCast Generator**
- Runs on .NET framework
SUT–LightSwitch—the example SUT

The system under test (SUT) is a lighting system that consists of a switch that turns lights on or off at the user’s request.

![Diagram of LightSwitch system]
SUT–LightSwitch—the example SUT (requirements)

- The light shall be **switched on** by the request from the controlling environment,
- The light shall be **switched off** by the request from the controlling environment.
- If the light is already on/off, requesting the same operation (turning light on/off respectively) shall not change the system state.
- If SUT receives not supported command, then it notifies the controlling environment.
- [https://d-mint.cc.ioc.ee/moodle/](https://d-mint.cc.ioc.ee/moodle/)
### SUT–LightSwitch–the example SUT (use cases)

<table>
<thead>
<tr>
<th>#</th>
<th>Precondition</th>
<th>Input (to the SUT)</th>
<th>Expected result (from the SUT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Light is off</td>
<td>Command turnOn</td>
<td>lightIsOn</td>
</tr>
<tr>
<td>2</td>
<td>Light is on</td>
<td>Command turnOff</td>
<td>lightIsOff</td>
</tr>
<tr>
<td>3</td>
<td>Light is off</td>
<td>Command turnOff</td>
<td>lightIsOff</td>
</tr>
<tr>
<td>4</td>
<td>Light is on</td>
<td>Command turnOn</td>
<td>lightIsOn</td>
</tr>
<tr>
<td>5</td>
<td>Light is on or off</td>
<td>Unknown command</td>
<td>Unrecognised command</td>
</tr>
</tbody>
</table>
SUT–LightSwitch–the example SUT (interface)

1. SUT interacts with outside world using console interface (standard input/output)
2. iLights interface defines commands and SUT responses

<table>
<thead>
<tr>
<th>#</th>
<th>Input (to the SUT)</th>
<th>Output (from the SUT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>string command</td>
<td>string currentLampState</td>
</tr>
</tbody>
</table>

Text constants for input and respective output

<table>
<thead>
<tr>
<th>#</th>
<th>Input (to the SUT)</th>
<th>Output (from the SUT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ready</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>turnOn</td>
<td>lightIsOn</td>
</tr>
<tr>
<td>3</td>
<td>turnOff</td>
<td>lightIsOff</td>
</tr>
<tr>
<td>4</td>
<td>xyz</td>
<td>Unrecognized command</td>
</tr>
<tr>
<td>5</td>
<td>exit</td>
<td></td>
</tr>
</tbody>
</table>
SUT–LightSwitch–the example SUT (test environment)

Adapter between test tool and SUT is needed.
System Adapter used in example (general)

- System Adapter (SA) connects testing tool (TestCast (TC)) with System Under Test (SUT).
- TRI - TTCN-3 standardizes interface between testing tool and SA, this interface is called TTCN-3 Runtime Interface.
- Interface between SA and SUT is always proprietary and therefore needs to be implemented within SA.

- TRI interface is mapped for different languages (C, C++, C#, Java) (Part 5: TTCN-3 Runtime Interface)
- Implementation is tool dependent.
- Most important is what to implement in the methods of the interfaces (i.e. triSend, triEnqueueMsg, triMap)
System Adapter used in example (implementation)

1. Implemented in C#, separate executable
2. SUT specific implementations for ITriCommunicationSA:
   - TriMap, TriUnmap
   - TriSend
   - TriExecuteTestCase, TriEndTestCase
3. SUT specific implementations for ITriCommunicationTE:
   - EnqueueMessage
4. Additional functionality:
   - SUT start, stop
   - Msg traffic logging in SA window
Workflow of MBT (Elvior approach)

- Create SUT model.
- Prepare test data, messages, configuration, functions in TTCN-3.
- Create system adapter according to TTCN-3 TRI.
- Create codecs.
- Generate tests for specified test goal.
- Execute tests.
- Evaluate results and continue with next increment.
State Model of SUT

State model that describes behaviour of the light switch implementation

UnknownCmd/UnrecognizedText; LightIsOff

TurnOn/LightIsOn; fn:silence(1.0)

UnknownCmd/UnrecognizedText; LightIsOn

TurnOn/LightIsOn; fn:silence(1.0)

TurnOff/LightIsOff; fn:silence(1.0)

Exit/Exiting

Initial

T0

/Greetings

T8

LightSwitch_Off

T3

TurnOff/LightIsOff;
fn:silence(1.0)

Exit/Exiting

Final

T6

T1

T2

TurnOff/LightIsOff;
fn:silence(1.0)

T4

T5

T7
## Inputs from external tools (Poseidon case tool)

- **Initial**: /Greetings
- **TurnOff/LightIsOff**: fn:silence(1.0)
- **TurnOn/LightIsOn**: fn:silence(1.0)
- **Exit/Exiting**: Final

