Testing, Validating, and Verifying with Model-Based Design

Phil Rottier

MATLAB EXPO 2015
UNITED KINGDOM
Summary

- MATLAB®, Simulink® and Stateflow® help individuals and teams rapidly develop complex designs
- These designs should be supported by appropriate levels of testing
  - Demonstrate compliance with requirements
  - Identify defects as early as possible
- Efficient testing is as important in speeding up design times as having capable design tools. This talk
  - describes the range of test, verification and validation techniques that can be applied to models
  - discusses the classes of defects they can best discover
Definitions

- Verification: have we done what we said we would do?
  - the process of determining that a computer model, simulation, or federation of models and simulations and their associated data accurately represent the developer's conceptual description and specifications

- Validation: is what we said we would do the right thing to do?
  - the process of determining the degree to which a model, simulation, or federation of models and simulations, and their associated data are accurate representations of the real world from the perspective of the intended use(s)

- Testing: activities that support V&V
  - System / unit tests
  - Closed / open loop
Testing, Validating, and Verifying with Model-Based Design

- Applicable at any point in the design process
Testing, Validating, and Verifying with Model-Based Design
Typical questions / issues

- Does the model meet its requirements?
  - Including requirements for non-standard / unexpected or extreme operating conditions
  - Not all requirements relate to model execution – may have “static” requirements too
  - Is the model fit for purpose?
  - Does the model contain errors?

- The model (or data on which it depends) has changed
  - Does it still work?
  - Does it still meet its requirements?

- Re-use of existing modelling components / library blocks
  - Does it interface correctly into the new application?
  - Does it still meet its requirements?

- Generating code
  - Does the code match the model?
Testing, Validating, and Verifying with Model-Based Design

- Core Simulink
  - Signal Builder block
  - Simulink Data Inspector
  - Simulink Model Verification
  - Model Advisor
  - Simulink Project

- Additional products
  - Simulink VnV
  - Simulink Design Verifier
  - Simulink Test

- Also
  - MATLAB unit test framework
  - Parallel computing (for test case sweeps, batch)
Example model

- Based on “sf_car” demo shipping with Stateflow®
Signal Builder block

- Interactive scenario definition
- Integration with SL VnV
  - Requirements link
  - Batch facility
  - Coverage measurement
Simulink® Simulation Data Inspector
Simulink® Model Verification

- Library of blocks for “on the fly” in-model verification
Demo 1: sfCar

- Requirements
- Signal builder
- Model workspace
- Data inspector
- Verification
- Compliance
  - Manually scripted, the hard way
Model Advisor

- Applies predefined checks to model
  - Confirms adherence to modelling standards
  - Can find some types of modelling errors
- Creates HTML report summarising check results
Model Advisor

- Libraries of additional checks associated with additional products
- With Simulink Verification and Validation
  - Write your own (sl_customization.m)
Demo 2: sfCar

- Simple example
  - Disconnected / missing connections
- HTML report generation
  - For full set of Simulink checks
Simulink® Project – Project management and automation tools

- Project file structure
- Artefacts / labels
- Source control integration
- Dependency analysis
- Impact analysis

- Helps you to
  - Manage your project
  - Identify impacts
  - Work out which tests need to be run
MATLAB® Unit Testing Framework

MATLAB® Unit Testing Framework

MATLAB EXPO 2015
UNITED KINGDOM

17
Demo 3: slProject

- Structure of a Simulink Project
- SVN integration
- Unit test framework
- Batch job feature
- Impact analysis
Simulink VnV

- Model Guidelines Compliance
  - Check that models comply with standards and modelling guidelines

- Requirements Traceability
  - Link between model objects and associated requirements defined in external documents

- Model Coverage Analysis
  - Measure model coverage in simulation
  - Programmatic interface to coverage analysis

- Model-Based Testing
  - Configure Model Verification blocks
  - Manage component test data, execute test suite in SIL or PIL simulation

- Tool Qualification and Certification
  - Qualify Simulink Verification and Validation for DO and IEC Certification
Demo 4: sfCar with VnV

- Requirements link
- Signal builder integration
  - Requirements link
  - Verification blocks
- Coverage analysis
  - Interactive
  - Via signal builder
  - Programmatic
- Scripted test harness generation and execution
- Verification of sim/sil equivalence
- Additional Advisor functionality
Simulink VnV: Types of coverage

- **Condition coverage**
  - Analyzes blocks that output the logical combination of their inputs and Stateflow transitions.
  - Full coverage causes each input to each instance of a logic block in the model and each condition on a transition to be true at least once during the simulation, and false at least once during the simulation.

