Phase Two Feasibility Study for Software Safety Requirements Analysis Using Model Checking

NASA Formal Methods Symposium
April 13, 2010

Greg Turgeon
PPT Solutions, Inc.
Huntsville, Alabama
Agenda

• Introduction
• Overview of System
• Methodology
  – Stages of Analysis
  – Toolset
• Results
  – Defects
  – Assumptions
  – Safety Properties
• Return on Investment
• Ending Thoughts
Introduction
Introduction

• Overall goal is to utilize more objective techniques for analyzing software system safety
• Approach uses formal methods techniques and tools that are gaining acceptance in industry
• Study used model checking to analyze a sample system
• Phase 1 used a tool from Rockwell Collins
• Phase 2 used a COTS tool, SCADE
Phase 1 Conclusions

• Tool set was useful in detecting potential safety issues.
• Could not model entire system at once. Had to partition it.
• Training in MATLAB/Simulink and the Rockwell tools plus college level math necessary.
• 15% of SRS modeled in 1200 labor hours. Training and learning the tools was a significant portion of the hours.
Goals of Phase 2

• Examine other tool sets for scalability and ease of use
• Determine metrics for rate of development, assumptions per requirement, and defects per requirement
• Create training package to complement existing Simulink and Gryphon training
Overview of System
Description of Sample System

Manager 1
- Element A
- Element B
- Element C

Manager 2
- Element D
- Element E

Manager 3
- Element F
- Element G

Manager 4
- Element H
- Element I
- Element J

VMS

Other Managers

Inputs from and outputs to other subsystems

Subsystem Manager

Air Data System

FPCU

Air Data System
Methodology
Methodology - Lifecycle

<table>
<thead>
<tr>
<th>System Requirements Definition</th>
<th>System Design</th>
<th>System Implementation</th>
<th>System Verification</th>
</tr>
</thead>
</table>

Systems Safety Analysis | Systems Safety Requirements | Systems Safety Implementation Analysis and Verification

Safety Properties

Stage 1: Software Formal Methods Safety Analysis

Future work

Stage 2: Software Formal Methods Safety Analysis

Future work

System/Software Safety Findings

Safety Property Report

Software Requirements Document

Software Requirements Definition

Software Design Document

Software Design
Methodology – SCADE

Stage 1: Software Formal Methods Safety Analysis

1. Manually create model from requirements document
2. Write properties that verify functionality of model and that verify safety properties
3. Run SCADE’s automatic verification tool

Possible reasons for property being falsifiable:
1) Incorrect model
2) Incorrect property
3) Incorrect requirement in document

Stage 2 Analysis
Safety Analysis Tool Set
Portion of SCADE Model
Portion of SCADE Model – State Transitions

Diagram from the SRS

SCADE implementation
Obtain Result (automatic)
Obtain Result (automatic)
Results
Defects Detected

- Total Number of Requirements Modeled: 925 (82% of SRS)
  - Other 18% either didn’t add value to the model or were unrelated to software.

- Total Number of Requirements Defects Found: 198
  - Found through manual IV & V: 54 (27%)
  - Found through creation of model: 67 (34%)
  - Found as a result of model checking: 77 (39%)
  - Formal methods approach using SCADE method found 144 defects (73% of all defects found) traditional IV&V would miss.

Possible Defects:

- Model Checking: 39%
- Model Creation: 34%
- Manual IV&V: 27%
## Example Defects

<table>
<thead>
<tr>
<th>Method of Detection</th>
<th>Requirement and Defect</th>
</tr>
</thead>
</table>
| Manual IV&V             | *If in State A and Input I is true, go to State S.*  
*If in State A and Input I is true, go to State T.*  
Requirements conflict because states S and T are mutually exclusive. |
| Model Creation          | *If the VMS is Master and the other VMS fails, then it shall remain Master.*  
What if the VMS is Slave and the other VMS fails? No requirement for this situation. |
| Model Checking          | *If the Launch Abort Command input is true, then the launch abort sequence should begin.*  
Modeled property shows a counterexample where receiving a “Launch Abort Command” does not result in the software signaling to abort the launch sequence. |
### Assumptions

- Made 121 assumptions, or about 1 assumption for every 7.6 requirements.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Assumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>If the VMS is in State S and is the master, it shall send a synchronization message to the other VMS.</td>
<td>The VMSs are communicating.</td>
</tr>
<tr>
<td>Thirty-five seconds after power up, the VMS shall begin periodic communications with the other VMS on the datalink.</td>
<td>“Periodic” communications occur on every cycle.</td>
</tr>
<tr>
<td>When counting the number of times Event E occurs, the software shall not log more than the limit specified in the spreadsheet.</td>
<td>The name of the spreadsheet and where to find it are not indicated. Assumed that the limit was a constant.</td>
</tr>
<tr>
<td>The software shall read the RT address of its terminal to determine its own identity.</td>
<td>The RT address signal is latched.</td>
</tr>
</tbody>
</table>
Defect Criticality Breakdown

62.5% of all “Catastrophic” defects were found through Model Verification.

