Increasing Design Confidence with Model and Code Verification

The Cost of Failure...

Ariane 5

$7,500,000,000

Rocket & payload lost
The Cost of Failure...

USS Yorktown

0 Knots
Top speed

The Cost of Failure...

Therac-25

6 Casualties
due to radiation overdose
Motivation

It is easier and less expensive to fix design errors early in the process when they happen.

Model-Based Design enables:

1. Early testing to increase confidence in your design
2. Delivery of higher quality software throughout the workflow

MATLAB EXPO 2016

Gaining Confidence in our Design

MATLAB EXPO 2016
Application: Cruise Control
Control speed according to setpoint

50 km/h

System
Inputs

ECU

System Inputs

ECU system

1

Cruise Control Module (MBD)

Fuel Rate Control Module

Shift Logic Control Module

2

Outputs

Legacy code

MATLAB EXPO 2016
Application: Cruise Control

System Inputs

ECU system

Legacy code

Outputs

Cruise Control Module (MBD)

Fuel Rate Control Module

Shift Logic Control Module

Inputs

Cruise_onoff
Brake
Speed
Coast set
Accel reset

Outputs

Engaged
Target speed
Gaining Confidence in our Design

Ad-hoc testing

Effort / Time

Confidence

MATLAB EXPO 2016

Ad-hoc Tests

New “Dashboard” blocks facilitate early ad-hoc testing
Gaining Confidence in our Design

Confidence

Ad-hoc testing

Effort / Time

MATLAB EXPO 2016

Finding Design Errors: Dead Logic

Result:
- Design error detection completed normally.
- 2/27 objectives are dead logic.

Results:
- Generate detailed analysis report
- Open harness mode

Controlled parameters:
- [after (incdec/holdrate... *10, tick)]

MATLAB EXPO 2016
Gaining Confidence in our Design

Ad-hoc testing

Design error detection

Confidence

Effort / Time

Simulation Testing Workflow

Requirements

Design

Did we meet requirements?

Review functional behavior

Did we completely test our model?

Structural coverage report

MATLAB EXPO 2016
Did We Completely Test our Model?

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

Requirements Based Functional Testing with Coverage Analysis

- All 14 requirements based test cases pass
- By analyzing model coverage results we find:
  - Missing test cases for vehicle speed exit conditions, and
  - Missing requirements (and test cases) for “hold” or continuous speed button input
Functional Testing with Added Requirements & Test Cases

- Added 2 new requirements for the “hold” case for speed setting input buttons
- Added 5 test cases to the original 14 requirements based test cases
  - 3 test cases for the 2 new requirements
  - 2 test cases for the missing test cases for the vehicle speed exist conditions
- 4/5 new functional test cases pass
  - Failed test case showed overshoot beyond target speed limits
  - Coverage analysis highlighted transitions with design errors
  - Fixed comparison operators, (<) \rightarrow (<=), and (>) \rightarrow (>=)
- Now all (19) functional test cases pass with 100% model coverage!
Gaining Confidence in our Design

Confidence

Effort / Time

Gaining Confidence

- Ad-hoc testing
- Design error detection
- Functional & structural tests

MATLAB EXPO 2016

Model Advisor – Model Standards Checking

Model Advisor Report for 'Step_02_JumpJump'

See Also

- MathWorks Automotive Advisor Board Guidelines for 2007

MATLAB EXPO 2016
Gaining Confidence in our Design

- Ad-hoc testing
- Design error detection
- Functional & structural tests
- Modeling standards
- Effort / Time
- Confidence

Equivalence Testing:
Model vs SIL or PIL Mode Testing

- Model Testing
- SIL or PIL Mode Testing
- Coverage → 100%
- Model used for production code generation
- Embedded Coder
- Generated C code
- Target compiler and linker
- Object code
- Execution
- Simulation
- Result vectors (base line) \( \alpha_{\text{ref}}(t) \)
- Signal comparison
- Result vectors \( \alpha_{\text{test}}(t) \)

MATLAB EXPO 2016
Code Generation with Model-to-Code Traceability

Rich_Mixture
entry: fuel_mode = RICH;

Single_Failure

MATLAB EXPO 20

Code Equivalence Check Results: Model vs Code

Code Coverage

<table>
<thead>
<tr>
<th>File</th>
<th>Contents/Complexity</th>
<th>D1</th>
<th>C1</th>
<th>MCDC</th>
<th>Start</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CruiseControl.c</td>
<td>22.97%</td>
<td>98%</td>
<td>90%</td>
<td>100%</td>
</tr>
<tr>
<td>2</td>
<td>CruiseControl_Init</td>
<td>1</td>
<td>--</td>
<td>--</td>
<td>100%</td>
</tr>
<tr>
<td>3</td>
<td>CruiseControl</td>
<td>20.97%</td>
<td>98%</td>
<td>90%</td>
<td>100%</td>
</tr>
<tr>
<td>4</td>
<td>CruiseControl_update</td>
<td>1</td>
<td>--</td>
<td>--</td>
<td>100%</td>
</tr>
</tbody>
</table>

