Early Verification and Validation in Model-Based Design

Amory Wakefield
Technical Marketing
The MathWorks
Introductions

I spend most of my time:

A. Creating specifications and requirements (systems and software)

B. Implementation based on specification and requirements created by somebody else (generating, writing, deploying, debugging code)

C. Both

D. None of the above
Address the Entire Development Process

**Requirements**

**Design**
- Environment
- Physical Components
- Algorithms
- VHDL, Verilog
- C, C++
- FPGA, ASIC, MCU, DSP

**Implementation**

**System V&V**
- Requirements Validation
- Robustness Testing
- Modeling Standards Checking

**Component V&V**
- Design Verification
- Model Testing
- Coverage and Test Generation
- Property Proving
- Code Verification
- Code Correctness
- Processor-In-The-Loop Testing

**Integration Testing**
- Software Integration Testing
- Hardware-in-the-Loop Testing
- Hardware Connectivity

**Hand Generate**

**Generate**

**Generate**
Methods for Verification and Validation

- Traceability
  - Requirements to model and code
  - Model to code
- Modeling and Coding Standards
  - Modeling standards checking
  - Coding standards checking
- Testing
  - Model testing in simulation
  - Processor-in-the-loop
- Proving
  - Proving design properties
  - Proving code correctness
Increasing Confidence in Your Designs

Verification Method

- Traceability
- Modeling and Coding Standards Checking
- Model and Code Testing
- Proving Design Properties and Code Correctness

Confidence
Traceability

- **Tracing Requirements ↔ Model**
  Simulink Verification and Validation

- **Tracing Model ↔ Source Code**
  Real-Time Workshop Embedded Coder

- **Tracing Requirements ↔ Source Code**
  Simulink Verification and Validation

- Hand-Generate
- Generate
- Generate

- Digital Electronics
  - VHDL, Verilog
  - FPGA
  - ASIC

- Embedded Software
  - C, C++
  - MCU
  - DSP

- Environment
- Physical Components
- Algorithms

- Design

- Functional Requirements

- System V&V
  - Comp. V&V
  - Integration

- Implement
- Integration

- MathWorks Automotive Conference ’08
Tracing Requirements ↔ Model
Simulink Verification and Validation

- Creating links between text documents and model objects

3 Functional Requirements

This paragraph describes the logic of a seat belt reminder functionality you can find almost in all medium class cars. The goal of this function is to warn the user about unfastened seat belts.

3.1 KEY OFF

At key off, the seatbelt icon which is showed in the car dashboard must be switched off (SeatBeltIcon = 0).

3.2 KEY ON

During key on the seat belt icon status is related to the two inputs Speed and SeatBeltPlaced. If the seat belt is fastened, SeatBeltIcon is always off. If the seat belt is unfastened, we consider two different cases.

3.2.1 LOW SPEED

If the speed is less or equal than 15 km/h, the seat belt icon is on.

3.2.2 HIGH SPEED

If the speed is greater than 15 km/h, the seat belt icon must flash 1 Hz with a 50% duty cycle. This warning is to capture the driver's attention that he needs to fasten his seat belts (an extra buzzer could also be used).
Tracing Requirements ↔ Source Code
Simulink Verification and Validation
Real-Time Workshop Embedded Coder

- Including requirements in the generated source code
Requirements Traceability—Overview

Simulink Verification and Validation

- Bidirectional linking with external documents
  - For Simulink and Stateflow
  - Extensibility API
  - Report generation
- DOORS integration
  - Linking with read-only requirement documents
- Real-Time Workshop Embedded Coder integration
  - Embeds requirements as comments in source code

Supported document formats:
- Telelogic DOORS
- Microsoft Word
- Microsoft Excel
- PDF
- HTML
- Text

MathWorks
Automotive Conference ’08
Modeling and Coding Standards

- **Modeling Standards Checking**
  Simulink Verification and Validation

- **Coding Standards Checking**
  PolySpace™ Client for C/C++

Diagram showing the process of Design, Implementation, and Verification.