<table>
<thead>
<tr>
<th>Transition</th>
<th>Starting state</th>
<th>Trigger (cmd)</th>
<th>Effect (output)</th>
<th>Next state</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>LightSwitch_Off</td>
<td>TurnOn</td>
<td>LightIsOn, Silence</td>
<td>LightSwitch_On</td>
</tr>
<tr>
<td>T2</td>
<td>LightSwitch_On</td>
<td>TurnOff</td>
<td>LightIsOff, silence</td>
<td>LightSwitch_Off</td>
</tr>
<tr>
<td>T3</td>
<td>LightSwitch_Off</td>
<td>TurnOff</td>
<td>LightIsOff, silence</td>
<td>LightSwitch_Off</td>
</tr>
<tr>
<td>T4</td>
<td>LightSwitch_On</td>
<td>TurnOn</td>
<td>LightIsOn, Silence</td>
<td>LightSwitch_On</td>
</tr>
<tr>
<td>T7</td>
<td>LightSwitch_On</td>
<td>UnknownCmd</td>
<td>LightIsOn, Silence</td>
<td>LightSwitch_On</td>
</tr>
<tr>
<td>T8</td>
<td>LightSwitch_Off</td>
<td>UnknownCmd</td>
<td>LightIsOff, Silence</td>
<td>LightSwitch_Off</td>
</tr>
</tbody>
</table>
1. TestData.ttcn describes the possible messages (commands) sent to SUT, such as turnOn and turnOff (for turning the switch on/off), and possible response types from SUT (such as lightIsOn, lightIsOff).

2. TestConfiguration.ttcn describes test component (Tester), its port for message exchange (iLights) and the message types (Command and Output). In addition, it defines the function silence(float duration_sec) for better visualization of the LightSwitch SUT.
Test cases generation

Precondition: Eclipse framework and TestCast Generator are installed

Steps for test scripts generation:

► Creating a new TestCast Generator project
► Handling test generation inputs (from external tools)
► Linking external test inputs to a test generation task resource set
► Defining guidelines for a test generation task
► Generating TTCN-3 test scripts
Test cases generation – TestCast Generator preferences
Execution of generated test cases

Precondition: TestCast TTCN3 tool installed, system adapter exists (TRI), SUT is reachable.
WEB page testing – industrial case study 1

Selenium Core

Remote Control Server

System Adapter

Test purpose

SUT model

TestCast

TTCN-3 test tool

TestCast Generator

www.elvior.com
Feeder Box Control Unit (FBCU). It is a subsystem of the street lighting control system functioning today in Tartu, the second biggest city of Estonia.
Industrial case study 2 – test environment

Poseidon → XMI → TestCast Generator

TestCast TTCN-3 test tool

TestCast-LabView adapter

LabVIEW

Power supply module

Digital/analog module

Hardware adapter

FBCU (SUT, hybrid embedded system)
Industrial case study 2 – SUT state model

Model of FBCU power management (31 states, 73 transitions)
Using MBT in this case study is very efficient, because FBCU behavior is complex and it is easier to change model than rewrite test code – proved in practice.

Numbers (first increment):

<table>
<thead>
<tr>
<th></th>
<th>Time</th>
<th>Code lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>TTCN-3 code (messages, test data, configuration)</td>
<td>~ 15 days</td>
</tr>
<tr>
<td>2</td>
<td>System adapter</td>
<td>150 days</td>
</tr>
<tr>
<td>3</td>
<td>Model building</td>
<td>~ 45 days</td>
</tr>
<tr>
<td>4</td>
<td>Generated tests</td>
<td>NA</td>
</tr>
</tbody>
</table>
Industrial case study 2 – results, increment 2

FBCU changed significantly, new model was built from scratch.

Numbers (second increment):

<table>
<thead>
<tr>
<th></th>
<th>Time</th>
<th>Code lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Model building</td>
<td>~ 10 days</td>
</tr>
<tr>
<td>4</td>
<td>Generated tests</td>
<td>NA</td>
</tr>
</tbody>
</table>

3 fatal bugs found.
Conclusion

- There are common tasks to be solved in both cases (manual and model based TTCN-3 testing).
- Using MBT with TTCN-3 gives extra advantage (TTCN-3 is dedicated for tests, it is natural to generate TTCN-3).
- Building the model formalizes SUT behavior and therefore discloses ambiguity in SUT specifications.
- Model building is resources consuming work, it pays back in maintenance phase – it is easier to alter model and generate tests again.
- MBT advantages are more visible with complex SUT models.
- MBT gives very handy approach for exploratory testing.
Thank you!

Questions?

References:
- testcast.elvior.com
- www.d-mint.org

Supported by: