Increasing Design Confidence

*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
Gaining Confidence in our Design

Confidence

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

Effort / Time
Application: Cruise Control
Control speed according to setpoint

50 km/h
Application: Cruise Control

ECU

1. Cruise Control Module (MBD)
2. Legacy code

ECU system

Fuel Rate Control Module

Shift Logic Control Module

System Inputs

Outputs
Application: Cruise Control

ECU

System Inputs

ECU system

Cruise Control Module (MBD)

Legacy code

Fuel Rate Control Module

Shift Logic Control Module

Outputs
Application: Cruise Control

**Inputs**
- Cruise_onoff
- Brake
- Speed
- Coast set
- Accel reset

**Outputs**
- Engaged
- Target speed

Cruise Control Module (MBD)
Gaining Confidence in our Design

- Ad-hoc testing
- Design error detection
- Functional & structural tests
- Modeling & coding standards
- Code equivalence & integration checks
Ad-hoc Tests

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

Confidence

Effort / Time

Ad-hoc testing
Design error detection

Functional & structural tests
Modeling & coding standards
Code equiv. & integration checks
Finding Design Errors: Dead Logic

Design Error Detection
- Dead logic
- Identify active logic

Results:
- Generate detailed analysis report
- Open harness model

Design error detection completed normally.
2/70 objectives are dead logic.
68/70 objectives are active logic.

Compute target speed.CRUISE.ON."[after(incdec/holdrate... *10, tick)]
Transition: Transition trigger expression F DEATH LOGIC
Transition: Transition trigger expression T ACTIVE LOGIC
Finding Unintended Behavior

- **Dead logic** due to "uint8" operation on `incdec/holdrate*10`
- **Fix** change the order of operation `10*incdec/holdrate`
Gaining Confidence in our Design

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

Confidence

Effort / Time
Simulation Testing Workflow

Requirements

Did we completely test our model?

Design

Did we meet requirements?

Review functional behavior

Functional

Structural

Structural coverage report

Did we meet requirements?
Did We Completely Test our Model?

Model Coverage Analysis

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

![Stateflow chart showing CruiseControl/Compute target speed - Simulink]

```
(!AccelResSw || ...)
(tspeed >= maxtspeed)

(hasChangedTo(CoastSetSw)
(tspeed <= mintspeed)
```

Coverage: CruiseControl_Harness

Chart "Compute target speed"

Full Coverage
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

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

Effort / Time vs. Confidence
Model Advisor – Model Standards Checking
Gaining Confidence in our Design

- Ad-hoc testing
- Design error detection
- Functional & structural tests
- Modeling standards
- Model & code equivalence checks
Equivalence Testing: Model vs SIL or PIL Mode Testing
Code Generation with Model-to-Code Traceability
Code Generation with Model-to-Code Traceability
Code Equivalence Check Results: Model vs Code
Code Equivalence Check Results: Model vs Code
Code Equivalence Check Results: Model vs Code

**Code Coverage**

<table>
<thead>
<tr>
<th>File Contents/Complexity</th>
<th>D1</th>
<th>CI1</th>
<th>MCDC</th>
<th>Stmt</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. CruiseControlInit</td>
<td>22.97%</td>
<td>98%</td>
<td>96%</td>
<td>100%</td>
</tr>
<tr>
<td>2. CruiseControl</td>
<td>1</td>
<td>--</td>
<td>--</td>
<td>100%</td>
</tr>
<tr>
<td>3. CruiseControlInitial</td>
<td>20.97%</td>
<td>98%</td>
<td>96%</td>
<td>100%</td>
</tr>
<tr>
<td>4. CruiseControlInitialize</td>
<td>1</td>
<td>--</td>
<td>--</td>
<td>100%</td>
</tr>
</tbody>
</table>

```c
/* Output and update for referenced model: 'CruiseControl' */
void CruiseControl(const boolean_T rtu_CruiseOnOff, const boolean_T rtu_Brake, 
const uint8_T *rtu_Speed, const boolean_T *rtu_CoastSetSw, 
const boolean_T *rtu_AccelResSw, boolean_T *rtu_engaged, 
uint8_T *rtu_tspeed);

if (CruiseControl_DW.temporalCounter_i1 < MAX_uint32_T) {
    Sim Output 1(CruiseControl : normal)
    Sim Output 2(CruiseControl : normal)
    Not_Engaged_with_Enable
    Disengage_with_Disable
    Engage_1st_with_CoastSetSW
    No_Engage_1st_with_AccelResSW
    Disengage_with_Brake
    Re_Firm_with_CoastSetSw
}
```

**Decisions analyzed:**

<table>
<thead>
<tr>
<th>CruiseControl_DW.temporalCounter_i1 &lt; MAX_uint32_T</th>
<th>#1</th>
<th>#2</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>0/399</td>
<td>--</td>
<td>0/420</td>
</tr>
<tr>
<td>true</td>
<td>399/399</td>
<td>--</td>
<td>420/420</td>
</tr>
</tbody>
</table>
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

ECU

System Inputs

ECU system

Cruise Control Module (MBD)

Fuel Rate Control Module

Shift Logic Control Module

Outputs

Legacy code

Code Integration Analysis
Code Integration Analysis

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

ECU system

Legacy code

Gear
Engaged
Target speed
Fuel Rate
Finding Dead Code During Integration

ECU

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

Outputs
- Gear
- Engaged
- Target speed
- Fuel Rate

Inaccurate scaling for speed

Legacy code

Cruise Control Module (MBD)
Fuel Rate Control Module
Shift Logic Control Module

Dead code
Finding Dead Code with Polyspace

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

Maximum target speed = 90

Dead code

```c
/* Entry 'STANDBY': '<S5>:52' */
*rtv_Engaged = false;
} else if (rtu_Speed > maxtspeed) |
/* Transition: '<S5>:55' */
/* Exit Internal 'ON': '<S5>:54' */
localDW->is_ON = IN_NO_ACTIVE_CHILD;
localDW->is_CRUISE = IN_STANDBY;

/* Entry 'STANDBY': '<S5>:52' */
*rtv_Engaged = false;
} else if (rtu_Speed < mintspeed) {
/* Transition: '<S5>:113' */
```
Root Cause for Dead Code: Speed Sensor Input Hand 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 counts to mph.

Overlooked changing CONV_FACTOR for new ADC.
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;
    }
}
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++;
        }
    } else {
        i = get_bus_status();
    }
    if (i >= 0) {
        *(p - i) = 10;
    }
}
Gaining Confidence in our Design

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

Textual requirements → Executable specification → Model used for production code generation

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
Change the world by

Accelerating the pace
of discovery, innovation, development, and learning
in engineering and science