- **Design**
  - Environment
  - Physical Components
  - Algorithms

- **Implementation**
  - Digital Electronics
    - VHDL, Verilog
  - Embedded Software
    - C, C++
  - FPGA, ASIC, MCU, DSP

- **Verification and Validation**
  - System V&V
  - Comp. V&V
  - Integration
Simulink Model Advisor

Model Checking

- Enforce modeling best practices
- Detect and troubleshoot modeling and code generation issues
- Check models for (a subset of) known version upgrade issues

- Automated report is a useful process audit document:
  - More detailed summary
  - Valid check states: Pass, Fail, Warning, and Not Run
Modeling Standards Checking
Simulink Verification and Validation

- Static analysis of models for
  - Requirement consistency
  - Custom checks for company-specific processes
- Standards
  - MAAB Style Guidelines
  - DO-178B
  - IEC 61508
  - Custom – using extensibility API

- Benefits
  - Prevent problems early in the design process
  - Automate time consuming review work
MAAB Style Guide Checks

- MathWorks Automotive Advisory Board (MAAB)

- Consistency
- Interoperability
- Error prevention
- Knowledge sharing
IEC 61508 Modeling Standards Checks

This Absolute Value block is operating on an unsigned value which could result in unreachable code.

This Relational Operator block is not outputting a Boolean data type which may lead to unpredictable results in the generated code.
Coding Standards Checking
PolySpace Client for C/C++

- Configure rules and run as part of the static check of the C source code
- MISRA-C:2004
- Covers 122/142 rules
  - 102 fully supported
  - 20 partially supported
Testing

Functional Requirements

- Model Testing
  - SystemTest
  - Simulink Verification and Validation
  - Simulink Design Verifier

  Verify that design meets requirements

- Code Testing
  - Real-Time Workshop
  - Embedded Coder
  - Embedded IDE Link products
  - Target Support Package products

  Verify that the behavior of source code and object code matches the model
Demo

- Model testing using test cases stored in Excel
Improving Test Suite
Simulink Design Verifier

- Generating tests to reach coverage criteria

Test generation harness with the copy of the original model

Test inputs that ensure complete coverage
Code Testing with Generated Signals

**Simulink**

- Software-in-the-loop
  - On the host
- Processor-in-the-loop
  - On the target processor

- Independent code testing environment
  - Generated signals and model outputs are saved as a .mat data file
  - Exported input signals drive code tests
  - Exported model outputs become expectation values for code testing
Proving Design Properties
Simulink Design Verifier
Prove that design meets the key functional requirements

Proving Code Correctness
PolySpace Server for C/C++
Prove that code meets non-functional runtime requirements
Proving Properties – Workflows

Simulink Design Verifier

1. Authoring
   - Highly Iterative
   - Leads to improvement in design and in specifications

2. Execution and Reporting
   - Automated
   - Part of the regression testing harness

Benefits

- Leads to precise definition of low level functional requirements
- Once established, properties represent a model of design behavior
- Minimizes chance of implementing undesired behavior
Proving Code Correctness

- Verifying code integration
- Certification requirements
- Reliability concerns

- Proving Code Correctness
  PolySpace Server for C/C++
Code Correctness

Formal method: Abstract Interpretation

Results are proven for all possible executions of the code!!

```java
static void Pointer_Arithmetic()
{
    int tab[100];
    int i, *p = tab;

    for(i = 0; i < 100; i++, p++)
    {*
p = 0;

    if(get_bus_status() > 0)
    {
        if(get_oil_pressure() > 0)
            *p = 5; /* Out of bounds */
        else
            i++;
    }

    i = random_int();
    if (random_int()) *(p-i) = 10;

    if (0<i && i<=100)
    { p = p - i;
        *p = 5; /* Safe pointer access */
    }
}
Summary

- Model-Based Design enables early verification and validation

- Early verification and validation methods improve and optimize your existing development process

- Early problem detection significantly reduces time spent debugging – shorter time to resolution