Increasing Design Confidence: Model and Code Verification

Chuck Olosky
Application Engineer

Nishaat Vasi
Product Manager
Finding errors late is expensive and risky

Examples

- 1996: Ariane 5 rocket destroyed, software defect due to faster horizontal drifting speed of new rocket – $850 million lost

- 1999: Loss of Titan IV, incorrect software with wrong roll rate filter constant - $1.23 billion lost

- 2013: Recall of 344,000 minivans, software defect may cause application of brake without driver action

- 2013: Recall 7,100 cars, software fault results in automatic transmission to shift out of park without brake press
Application: Cruise Control

ECU system

Cruise Control Module (MBD)

Fuel Rate Control Module

Shift Logic Control Module

Outputs

System Inputs

Legacy code
Gaining Confidence in our Design

- Ad-hoc tests
- Early prototyping
- Design error detection
- Functional & structural tests
- Code integration checks
- Field tests
- Identify requirement gaps, assess model coverage, model-code equivalence
- Dead logic due to float-to-fixed model conversion
- Changing analog-to-digital converter from 14 to 12-bit results in dead code
- While going downhill, target speed increases with “reduce speed” button

Confidence vs. Effort / Time
Demo: Cruise Control Overview
Simulink, Simulink Design Verifier
Gaining Confidence in our Design

Confidence

Ad-hoc tests

Design error detection

Functional & structural tests

Code integration checks

Field tests

Effort / Time
Finding Unintended Behavior

Converting floating-point model to integer calibrations, signals…

- Dead logic due to “uint8” operation
Finding Unintended Behavior

- **Dead logic** due to “uint8” operation on `incdec/holdrate*10`
  
  ```
  debug>> incdec
  incdec = 1
  debug>> class(incdec)
  ans = double
  debug>> holdrate
  holdrate = 5
  debug>> class(holdrate)
  ans = uint8
  debug>> incdec/holdrate
  ans = 0
  ```

- **Fix** change the order of operation `10*incdec/holdrate`
  
  ```
  debug>> class(incdec)
  ans = double
  ```

Condition can never be false
Gaining Confidence in our Design

- Ad-hoc tests
- Design error detection
- Functional & structural tests
- Code integration checks
- Field tests

Confidence vs. Effort / Time
Simulation Testing Workflow

Did we meet requirements?

Review functional behavior

Did we completely test our model?

Structural coverage report

Requirements

Design

Functional

Structural
Did We Completely Test our Model?

Potential causes of less than 100% coverage:
- Missing requirements
- Over-specified design
- Design errors
- Missing tests
Demo: Simulation Based Testing

Simulink Test, Simulink Verification and Validation
Gaining Confidence in our Design

Confidence vs. Effort / Time

- Ad-hoc tests
- Design error detection
- Functional & structural tests
- Code integration checks
- Field tests
ECU System Architecture

Inputs
- Cruise_onoff
- Brake
- Speed
- Coast set
- Accel reset
- EGO Sensor
- MAP Sensor

ECU system

Cruise Control Module (MBD)

Fuel Rate Control Module

Shift Logic Control Module

Outputs
- Gear
- Engaged
- Target speed
- Fuel Rate

Legacy code
Demo: Code Integration Errors

Polyspace Code Prover
Code Integration Issues

Target speed parameter propagated to “Cruise_ctrl.c” [0 … 40]

Maximum target speed = 90

Dead code
Search for Parameter in Upstream Source

/* Transition: '<S5>:114' */
/* Exit Internal 'ON': '<S5>:154' */
localIDW->is_ON = IN_NO_ACTIVE_CHILD;
localIDW->is_CRUISE = IN_STANDBY;

/* Entry 'STANDBY': '<S5>:52' */
*rtty Engaged = false;
}
else if (rtu_Speed > maxspeed)
{ ...
Use Call Graphs for Multi-File Traceability
Root Cause for Dead Code

Changing analog-to-digital converter from 14 to 12-bit results in dead code

MASK – accounts for scaling down for new ADC from 14-bit to 12-bit

CONV_FACTOR – accounts for translating sensor input to miles/hrt

Overlooked changing CONV_FACTOR for new ADC
Find Dead Code During Integration

**Inputs**
- Cruise_onoff
- Brake
- Speed
- Coast set
- Accel reset
- EGO Sensor
- MAP Sensor

**Outputs**
- Gear
- Engaged
- Target speed
- Fuel Rate

**ECU system**
- Cruise Control Module (MBD)
- Fuel Rate Control Module
- Shift Logic Control Module

Dead code
Legacy code

Inaccurate scaling for speed
Gaining Confidence in our Design

- Ad-hoc tests
- Design error detection
- Functional & structural tests
- Code integration checks
- Field tests
Field Calibration Tests Uncover Error

- **Problem:** While going downhill, target speed increases with “reduce speed” button and assumes random values
  - Functional tests pass for model
  - No redundancies in model (100% coverage achieved)
  - Nominal signal and parameter values worked in simulation

- **Debug Options:**
  1. Create test to reach this Cal condition
  2. Use static analysis tools to identify/ prove correctness
Using Model-Based Design to Reproduce Field Issue

- Construct a model of field issue
- Constrain inputs to represent field issue
- Create model of field issue behavior
- Ask tool to prove whether errant condition can occur
Field Issue Behavior Model
Target speed increases with “reduce speed” button

a) I set the target speed to the vehicle speed (40 mph) while going downhill on the track

b) I was pulsing the “reduce speed” button until it decreased the target speed to the 20 mph limit

c) The next time I hit the “reduce speed” button it increased the target speed from 20 mph to 33 mph
Demo: Reproducing Field Issues

Simulink Design Verifier
Gaining Confidence in our Design

Confidence

Ad-hoc tests

Design error detection ✔

Functional & structural tests ✔

Requirement proofs ✔

Code integration checks ✔

Field tests ✔

Effort / Time
Industry Examples of Finding Errors Early

- **Lear Body Control Electronics**
  - Found >95% of requirements issues before implementation (compared to 30% prior)

- **DLR Autonomous Humanoid Robot**
  - Functional defects reduced by 80%

- **Weichai Common-Rail Diesel Engine**
  - Detect 60% - 70% bugs before integration

- **Airnamics Unmanned Aerial System**
  - Found 95% control software bugs before first flight
# MathWorks V&V Product Portfolio

<table>
<thead>
<tr>
<th>Product</th>
<th>Capabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulink Test</td>
<td>Author, execute, and manage simulation-based tests for models and generated code</td>
</tr>
<tr>
<td>Simulink Verification &amp; Validation</td>
<td>Trace to requirements, check model standards, perform coverage analysis</td>
</tr>
<tr>
<td>Simulink Design Verifier</td>
<td>Identify design errors, automatically generate test vectors, verify designs against requirements</td>
</tr>
<tr>
<td>Polyspace Bug Finder</td>
<td>Find software bugs and check compliance to MISRA</td>
</tr>
<tr>
<td>Polyspace Code Prover</td>
<td>Prove the absence of run-time errors in software</td>
</tr>
</tbody>
</table>
Thank You