Proving the Shalls in Practice: Formal Requirements and Proofs in Commercial Avionics Software

Mike Whalen
Program Director
University of Minnesota Software Engineering Center
Acknowledgements

- Rockwell Collins (Steven Miller, Darren Cofer, Lucas Wagner)
- University of Minnesota (Mats P. E. Heimdahl)
- NASA Langley Research Center (Ricky Butler)
- Air Force Research Labs (RD Directorate)
- Lockheed Martin (Walter Storm)
- NSA (Randy Johnson and Brad Martin)
Who Am I?

- **My main aim is in reducing verification and validation (V&V) cost and increasing rigor**
  - Applied automated V&V techniques on industrial systems at Rockwell Collins for 6 ½ years
  - Proofs, bounded analyses, static analysis, automated testing
  - Combining several kinds of assurance artifacts
  - I’m interested in requirements as they pertain to V&V.

- **I am not a requirements researcher**

- I have a lot of experience formalizing requirements
  - …For discrete systems
  - …Using synchronous observers and temporal logic
  - PhD in translation proofs (2005)
DoD software is growing in size and complexity

Total Onboard Computer Capacity (OFP)


Robert Gold, OSD
Airbus Aircraft Software Growth


Slide © Rockwell Collins, 2006
Emerging Software Size and Complexity

- Advanced system attributes (on-board intelligence and adaptive control laws) will be required to accommodate emerging functional requirements.
- This will increase the size and complexity of control systems beyond the capability of current V&V practices.

Slide courtesy Lockheed Martin, Inc. August, 2011  RE 2011: Mike Whalen
Current V&V Processes

- Concept Formation
- Requirements Specification
- Design
- Implementation
- Object Code
- Analysis
- Test
- Analysis
- System Test
- Unit Test
- Integration Test
- Integration
- System

August, 2011  RE 2011: Mike Whalen
Verification in the Model-Based Development Context

Concept Formation

Requirements

Model

Object Code

Properties

System

Integration

Analysis

Manual + Autogenerated Tests

August, 2011   RE 2011: Mike Whalen
Model Checking

Even Small Systems Have Trillions (of Trillions) of Possible Tests!

Testing Checks Only the Values We Select

Finds every exception to the property being checked!

Model Checker Tries Every Possible Value!
An Example: FCS 5000 Flight Control Mode Logic Requirements

- **1 Mode Annunciations**
  - **1.1 Selection**
    - If this side is active and the mode annunciations are off, the mode annunciations shall be turned on when the onside FD is turned on.
  - **1.2 Deselection**
    - If this side is active and the mode annunciations are on, the mode annunciations shall be turned off if the onside FD is off, the offside FD is off, and the AP is disengaged.
    - If this side is active and the mode annunciations are on, the mode annunciations shall not be turned off if the onside FD is on, or the offside FD is on, or the AP is engaged.
  - **1.3 Operation**
    - The mode annunciations shall not be on at system power up.
    - If this side is active the mode annunciations shall be on if and only if the onside FD cues are displayed, or the offside FD cues are displayed, or the AP is engaged.
FCS5000: Constructive Modeling
FCS5000: Constructive Modeling

Mode Controller A

Mode Controller B

Modeled in Simulink

16 System Modes
73 events
272 Transitions
6.8 x 10^{21} Reachable States
# FCS-5000 Formalized Requirements

<table>
<thead>
<tr>
<th>Ref.</th>
<th>English Requirements</th>
<th>SMV Proof</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><strong>1 Mode Annunciations</strong></td>
<td></td>
</tr>
</tbody>
</table>

## 1.1 Selection

1.1.0-1 If this side is active and the mode annunciations are off, the mode annunciations shall be turned on when the inside FD is turned on.

- SPEC AG((Mode_Annunciations_On & Inside_FD_On) -> AX(\(ls\_This\_Side\_Active = 1 \& \text{Inside\_FD\_On}\) -> Mode_Annunciations_On))

1.1.0-2 If this side is active and the mode annunciations are off, the mode annunciations shall be turned on when the offside FD is turned on.

- SPEC AG((Mode_Annunciations_On & Offside_FD_On = FALSE) -> AX(\(ls\_This\_Side\_Active = 1 \& \text{Offside\_FD\_On = TRUE}\) -> Mode_Annunciations_On))

1.1.0-3 If this side is active and the mode annunciations are off, the mode annunciations shall be turned on when the onside FD is turned on.

