Verify, Validate and Document Models and Code
The Challenges

- Product Innovation
- ‘Going Smart’, ‘Internet of Things’
  → Software everywhere

How will you know your system works?
Three Key Takeaways

- Find problems and bugs early in the design and code
- Use mathematical analysis methods to prove software correctness
- Reproduce field issues via property proving
Application example: Cruise Control

Issues during testing

1) Wasting a couple of days on the test bench to find a code integration issue without success

2) While going downhill, target speed increase with “reduce speed” button
Verification and Validation in Context

- Ad-hoc tests
- Design error checks
- Functional tests/Model coverage
- Equivalence tests

Effort / Time

Confidence

- RESEARCH
- REQUIREMENTS

ANALYSIS – SPECIFICATION - DESIGN
- Model
  - Architecture
  - Algorithms
  - Schematics
  - Environment
  - Constraints
  - Physical Domains

IMPLEMENTATION
- C, C++
- VHDL, Verilog
- Structured Text
  - MCU
  - DSP
  - FPGA
  - ASIC
  - PLC
  - PAC

TEST CASES

TEST & VERIFICATION

INTEGRATION
Ad-hoc testing
Customizable Reports
Verification and Validation in Context

Confidence vs. Effort / Time

- Ad-hoc tests
- Equivalence tests
- Functional tests/Model coverage
- Design error checks

RESEARCH

ANALYSIS – SPECIFICATION- DESIGN

- Architecture
- Environment
- Algorithms
- Constraints
- Schematics
- Physical Domains

IMPLEMENTATION

- C, C++
- VHDL, Verilog
- Structured Text
- MCU
- DSP
- FPGA
- ASIC
- PLC
- PAC

INTEGRATION

TEST & VERIFICATION

TEST CASES

REQUIREMENTS

TEST CASES
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

- **Fix** change the order of operation 10*incdec/holdrate

Condition can never be false
Finding Unintended Behavior

- Dead logic due to “uint8” operation on incdec/holdrate*10
- Fix change the order of operation 10*incdec/holdrate

MathWorks
Verification and Validation in Context

Confidence vs. Effort / Time

- Ad-hoc tests
- Design error checks
- Functional tests / Model coverage
- Equivalence tests

RESEARCH → REQUIREMENTS

ANALYSIS – SPECIFICATION – DESIGN
- Architecture
- Algorithms
- Schematics
- Environment
- Constraints
- Physical Domains

IMPLEMENTATION
- C, C++
- VHDL, Verilog
- Structured Text
- MCU, DSP, FPGA, ASIC, PLC, PAC

INTEGRATION

TEST & VERIFICATION

TEST CASES

MATLAB CONFERENCE 2015
Simulation Testing Workflow

Did we meet requirements?

Review functional behavior

Did we completely test our model?

Structural coverage report
Verification and Validation in Context

Confidence

Effort / Time

- Ad-hoc tests
- Design error checks
- Functional tests/Model coverage
- Equivalence tests

RESEARCH

REQUIREMENTS

ANALYSIS – SPECIFICATION- DESIGN

MODEL

- Architecture
- Environment
- Algorithms
- Constraints
- Schematics
- Physical Domains

IMPLEMENTATION

- C, C++
- VHDL, Verilog
- Structured Text
- MCU
- DSP
- FPGA
- ASIC
- PLC
- PAC

INTEGRATION

TEST CASES

TEST & VERIFICATION
Equivalence Testing
Equivalence Testing

Model

Configuration Parameters: plcdemo_cruise_control/Configuration (Active)

Create code generation report

Code Generation Report for 'Controller'

Summary

Code generation for model "Controller"

Model version: 1.189
Simulink Coder version: 8.8 (R2015a) 69-Feb-2015
C source code generated on: Thu May 28 12:18:09 2015

Configuration settings at the time of code generation: click to open
Code generation objectives: Unspecified
Validation result: Not run

MATLAB CONFERENCE 2015
Verification and Validation in Context

Confidence

Effort / Time

Ad-hoc tests

Design error checks

Functional tests/ Model coverage

Equivalence tests

RESEARCH

REQUIREMENTS

ANALYSIS – SPECIFICATION - DESIGN

MODEL

- Architecture
- Algorithms
- Schematics

Environmental
- Constraints
- Physical Domains

IMPLEMENTATION

- C, C++
- VHDL, Verilog
- Structured Text

- MCU
- DSP
- FPGA
- ASIC
- PLC
- PAC

INTEGRATION

TEST CASES

TEST & VERIFICATION

RESEARCH

REQUIREMENTS

ANALYSIS – SPECIFICATION - DESIGN

MODEL

- Architecture
- Algorithms
- Schematics

Environmental
- Constraints
- Physical Domains

IMPLEMENTATION

- C, C++
- VHDL, Verilog
- Structured Text

- MCU
- DSP
- FPGA
- ASIC
- PLC
- PAC

INTEGRATION

TEST CASES

TEST & VERIFICATION
Verification and Validation in Context

Confidence

Effort / Time

- Ad-hoc tests
- Design error checks
- Functional tests/ Model coverage
- Equivalence tests