- **Decision coverage**
  - Analyzes elements that represent decision points in a model, such as a Switch block or Stateflow states.
  - For each item, decision coverage determines the percentage of the total number of simulation paths through the item that the simulation actually traversed.

- **Modified condition/decision coverage**
  - Analyzes blocks that output the logical combination of their inputs and Stateflow transitions to determine the extent to which the test case tests the independence of logical block inputs and transition conditions.
Simulink Design Verifier

- Systematic Model Verification
  - Identify and configure model components for analysis

- Design Error Detection
  - Statically detect run-time errors and dead logic, derive design ranges

- Test Case Generation
  - Generate systematic test cases from model, extend and combine test cases for full test suite

- Requirements-Based Verification
  - Verify design against requirements, specify analysis input constraints

- Complexity Management
  - Handle incompatibilities, optimize analysis for large and complex models

- Results Interpretation and Use
  - Log and review analysis results, generate report, create test harness model

- Model Simplification with Dependency Analysis
  - Trace dependencies of ports, signals, and blocks, slice larger models into simplified standalone models

- Tool Qualification and Certification
  - Qualify Simulink Design Verifier for IEC Certification
Simulink Test
Tool for authoring, managing, and executing simulation-based tests

3 Major Components:

1. Test Harnesses
   - Synchronized testing environment
   - Enables unit testing without requiring new model
   - Component or system
   - Configure, build, simulate
   - Supports MIL, SIL, or PIL
   - Supports direct “adhoc” testing

Test Harness
Main Model
Exercise
CUT
Analyze

Simulink Test
*Tool for authoring, managing, and executing simulation-based tests*

3 Major Components:

2. Test Sequence Block
   - MATLAB Action Language
   - Steps are temporal or logic-based
   - Create complex test inputs and assessments
   - Trouble shooting aids
Simulink Test
Tool for authoring, managing, and executing simulation-based tests

3 Major Components:

3. Test Manager

- Create Test Cases
- Group into Suites and Test Files
- Execute individual or batch
- View summary and detailed results
- Archive, export, report
Demo 7: sfCar with Simulink Test

- System test
  - baseline
- Unit test
- Multiple unit tests
Simulink Test

- Test Case Templates
- Simulation Testing
- Baseline Testing
- Equivalence Testing
- Programmatic API
Conclusion

- Does the model meet its requirements?
  - Including requirements for non-standard / unexpected or extreme operating conditions
  - Not all requirements relate to model execution – may have “static” requirements too
  - Is the model fit for purpose?
  - Does the model contain errors?

- The model (or data on which it depends) has changed
  - Does it still work?
  - Does it still meet its requirements?

- Re-use of existing modelling components / library blocks
  - Does it interface correctly into the new application?
  - Does it still meet its requirements?

- Generating code
  - Does the code match the model?
Conclusion

- Core Simulink
  - Signal Builder block
  - Simulink Data Inspector
  - Simulink Model Verification
  - Model Advisor
  - Simulink Project

- Additional products
  - Simulink VnV
  - Simulink Design Verifier
  - Simulink Test

- Also
  - MATLAB unit test framework
  - Parallel computing (for test case sweeps, batch)
Conclusion

- **Core Simulink**
  - Signal Builder block
  - Simulink Data Inspector
  - Simulink Model Verification
  - Model Advisor
  - Simulink Project

- **Additional products**
  - Simulink VnV
  - Simulink Design Verifier
  - Simulink Test
  - **Simulink Code Inspector**
  - Polyspace bug finder / code prover

- **Also**
  - MATLAB unit test framework
  - Parallel computing (for test case sweeps, batch)
Questions?