- SPEC AG((Mode_Annunciations_On & Inside_FD_On) -> AX(\(ls\_This\_Side\_Active = 1 \& \text{Inside\_FD\_On}\) -> Mode_Annunciations_On))

## 1.2 Deselection

1.2.0-1 If this side is active and the mode annunciations are on, the mode annunciations shall be turned off if the onside FD is off, the offside FD is off, and the AP is disengaged.

- SPEC AG((Mode_Annunciations_On -> AX(\(ls\_This\_Side\_Active = 1 \& \text{Inside\_FD\_Off} \& \text{Offside\_FD\_Off} \& \text{AP\_Disengaged}\) -> !Mode_Annunciations_On))

1.2.0-2 If this side is active and the mode annunciations are on, the mode annunciations shall not be turned off if the onside FD is on, or the offside FD is on, or the AP is engaged.

- SPEC AG((Mode_Annunciations_On -> AX(\(ls\_This\_Side\_Active = 1 \& (\text{Inside\_FD\_On} \text{\lor \text{Offside\_FD\_On}} \text{\lor \text{AP\_Engaged}})\) -> !Mode_Annunciations_On))

## 1.3 Operation

### 1.3.0-1 The mode annunciations shall not be on at system power up.

- SPEC !(Mode_Annunciations_On)

### 1.3.0-2 If this side is active the mode annunciations shall be on if and only if the onside FD cues are displayed, or the offside FD cues are displayed, or the AP is engaged.

- SPEC !(ls\_This\_Side\_Active = 1) -> (Mode_Annunciations_On <-> (Inside\_FD\_On | Offside\_FD\_On = TRUE | AP\_Engaged))
Checked ~280 properties

- Found Many Errors
  - 27 errors found in the model
  - Equally many errors were due to errors in shall statements!
Iterative Verification

- Concept Formation
- Requirements
- Analysis
- Properties
- Model
- Integration
- Object Code
- System

Manual + Autogenerated Tests

August, 2011  RE 2011: Mike Whalen
Main Message

To be able to develop large systems with high confidence: **requirements must be more tightly integrated into development**

- Recent tools can automate generation of many V&V artifacts
  - Test cases to high level of coverage (e.g. MCDC)
  - Proofs
- *...but tools can only be effective with (relatively) complete requirements written in a formal notation*
- Analysis tools illuminate when requirements are faulty
Some Applications
Example Requirement:
Drive the Maximum Number of Display Units
Given the Available Graphics Processors

Counterexample Found in 5 Seconds

Checked 573 Properties -
Found and Corrected 98 Errors
in Early Design Models

Modeled in Simulink
Translated to NuSMV
4,295 Subsystems
16,117 Simulink Blocks
Over $10^{37}$ Reachable States

ADGS 2100 Adaptive Display and Guidance System
CerTA FCS Phase I

- Sponsored by AFRL
  - Wright Patterson VA Directorate
- Compare FM & Testing
  - Testing team & FM team
- Lockheed Martin UAV
  - Adaptive Flight Control System
  - Redundancy Management Logic
  - Modeled in Simulink
  - Translated to NuSMV model checker

### Phase I Results

<table>
<thead>
<tr>
<th>Subsystem/Block</th>
<th>Charts / Transitions / TT Cells</th>
<th>Reachable State Space</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Triplex voter</td>
<td>10 / 96</td>
<td>3 / 35 / 198</td>
<td>6.0 * 10^{13}</td>
</tr>
<tr>
<td>Failure processing</td>
<td>7 / 42</td>
<td>0 / 0 / 0</td>
<td>2.1 * 10^{4}</td>
</tr>
<tr>
<td>Reset manager</td>
<td>6 / 31</td>
<td>2 / 26 / 0</td>
<td>1.32 * 10^{11}</td>
</tr>
<tr>
<td>Totals</td>
<td>23 / 169</td>
<td>5 / 61 / 198</td>
<td>N/A</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Effort (% total)</th>
<th>Errors Found</th>
</tr>
</thead>
<tbody>
<tr>
<td>Testing</td>
<td>60%</td>
</tr>
<tr>
<td>Model-Checking</td>
<td>40%</td>
</tr>
</tbody>
</table>

August, 2011  RE 2011: Mike Whalen  21
...And Several More

Examples of Using Formal Methods

AAMPS Certified Microprocessor

- Formal proof of the MILS security partitioning implemented in the AAMPS microprocessor.
- Example of the industrial use of the MILS partitioning in an AML.