47 /* Output and update for referenced model: 'CruiseControl' */
48 #if cruiseControlChangeEvent_t.rto_CruiseOnOff, cruiseControl_T.rto_Brake,
49   cruiseControl_T.rto_Speech, cruiseControl_T.rto_CruiseSets,
50 (cruiseControl_T.rto_Accelerate, cruiseControl_T.rto_ThetaEngaged,
51   cruiseControl_T.rto_ThetaEngaged)
52 { ...
53 /* Chart: 'rto/Compute_target speed' */
54 /* Gateway: compute target speed */
55 if (CruiseControl_DW.temporalCounter_s1 < MAX_uint32_t) {
56   CruiseControl_DW.temporalCounter_s1 += 1;
57 }

49 CruiseControl_DW.temporalCounter_s1 = 0;
50 CruiseControl_DW.temporalCounter_s1 = 0;
51

Decisions analyzed:

<table>
<thead>
<tr>
<th>Decision</th>
<th>#1</th>
<th>#2</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>0</td>
<td>0</td>
<td>0.420</td>
</tr>
<tr>
<td>True</td>
<td>3</td>
<td>1</td>
<td>4.200</td>
</tr>
</tbody>
</table>

26
Code Equivalence Check Results: Model vs Code

- Re-used full coverage test vectors and harnesses from Model Verification testing
- Ran test vectors on generated code using Model Reference SIL mode
- Equivalence test performed in Simulink Test, including test execution, evaluation and presentation of the results
- Compared Model Coverage to Code Coverage using the SIL Code Coverage Report
- Successfully demonstrated code behavior matches model behavior!

Gaining Confidence in our Design

- Ad-hoc testing
- Design error detection
- Functional & structural tests
- Modeling standards
- Model & code equivalence checks
- Code integration analysis
**Code Integration Analysis**

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

**ECU system**

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

**ECU**

1. Cruise Control Module (MBD)
2. Fuel Rate Control Module
3. Shift Logic Control Module

System Inputs → Outputs

Legacy code
Finding 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
- Cruise Control Module (MBD)
- Fuel Rate Control Module
- Shift Logic Control Module

Legacy code
- Inaccurate scaling for speed

Inputs to ECU
- Cruise onoff
- Brake
- Speed
- Coast set
- Accel reset
- EGO Sensor
- MAP Sensor

Outputs from ECU
- Gear
- Engaged
- Target speed
- Fuel Rate

Finding Dead Code with Polyspace

Target speed parameter propagated to "Cruise_ctrl.c"
[0 ... 40]

Maximum target speed = 90

Dead code

MATLAB EXPO 2016
Polyspace Code Analysis

Start with C/C++ source code

```
static void pointer_arithmetic (void) {
    int array[100];
    int *p = array;
    int i;
    for (i = 0; i < 100; i++) {
        *p = 0;
        p++;
    }
    if (get_bus_status() > 0) {
        if (get_oil_pressure() > 0) {
            *p = 5;
        } else {
            i++;
        }
    }

    i = get_bus_status();
    if (i >= 0) {
        *(p - i) = 10;
    }
}
```

Polyspace Code Analysis

Source code painted in **green**, **red**, **gray**, **orange**

- **Green**: reliable safe pointer access
- **Red**: faulty out of bounds error
- **Gray**: dead unreachable code
- **Orange**: unproven may be unsafe for some conditions
- **Purple**: violation MISRA-C/C++ or JSF++ code rules

**Range data tool tip**
Gaining Confidence in our Design

Confidence

Effort / Time

Ad-hoc testing
Design error detection
Functional & structural tests
Modeling standards
Model & code equivalence checks
Code integration analysis

MATLAB EXPO 2016

Conclusion: Model-Based Design Verification Workflow

Model Verification
Discover design errors at design time
Module and integration testing at the model level
Review and static analysis at the model level

Code Verification
Gain confidence in the generated code
Equivalence testing
Prevention of unintended functionality

Model used for production code generation
Generated C code
Object code

Workflow approved by TÜV SÜD for development of safety-critical software in accordance with ISO 26262 (automotive), IEC 61508 (industrial), EN 50128 (railway), IEC 62304 (medical devices)
Conclusion

It is easier and less expensive to fix design errors early in the process when they happen.

Model-Based Design enables:

1. *Early testing to increase confidence in your design*
2. *Delivery of higher quality software throughout the workflow*

MATLAB EXPO 2016