Coverage Metrics and Requirements-Based Testing

Michael Whalen
Mats Heimdahl
University of Minnesota

AFRL S5 Workshop
6/17/2010
### Property-Based Software Engineering: Benefits of Formalizing Requirements

<table>
<thead>
<tr>
<th>Level of Scale</th>
<th>Requirements</th>
<th>Design / Code</th>
<th>Test</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subsystem</td>
<td>Completeness and consistency checking</td>
<td>Automated proof (model checking)</td>
<td>Automated test generation from requirements</td>
<td>Runtime monitors for failure recovery</td>
</tr>
<tr>
<td>System</td>
<td></td>
<td>Partial analysis and static analysis (ASTREE, Coverity)</td>
<td>Test oracles for automated unit and integration test</td>
<td></td>
</tr>
<tr>
<td>System of Systems</td>
<td></td>
<td></td>
<td></td>
<td>This Talk</td>
</tr>
</tbody>
</table>

This Talk
Software-Enabled Control

Software V&V Research:
- Model-Based Formal Methods
- Structural Testing
- Run-time Monitoring

Aerospace Research:
- V&V of flight control system and vehicle health management;
- discrete and continuous-time dynamics
Design & Development Space

Plant Description

NSF CPS Project

Software-Enabled Control

CONCRETE

ABSTRACT
IGERT (Pending)

Collaboration:
- Computer Science
- Aerospace
- Mechanical
- Civil
- Electrical
- Biomedical
- Human Factors

Human Centered Automation
Human Factors
Cognitive Science

Fluid and Aero Dynamics
Sensors

IGERT: Cyber Physical Systems—A Confluence of Human, Machine, and Physical Environment

Control Theory
Software Engineering
Validation & Verification
Presentation Overview

Overview

Automating the Testing Process

Structural Coverage Testing

…Of Requirements

Conclusions and Future Directions
Automated Test Case Generation

Requirements-Based Testing
- State Requirements as Properties
- Use Bounded Model Checker to Generate Test Cases
- Goal is to Cover the Requirement

Conformance Testing
- Autogenerate Test Cases From Model
- Commercial Tools Available
  - (T-VEC, REACTIS)
- Show Code Conforms to the Model
- Goal is Structural Coverage (e.g. MC/DC)

*Slide from “Proving the Shalls” by Steve Miller at Rockwell Collins, 2007*
Can I achieve structural coverage?

Engineer

Model or Code
(Simulink, SCADE, C, Java)

Automatic Translation

Tests OK

TCG Model
(T-VEC, Reactis, …)

Autotest Tool
(T-VEC, Reactis)

Failed Test Case

Properties
(CTL, LTL, PSL, …)

Automatic Check

Translation

Requirements
(Properties)

Test Cases

Tests OK

Can I achieve structural coverage?
Can structural coverage test metrics be rendered ineffective by restructuring a program?
Modified Condition/Decision Coverage (MC/DC)

- To satisfy MC/DC
  - Every basic condition in a decision in the model should take on all possible outcomes at least once, and
  - Each basic condition should be shown to independently affect the decision’s outcome

If (A or B) then stmt 1
    else stmt 2

\[ X = A or B \]
If (X) then stmt 1
    else stmt 2

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A or B</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

Independent effect of B

Independent effect of A
Non-Inline Implementation:
bool AndOr (bool a, bool b, bool c)
{
    bool local;
    local = b | c;
    return (a && local);
}

bool Compute (bool x, bool y, bool z,
        int temp, int thresh)
{
    bool run, no_danger, no_alarm;
    run = AndOr(x, y, z);
    no_danger = (temp <= thresh);
    if (no_danger) then no_alarm = true;
    else no_alarm = false;
    return (run && no_alarm) ;
}

Inline Implementation:
bool Compute (bool x, bool y, bool z,
        int temp, int thresh)
{
    bool no_alarm;
    if (temp <= thresh) then no_alarm = true;
    else no_alarm = false;
    return ((x && (y | z)) && no_alarm) ;
}
Defeating Structural Cov. Criteria