---

TEST & VERIFICATION

- RESEARCH
- REQUIREMENTS

ANALYSIS – SPECIFICATION- DESIGN

- MODEL
  - Architecture
  - Algorithms
  - Schematics
  - Environment
  - Constraints
  - Physical Domains

IMPLEMENTATION

- C, C++
- VHDL, Verilog
- Structured Text

- MCU
- DSP
- FPGA
- ASIC
- PLC
- PAC

INTEGRATION
Application Example: Cruise Control
## Checking Source Code

<table>
<thead>
<tr>
<th>What Polyspace does</th>
</tr>
</thead>
<tbody>
<tr>
<td>Checks coding rule conformance (MISRA-C/C++, JSF++, Custom)</td>
</tr>
<tr>
<td>Provides metrics (Cyclomatic complexity etc.)</td>
</tr>
<tr>
<td>Quickly finds potential errors</td>
</tr>
<tr>
<td>Proves the existence of errors</td>
</tr>
<tr>
<td>Proves the absence of errors</td>
</tr>
<tr>
<td>Indicates when you’ve reached the desired quality level</td>
</tr>
</tbody>
</table>

+ No test cases
+ No compilation
Color Coding in Code

- **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

```java
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;}
    }
```
Identify Run-Time Error in Integrated Code

```
/* sum: '<S1>/Sum1' */
localB->Sum1 = rty_Tspeed - rtu_Speed;

/* Outputs for Enabled SubSystem: '<S1>/PI Controller' incorporate
 * EnablePort: '<S1>/Enable'
 */
if (*rty_Engaged) {
    /* Sum: '<S1>/Sum' incorporates:
        * DiscreteIntegrator: '<S1>/Discrete-Time Integrator'
        * Gain: '<S1>/Kp'
        * Gain: '<S1>/Kp1'
    */
    rty_Trottle = 0.02 * localsB->Sum1 + 0.01 *
    Dereference of parameter 'rty_Trottle' (pointer to float 64, size: 64 bits):
    Pointer is not null.
    Points to 8 bytes at offset 512 in buffer of 8 bytes, so is outside bounds.
    Pointer may point to variable or field of variable:
    'gvar_B_ECU_system'.
}

/* Update for Enabled SubSystem: '<S1>/PI Controller' incorporate */
```
Identify Run-Time Error in Integrated Code

```c
if (*rt_y_Engaged) {
    /* Sum: '<S5>/Sum' incorporates:
       * DiscreteIntegrator: '<S5>/Discrete-Time Integrator'
       * Gain: '<S5>/Kp'
       * Gain: '<S5>/Kp1'
     */
    *rt_y_Throttle = 0.02 * locals->Sum1 + 0.01 * localDW->DiscreteTimeIntegrator_DSTATE;
}
/* End of Outputs for SubSystem: '<S1>' */

/* Update for atomic system: '<Root>/Cruise_ctrl_Update' */
void Cruise_ctrl_Update(boolean_T *rt_y_Cruise_ctrl_Update_T *localDW) {
    /* Update for Enabled SubSystem: '<S1>' */
    /* Update for EnablePort: '<S5>/Enable' */
}
```
Verification and Validation in Context

Confidence

Effort / Time

Is there more?

RESEARCH

REQUIREMENTS

ANALYSIS – SPECIFICATION - DESIGN

MODEL

Architecture

Environment

Algorithms

Constraints

Schematics

Physical Domains

IMPLEMENTATION

TEST CASES

TEST & VERIFICATION

INTEGRATION

C, C++, VHDL, Verilog, Structured Text

MCU, DSP, FPGA, ASIC, PLC, PAC

Design error checks

Functional tests/Model coverage

Equivalence tests

Integration tests

Ad-hoc tests

TEST & VERIFICATION

INTEGRATION

C, C++, VHDL, Verilog, Structured Text

MCU, DSP, FPGA, ASIC, PLC, PAC

Research

Requirements

Analysis – Specification - Design

Model

Architecture

Environment

Algorithms

Constraints

Schematics

Physical Domains

Implementation

Test cases

Test & Verification

Integration

C, C++, VHDL, Verilog, Structured Text

MCU, DSP, FPGA, ASIC, PLC, PAC
Using Model-Based Design to Reproduce Field Issue

- **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
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 produce a test case

MATLAB CONFERENCE 2015
Generated Test Case to Reproduce Field Issue

- Automatic generation of harness model
Debugging

- Going downhill, actual speed could increase (say to 25mph)

- If coast button (reduce speed) is set again, target speed takes on actual speed value of 25mph (jumps up from 20mph)
Verification and Validation Best Practice

- Ad-hoc tests
- Design error checks
- Functional tests/Model coverage
- Requirement proofs
- Equivalence tests
- Integration tests
- Field tests

Confidence vs. Effort / Time
Three Key Takeaways

- Find problems and bugs early in the design and code
- Use formal methods to prove software correctness
- Reproduce field issues via property proving