Examples of Using Formal Methods

Integrity-178B Real-Time OS Evaluation

Turnstall

- High-assurance cross domain data exchange and communication between different systems.

High Assurance Systems

- HW/SW codesign
  - Target: 40 GB throughput
  - Core is controller
  - Translates rest of system
  - Specialized high assurance co-processor (bottom) implements specialized HW

- Formal analysis of controller using Stateflow/Simulink/Stateflow

Prover model checker

Challenges:
- Complex data structures make verification difficult
- Too much data for stateful verification, overwhelms tools
- Manual translation of properties into inductive invariants

Formal Analysis of a Triplex Sensor Voter in an Industrial Context

Michael Dierkes
Rockwell Collins France

FMICS 2011 workshop
August 30, 2011
Trento
What did we learn?
Model Checking Works!

- …On certain kinds of models
  - On every large project we examined we could apply model checking on important parts
  - **We have always found errors**
  - …but there are many places where it does not work
- **Where it works, it significantly outperforms testing**
- Argument for use must be economic, not “correctness”
  - Significant productivity benefits to requirements, modeling, and testing phases
  - Requirements and models are much better, so testing is significantly less expensive
  - Much cheaper to re-run analysis after requirements changes than to rewrite tests
  - Testing is the most expensive part of safety critical development
Requirements Are (at least) Equally Likely to be Wrong as Code

- Most declarative requirements documents are **inconsistent**
- Not obviously so: has to do with the interaction of requirements
- Example:
  - When button X is pressed, mode shall be A
  - When button Y is pressed, mode shall be B
  - **What happens when both X and Y are pressed simultaneously?**
- In some logics, this kind of inconsistency is **straightforward to automatically detect** (universal models for LTL or ACTL)
- Other kinds of inconsistency are much harder to detect; require detailed information about the environment & physics
Saying What We Want is Hard

- This is not new
- Writing down properties in a formal notation is can be challenging
  - But not harder than writing correct code!
- Hard part is actually figuring out exactly what you want.
- Doesn’t have to be done all at once
Nobody agrees on what a requirement is...

- **Rockwell Collins:**
  - multiple layers of “shall” statements

- **Lockheed Martin:**
  - Block diagrams for controllers
  - Operational scenarios

- **NSA:**
  - Set of Z schemas
    - Functional software specification
    - Handful of datatype invariants

- **Microsoft (almost everything):**
  - Parameterized unit tests…

- **Airbus:**
  - SCADE Models(!)
Declarative Requirements

- Define Acceptable Systems through Properties that
  - Relate Outputs to Inputs
  - Constrain the Set of Acceptable Models

- Make No Assumptions About Internal System Design

- Specify a Set of Acceptable Systems
  
  **Property 1:**
  - Output 2 > Input 1 + Input 2

  **Property 2:**
  - Output 1 = Input 1/Input 3

  **Property 3:**
  - Output 2 = Input 1
Define Acceptable System(s) by Constructing a Model

Start with a Set of Base Types
- Booleans, Integers, Reals, …

and a Set of Constructor Types
- Records, Tuples, Arrays, …

Advantages
- Intuitive
- Models are Always Consistent
- Models are Always Complete (a Behavior Defined for All Inputs)

Disadvantages
- Inherently Based on Internal Structure
- Strongly Suggests a Design
- Easy to Overconstrain Specification
# Strengths and Weaknesses of Specification Styles

<table>
<thead>
<tr>
<th></th>
<th>Natural Language</th>
<th>Declarative</th>
<th>Constructive Model</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Ambiguity</strong></td>
<td>Likely</td>
<td>Eliminated</td>
<td>Eliminated</td>
</tr>
<tr>
<td><strong>Inconsistency</strong></td>
<td>Likely</td>
<td>Possible</td>
<td>Eliminated</td>
</tr>
<tr>
<td><strong>Incompleteness</strong></td>
<td>Likely</td>
<td>Possible</td>
<td>Eliminated</td>
</tr>
<tr>
<td><strong>Implementation Bias</strong></td>
<td>Possible</td>
<td>Possible</td>
<td>Likely</td>
</tr>
<tr>
<td><strong>Comprehensibility to Domain Experts</strong></td>
<td>Easily Understood</td>
<td>Harder to Understand</td>
<td>Harder to Understand</td>
</tr>
</tbody>
</table>

