Using Model-Based Testing during the life cycle of your product

Alexis DESPEYROUX & René-Christian TUYISHIME
Support And Pre-Sales
SUMMARY

- Introduction
- Tutorial
- Conclusion
Test in V cycle
Test challenges

Error correction cost

Nominal cost for errors correction

Where errors are introduced

- IBM Systems Sciences Institute
- Crosstalk, the Journal of Defense Software Engineering
Some Testing issues

- Test plans are written very later
  - Often after the system implementation
  - Errors detected later -> expensive correction

- Maintain test cases and test scripts

- Information on the requirements coverage rate

- Maintain resources

- Lack of communications between designers, developers and testers
  - Lack of processes
  - Projects are abandoned

- Etc.
Model-Driven Engineering

Functional Requirements

Design Model

Functional Requirements

CODE

Test Environment

EXE

Test Cases

Test Script
Design Model

- Embedded system development
- Abstract representation of the system based on specifications
  - Verification
  - Automatic code generation
  - Executable in the target environment
Model-Driven Engineering

Functional Requirements

Usage Model

Test Cases

Test Scripts

Test Environment
Usage Model

- **Behavior of the SUT (System Under Test)**
  - Stimulations or actions
  - Verifications
  - Constraints (time, ...)

- **Formal languages are used**
  - UML, Markov chains, states charts...

- **Test cases generation**
  - Test strategies

- **The usage model construction has to begin as soon as we have a big part of requirements**
  - Detect and remove ambiguities early
Model-Driven Engineering

- Functional Requirements
  - Design Model
    - CODE
      - EXE
  - Usage Model
    - Test Environment
      - Test Cases
        - Test Scripts
Test Strategies

- **Most probable approach**
  - Start
  - Close

- **Risk based Approach**
  - User Oriented - Limit
  - Start
  - Close
  - Custom Test profile

- **Arcs coverage approach**
  - Start
  - Close

- **Usage approach**
  - Random
  - Start
  - Close
  - Usage Test profile

**Test Strategies**

- **FREQUENCY FOCUS**
- **CRITICALITY, COMPLEXITY**
  - UPDATE FOCUS
- **REQUIREMENTS COVERAGE**
- **OPERATIONAL COVERAGE**
Test Cases

- Manual test cases
  - Test suite
  - Test generation report
  - Requirement coverage reports

- Translation in test scripts
  - According to test automation tool

- Executed in a Test environment
SUMMARY

- Introduction
- Tutorial
- Conclusion
Used tools

- MaTeLo (Markov Test Logic)
  - Model-Based Testing approach
  - Markov chains logic
MaTeLo Usage Models

- Setup “Transition” as “Test Step”
- Mapping of Requirements
- Configuration for Test Automation

MACRO CHAIN

- Setup “Transition” as “Test Step”
- Mapping of Requirements
- Configuration for Test Automation

BEGINNING

TRANSITION

STATE

END
Model Transition = Test Step

- Inputs Stimulation
  - Equivalence Classes
  - Timing

- Stimulation
  - Transfer Function
    - python
    - Scilab
    - MATLAB Simulink

- Verifications Points
  - Expected Results
  - Timing

- Transition Label
  - State n
  - State n+1

- Requirements
  - Select_Gear(5)
  - Accelerate(100%)
  - Check_Speed(185)

- Basic Steps (Optional)

- Model Transition:
  - Model Transition = Test Step

- Equivalence Classes
  - Req_1005
  - Switch to automatic mode

- Timing
  - Req_1006
  - Switch to manual mode
# MaTeLo Ecosystem

## Test Cases

<table>
<thead>
<tr>
<th>Native</th>
<th>Plugin</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bench language</td>
<td>Scripting</td>
</tr>
<tr>
<td>TESTSTAND</td>
<td>Python</td>
</tr>
<tr>
<td>EXAM</td>
<td>Selenium IDE</td>
</tr>
</tbody>
</table>
| TTCN-3 | Custom Script | ...
| ... | Open to ALL languages | ...

Open to ALL languages

Bench language

Scripting

Selenium IDE
Used tools

- NI TestStand
  - Test sequencer
    - Execute test sequences generated by MaTeLo
Used tools

- Matlab Simulink for design models
SUT: Cruise Control Simulation

INPUTS
- Control command

OUTPUTS
- Car Model
- Cruise Controller
Used tools

- NI VeriStand

Real time environment
Testing methodology

NI VeriStand

- System Models
  - Simulink®
  - LabVIEW®

- Implement

Functional Requirements

MaTeLo

- Scenarize

Real-Time Testing and Simulation Software

- Generate
  - Test Cases

- Import

- Translate
  - Test Stand Test Script

- Run

- Add

- Test Cases
  - Implement
  - Scenarize
  - Generate
  - Import
  - Translate
  - Run
SUT requirements

Req_01:
- A push on the ON button activates the cruise control and the led is switched on

Req_02:
- A push on the OFF button deactivates the cruise control and the led is switched off

Req_03:
- Pressing the brake pedal deactivates the cruise control and the led is switched off

Req_04:
- When the cruise control is activated:
  - A push on the button SET imposes the current speed as the target speed
  - One push on the button “Inc” increases the cruise control target speed by 1 km
  - One push on the button “Dec” decreases the cruise control target speed by 1 km
  - The increase or decrease of 1 km must last at maximum 50 ms

Req_05:
- The cruise control is effective between [30, 150] Km/h
SUMMARY

Live DEMO
SUMMARY

- Introduction
- Tutorial
- Conclusion
Conclusion

- **Quantified and optimized requirements coverage**
  - Model-Based Testing tools give requirements coverage indicators

- **Consolidation of functional requirements**
  - Ambiguities in specifications are removed early

- **Pertinent test cases (usage profiles, risks, ...)**
  - Possibility to define usage profiles
  - Risks are taken into account in the generation strategies

- **Easy test cases maintenance**
  - It is more easier to maintain an usage model than manual test cases

- **Easy test cases automation**
  - Test effort lowered