Defect Origin

- Catastrophic
- Critical
- Marginal
- Negligible

Manual IV&V

- Catastrophic: 8
- Critical: 34
- Marginal: 8
- Negligible: 4

Model Implementation

- Catastrophic: 5
- Critical: 21
- Marginal: 5

Model Verification

- Catastrophic: 7
- Critical: 40
- Marginal: 15
- Negligible: 15
Example Safety Properties

- Modeled 215 safety properties.
- 68 (32%) evaluated to false and returned a counterexample.

<table>
<thead>
<tr>
<th>Safety Property</th>
<th>Verification Operator Implemented As</th>
<th>Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>If the datalink is broken, can both VMSs be master?</td>
<td>After the first cycle, if VMS_A is not communicating and VMS_B is not communicating, then VMS_A is not master or VMS_B is not master.</td>
<td>false</td>
</tr>
<tr>
<td>An invalid VMS is declared halted.</td>
<td>If a VMS is not valid, then it is in the halted state.</td>
<td>false</td>
</tr>
<tr>
<td>If the engine oil temperature is out of range a fault will be set.</td>
<td>If the oil temp is less than -50 or greater than 110, then oilTempFault will be true.</td>
<td>false</td>
</tr>
<tr>
<td>Before the engine started message is sent, the engine start request message must be received.</td>
<td>Is it true that the signal engineStartComAccepted has never been true and the signal engineRunning is true?</td>
<td>false</td>
</tr>
</tbody>
</table>
Similar Results by Other Companies

Subtle Errors – LM Aero UAV Sensor Voting

OFP Triplex Voter
- 96 Simulink Subsystems
- 3 Stateflow Diagrams
- $6 \times 10^{13}$ Reachable States

Formal Verification
- 25 Informal Requirements
- 57 Formal Properties
- 40 Minutes to Analyze

Resulting In
- 24 Counterexamples
- 10 Design Modifications
- Several Requirements Clarifications

Formal verification has found subtle errors that would likely be missed by traditional testing.

- Lockheed Martin
Return on Investment
925 requirements modeled / 934 hours ≈ 1 requirement per hour
Example of the Frequency (%) of Where Errors Are Found, in Relationship to Where They Were Introduced

<table>
<thead>
<tr>
<th>Where Errors are Introduced (%)</th>
<th>Requirements Gathering and Analysis/Architectural Design</th>
<th>Coding/Unit Test</th>
<th>Integration and Component Test</th>
<th>Early Customer Feedback/Beta Test Programs</th>
<th>Post-product Release</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements Gathering and Analysis/Architectural Design</td>
<td>3.5 ($1x)</td>
<td>10.5 ($5x)</td>
<td>35 ($10x)</td>
<td>6 ($15x)</td>
<td>15 ($30x)</td>
<td>70</td>
</tr>
<tr>
<td>Coding/Unit Test</td>
<td>6 ($1x)</td>
<td>9 ($10x)</td>
<td>2 ($20x)</td>
<td>3 ($30x)</td>
<td></td>
<td>20</td>
</tr>
<tr>
<td>Integration and Component Test</td>
<td></td>
<td>6.5 ($1x)</td>
<td>1 ($10x)</td>
<td>2.5 ($20x)</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>Total</td>
<td>3.5</td>
<td>16.5</td>
<td>50.5</td>
<td>9</td>
<td>20.5</td>
<td>100 %</td>
</tr>
</tbody>
</table>

Taken directly from NIST Table 5-2 and 5-3
Return on Investment
Requirements Review Savings

• Assumptions
  – Entire Project: 45,550 man hours (based on known LOC)
  – Entire Project: 20% of lifecycle in requirements phase
  – Entire Project: 25% of requirements phase spent in review
  – Entire Project: Therefore, **2278** man hours in requirements review

• PPT spent **1376** total man hours on this effort
  – Or, 40% less time spent in requirements review

Assumptions based on metrics from *Introduction to the Team Software Process* by Watts S. Humphrey.
Return on Investment
Early Requirements Defect Removal

• We found 144 defects in the requirements phase that otherwise would not be found until later phases
• Effort to fix those 144 defects later in the lifecycle is approximately 3500 man hours
• Assuming a savings of 3500 man hours and $100 per man hour, early defect removal savings is $350K
• PPT cost to perform formal methods analysis: -$137K
• Net savings of $213K or 5% of the total project
Ending Thoughts
Conclusions

• Usability
  – SCADE is easy to learn and similar to Simulink

• Scalability
  – 80% + of requirements modeled
  – No problems with state space or execution time
  – No partitioning required

• Personnel requirements
  – Degree in a technical field
  – Training in SCADE

• Metrics
  – About 1 requirement modeled and verified per hour
  – Found about 1 defect per every 4.7 requirements
  – Made 1 assumption per every 7.6 requirements
Future Activities

• Use this technique more widely for IV&V
• Investigate SCADE for other uses throughout the software lifecycle
  – Automatic code generation
• Investigate other requirements specification methods (DOORS, UML)
• Apply technique to PLDs to realize cost savings and reduce rework
Sources


• Hands-on Modeling with the SAE AADL, Software Engineering Institute, course given October 5-6, 2009.

