INTEGRATED FRAMEWORK FOR DEVELOPMENT AND TESTING OF SAFETY-CRITICAL AUTOMOTIVE MECHATRONIC SYSTEMS

Presented by Nick VAN KELECOM
Industry challenges

A modern car realizes 100’s of vehicle functions...

...with increased algorithm complexity...

...Programmed in million lines of software code.

Each function needs to be validated and calibrated...

...at reduced development time and cost...

...and needs to be certified before going in production!
Extensive testing is necessary to solve these challenges...

... But current testing methods seem incapable.
Industry Testing Flaws

Disconnect between requirements and test cases

Requirements

Derive

Test Cases

Initiate

Test Environment Model for:
Unit Testing
Integration Testing

Validate

Loading incorrect version

Trigger incorrect tests / Skip important tests

Disconnect between results and test cases/requirements
Problem Identification

- Requirement – Test Case Traceability
- Test Trigger Management
- Test Environment Management
- Test Result Management
Central requirement and test case management
Enable tracing through unique identifiers

Ensure bidirectional traceability
Model-based architecture representation

Track development status for test trigger

Model Identity Card containing:
- Model status
- Model progress
- Requirements
- Test cases
- Description
- Test results

Filter models on status for correct testing phase
Model status triggers correct test phase
Support for continuous integration

User initiates status change
User checks results and takes appropriate next action

Central management software running on remote server (with SVN capabilities)
Commit results back
Dedicated testing server using Jenkins for communication (with SVN capabilities)

Extract model properties and pass to CI server
Automatically build correct test environment
Improve test case reusability

Requirement – Test Case Traceability
Test Trigger Management
Test Environment Management
Test Result Management

Test cases written textually
Activate
Dedicated testing server using Jenkins for communication (with SVN capabilities)
Automatically build correct test environment
Improve test case reusability

Requirement – Test Case Traceability
Test Trigger Management
Test Environment Management
Test Result Management

Test cases written textually
Activate
Dedicated testing server using Jenkins for communication (with SVN capabilities)
Create empty data sheet customized to MUT
Manually translate into actual test data
Generic platform for optimal test data reuse
Automatically build correct test environment

Improve test case reusability

Dependent on XiL testing platform:

Automatic test environment created customized to MUT:

- Functional unit testing with
  - Reference output signals (MiL)
  - Back-to-back test (SiL)
  - Performance integration testing with high-fidelity plant models
→ Proven re-usability of test cases

Hardware – Software integration testing with high-fidelity plant models running in real-time
Example:
Testing Black Box for MiL integration test

MUT
Plant Model

Extract and compare results

Trunk (L, L)
Rain (L, L)
Enduser (L, L)
Package (L, L)
Error (L, L)
CC_Action (L, L)

Test Environment Management
Automatically build correct test environment
Improve test case reusability

Test cases written textually

Activate

Dedicated testing server using Jenkins for communication (with SVN capabilities)

Create empty data sheet

Push test data

Run tests and bring results back

Testing Black Box

Create custom model-specific testing environment

Manually translate into actual test data
### General Overview

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Test Case Traceability</th>
<th>Test Trigger Management</th>
<th>Test Environment Management</th>
<th>Test Result Management</th>
</tr>
</thead>
</table>

- **Easy overview of nightly test run results**
- **Initiator for team planning**

### Model-Specific Overview

**General**
- Linked Model: EN 2652
- Compile Check: SUCCESS
- Compile Error Message: Model Block Count: 24
- Cyclomatic Complexity: 6

**Model Analysis results**
- General Passed: 60
- General Warning: 7
- General Failed: 8
- ISO 26262 Passed: 6
- ISO 26262 Warning: 2
- ISO 26262 Failed: 8

**Performance tests**
- Functionality Error Pass: 23
- Functionality Error Fail: 1

**Code Coverage results**
- MCCD Coverage (%): N/A
- Decision Coverage (%): 90
- Condition Coverage (%): N/A
- Execution Coverage (%): 100
More effort spent upstream to reduce later re-work

- Requirement Analysis
- Validation & Verification
- Vehicle Integration & Testing
- System Testing
- HiL
- System Integration (Testing)
- Unit Testing
- MIL
- Code Generation
- SIL
- Test Trigger Management
- Requirement/ Test Case Management
- Test Environment Management
- Test Result Management
Future Work

- Extend framework to HiL testing
- Improve architecture capabilities
  - Architecture design
  - Architecture testing
- Improve functional safety capabilities
- ...
Thank You

Nick Van Kelecom
Research Engineer MBSE

Embedded Systems & Controls
Simulation & Testing Solutions
Siemens Industry Software NV

nick.van_kelecom@siemens.com