Verifying and Validating Automatically Generated Code

Andrew Burnard
Land Rover
Electrical & Electronics
aburnard@landrover.com

Contents

• Verification and Validation
• Validation of the Code Generator
• Verification of the Generated Code
• Recommended V&V techniques
  • Technical problems
• Future tool improvements
Verification & Validation

• **Verification**
  - IEEE definition
    - ‘The process of evaluating a system or component to determine whether the products of a given phase satisfy the conditions imposed at the start of that phase’
    - *Are we building the system right?*

• **Validation**
  - IEEE definition
    - ‘The process of evaluating a system or component during or at the end of the development process to determine whether it satisfies the requirements’
    - *Are we building the right system?*

---

**Manual Coding V-Model**

- System Req.
- Controller Req.
- Software Design
- Software Coding
- Code
- Improvements
- System V&V
- Controller V&V
- Integration V&V
- Software Unit V&V
V & V of Automatically Generated Code

- Validating the Code Generator
  - Code Generator works for any model
- Verifying the Generated Code
  - Checking that Generated Code is correct

Validating the Code Generator

- Complex Software tool
  - Need to minimise errors
Tool Development Process

• Good process improves tool quality
• Proof of process
  - Presentation
  - Audit
  - Certified, e.g. SPICE, TickiT, CMM

Testing the Code Generator

• Developer test suite
  - More comprehensive
  - Not rigorous
• User test suite
  - Tailored to user
  - E.g. Autocoder validation project
    - TUV, Continental Teves and Ford
• Confidence through experience
  - User has few problems
Certified Code Generator

- Similar complexity as certifying a compiler
- Increases development time
- Increases the cost of the tool

Formal proof

- Rigorous
- Very complex
- Impossible?
Verifying Generated Code

- Is the code a correct implementation of the model
- Traditional methods for verifying manually generated code:
  - Peer review
  - Testing

![Flowchart showing the process from model to verified code with review and test steps]

Peer Reviewing (Manual Code)

- Maintenance
  - Documentation
  - Design
  - Structure
- Coding Errors
  - Random
- Functionality (Pre-model based design)
Peer Reviewing (AGC)

- Maintenance
  - Documentation
    - Automatic
  - Structure and Design
    - Automatic, influenced by Model structure and design
    - Review of model more beneficial
- Errors
  - Not random
    - Code generator is deterministic
  - Low density of errors
  - Unlikely to be detected by human review
- Functionality
  - Model Review

Testing Code

- Manual Code Generation
  - Frequently validating functionality
- Automatic Code Generation
  - Verify code with model
- Functionality
  - Validated using model simulation
### Comparing Test Results

- **There will be differences between simulation and code outputs**
  - What is acceptable?
- **Differences between floating point and fixed point**
  - Fixed point is usually less accurate
  - Quantisation errors
  - Saturation errors
- **But floating point can also be wrong**
Floating Point vs Fixed Point

Verifying the Generated Code

Error in Floating Point Results
Tools

- Static analysis
  - MISRA-C
  - Influences the Model design
- Complexity analysis
  - Influences the Model design
- Static program analysis
  - Checks for ‘bugs’ in the generated code
  - Data range testing

Static Program Analysis

- Static determination of run-time properties of code
- Abstract Interpretation
  - Analyse source code to statically locate run-time errors
- Errors analysed include:
  - Read access to non-initialized data,
  - De-referencing through null and out-of-bounds pointers,
  - Out-of-bounds array access,
  - Invalid arithmetic operations such as division by zero,
  - Overflow / underflow on arithmetic operations for integers and floating point numbers,
  - Illegal type conversions, e.g.: long to short, float to int,
  - Access conflicts for data shared between tasks,
  - Non-terminating function calls and loops,
  - Unreachable code (dead code).
Selected V&V Techniques for Automatically Generated Code

- Code Generator
  - Suppliers responsibility
- Peer Review
  - Not the code
  - Interfaces to ‘Base Software’
- Test
  - Prove what behaves like Simulink ‘is’ Simulink
  - Coverage: 100% MCDC
  - Abstract Interpretation – Remove ‘bugs’
- Static analysis
  - Model Review

Code Coverage

- Statement coverage
  - 1 Test required:
    - \((a > 1) \text{ AND } (b > 2)\)
- Branch coverage
  - 2 Tests required
    - \((a > 1) \text{ AND } (b > 2)\) if true
    - \((a <= 1) \text{ OR } (b <=2)\) if false
- Modified Condition and Decision Coverage (MCDC)
  - 3 Tests required
    - \((a > 1) \text{ AND } (b > 2)\) if true
    - \((a <= 1)\) if false 1st condition
    - \((a > 1) \text{ AND } (b <= 2)\) if false 2nd condition

```
if (a > 1) && (b > 2)
{
    c = 3;
}
```
Generating Test Cases

- **Custom Tool**
  - Based on Excel
  - Specify test cases
  - Simulate model or execute compiled code
- **Test cases**
  - User specifies inputs, calibration and tolerance
  - Simulation records outputs
  - Reference outputs compared to compiled code outputs
- **Results**
  - Pass/Fail – Code is same as model
  - Code coverage
- **Regression testing**

Coding Errors Found

- **Type definition and scaling errors**
  - Most common error
  - User rapidly fixes the problems before releasing code
- **Re-using variables**
  - Difficult to find
  - Comparison to simulation revealed the error
- **Confusion between bit-wise and logical ops in SF**
  - Code generator doesn’t check SF flag
- **Incorrect code from code generator!**
- **Model functionality Errors**
  - User’s reviewing the results of test
Unreachable Code

• Due to the Automatic Code Generator
  - Overhead of using AGC or Defensive programming
  - Every time model construct is encountered
  - Dispensation on coverage criteria
• Unreachable construct in Model
  - Error in model functionality
  - Need to be ‘fixed’

Recommendations

• Thorough code testing
  - Implementation errors
  - Required even if no code generator errors
• No Source code review
  - Model review is effective review for functionality
  - No random (human) coding errors
  - Some configuration errors can be located using static analysis
  - Systematic errors due to code generator are detected by testing
  - Although, a review of the generated code and ‘base’ software interface is required
• Data coverage
  - Static program analysis
**Improvements**

- **Data Coverage**
  - Matlab and Simulink are data analysis tools
- **Variable data**
  - Min/Max values
  - Scatter diagram for variables
  - 'Closeness' to comparators
- **Equivalence class analysis**
  - Analyse model for equivalent classes

**Questions**