*Early Life Cycle*  *Late*  

---

*Slide © Rockwell Collins, 2006*
Locality

- **Hypothesis:** easy to get “local” things right
- **Constructive model:**
  - Order of button presses is local and immediately visible
  - Constructive state machine is simpler and more compact representation than declarative one.
- **Declarative model:**
  - **Locality**
    - One perspective: relative location of requirements in document
    - Another: all properties are equivalently local (ordering doesn’t matter)
  - When models are complex, easier to describe abstract input-to-output properties than to understand the constructive model.
Systems are hierarchically organized
Requirements vs. design must be a matter of perspective
Need better support for $N$-level decompositions for requirements and design
  - DO-178B/C drive me crazy on this
    - two levels of decomposition: “high” and “low”
  - How do other standards do it?
Notation Matters... and It Doesn’t

• Computational model is very important,
  ◦ E.g., Dataflow, Declarative, Functional, Imperative
  ◦ Discrete Time vs. Dense Time
  ◦ DSLs for particular domains

• Syntax within a computational model is less important
  ◦ RSML-e vs. SCR vs. Simulink vs. SCADE

• Engineers have little difficulty learning temporal logic

• Engineers do not usually like formal requirements specification languages a la Z, B, VDM
If the mode is COOKING, then the microwave door shall be CLOSED.

\[
\text{SPEC AG}(\text{mode} = \text{COOKING} \rightarrow \text{door\_closed})
\]

Simulink

assert (!(\text{mode} == \text{COOKING}) \lor \text{door\_closed});

Java Code

Engineers get to choose!
void AddAndCount(List list, int item) {
  // Assume
  PexAssume.IsTrue(list != null);
  // Arrange
  var count = list.Count;
  // Act
  list.Add(item);
  // Assert
  Assert.AreEqual(count + 1, list.Count);
}

... adding I item increases Count by I
... given a non-null list
Capturing Environmental Assumptions

- Model checker finds any possible violation of a property
  - May be outside the operating envelope of the system
- Environmental assumptions are encoded as **invariants**
- May be related to…
  - Physical realism: “The vehicle is flying above ground”
  - Operating regions: “Only applicable when in start-up mode”
- Explicit formalization of design assumptions is very important
  - **These must be called out in documentation**
  - **These must be validated by domain experts!**
  - Example: Ariane V lateral velocity limits
Assessing Requirements Completeness

- DO178B uses testing as a measure of completeness
- Can we know when we have enough requirements (properties) through analysis?
- **This is a certification barrier for formal methods in avionics**
- Ideas:
  - Mutation checking [Chockler]
  - Functional input/output traces
  - Examination of Kripke Structures
- None of these approaches are currently practical
Getting to Nerdvana
Benefits of Formalizing Requirements

Level of Scale

Subsystem

System

System of Systems

Requirements

Completeness and consistency checking

Design / Code

Automated proof (model checking)

Partial analysis and static analysis (ASTREE, Coverity)

Test

Automated test generation from requirements

Test oracles for automated unit and integration test

Field

Runtime monitors for failure recovery

August, 2011  RE 2011: Mike Whalen  39
How we Will Develop Software

Concept Formation

Requirements

Properties

Model / Code

Object Code

Integration

Analysis

Analysis: Timing & Runtime Errors

Is model:
- correct w.r.t. functional requirements?
- free from unintended functionality?
Are requirements correct?

Does object code conform to specification model? Is it fast enough?

Use properties as test oracle

Autogenerated Tests

Specification Test

August, 2011     RE 2011: Mike Whalen
The #1 Rule of FITE Club… You talk about FITE Club!

The #8 Rule of FITE Club…

If this is the first time you hear about FITE Club…

you join the Club
Conclusions

- How we develop & verify software is going to change
- Possible to use powerful automation for **formal verification** and **creating tests**
- Focus swings back to **requirements**
  - Can be expressed temporal logic, JML, Spec#, code assertions, …
- Tools can already be used on substantial industrial models
  - In widespread use in hardware,
  - Increasing adoption in critical avionics, medical devices
- Coming to an IDE/build system near you in the near future!
Thank you!