Version 1:
Non-Inlined Implementation

\[
\text{expr1} = \text{in1 or in2;}
\]
\[
\text{out1} = \text{expr1 and in3;}
\]

Version 2:
Inlined Implementation

\[
\text{out1} = (\text{in1 or in2}) \text{ and in3;}
\]

Observable variable:
out1
Defeating Structural Cov. Criteria

Version 1:
Non-Inlined Implementation

```plaintext
expr1 = in1 or in2;
out1 = expr1 and in3;
```

Version 2:
Inlined Implementation

```plaintext
out1 = (in1 or in2) and in3;
```

Any test suite satisfying MCDC in the inlined implementation will catch the operator fault.
Experiment – Fault Finding

**Hypothesis** – A test suite generated to provide MC/DC over an inlined implementation will reveal more faults than a test suite generated to provide MC/DC over a non-inlined implementation.

NonInlined Vs Inlined Fault Finding

NB: Current model-based testing metrics for Simulink and SCADE are equivalent to *non-inlined implementations*.
Adequacy of Black Box Test Suites

- Is your black-box test suite adequate?
- Current practice
  - Examine coverage on an executable artifact
Adequacy of Black Box Test Suites

- Problems with current practice
  - **Indirect measure**
    - Defects of omission in implementation not exposed.

- Executable artifact is necessary
  - adequacy can only be determined late in the development process
The Idea...

- Objective, implementation-independent measure of adequacy of a black-box test suite
- Objective assessment of completeness of high-level requirements (given an implementation)
- Potential for autogeneration of black-box test suites.

Temporal Logic

Synchronous Observers

State Machines

Write requirements in a formal notation, then define structural coverage metrics directly over requirements, to provide:

- Objective, implementation-independent measure of adequacy of a black-box test suite
- Objective assessment of completeness of high-level requirements (given an implementation)
- Potential for autogeneration of black-box test suites.
Formalizing Requirements

“If the onside FD cues are off, the onside FD cues shall be displayed when the AP is engaged”

\[ G(\neg \text{Onside\_FD\_On} \land \neg \text{Is\_AP\_Engaged}) \rightarrow X(\text{Is\_AP\_Engaged} \rightarrow \text{Onside\_FD\_On}) \]

• Coverage Metrics
  • Requirements coverage:
    • Single test case that demonstrates that req. is satisfied
    • Prone to “dumb” tests, e.g.: test where AP is never engaged.
  • Antecedent Coverage
    • Single test for requirement, but:
    • Ensure that antecedent of implication is satisfied
Unique First Cause (UFC) Coverage

“System shall eventually generate an Ack or a Time Out”

Req. LTL property - F(A ∨ B)

Formal UFC obligation for A:  ¬(A ∨ B) U (A ∧ ¬B)
for B:  ¬(A ∨ B) U (B ∧ ¬A)
Experiment: Fault Finding

Hypothesis 1 ($H_1$): Conformance tests providing requirements UFC coverage are more effective than conformance tests providing MC/DC coverage over the model.

Hypothesis 1 rejected at 5% statistical significance on all but the Latctl system.
Why Was Fault Finding So Poor on DWM2?

Given:

\[
\text{LTLSPEC G(var_a > (}
\begin{array}{l}
\text{case}
\end{array}
\begin{array}{l}
\text{foo : 0 ;}
\text{bar : 1 ;}
\text{esac +}
\end{array}
\text{case}
\begin{array}{l}
\text{baz : 2 ;}
\text{bpr : 3 ;}
\text{esac})}\
\end{array}
\text{));}
\]

...yields 1 test obligation

Equivalent to:

\[
\text{LTLSPEC G(}
\begin{array}{l}
\text{case}
\end{array}
\begin{array}{l}
\text{foo & baz : var_a > 0 + 2 ;}
\text{foo & bpr : var_a > 0 + 3 ;}
\text{bar & baz : var_a > 1 + 2 ;}
\text{bar & bpr : var_a > 1 + 3 ;}
\text{esac}
\end{array}
\text{));}
\]

...yields 32 test obligations

Requirements-based test metrics are significantly impacted by requirements structure
Experiment: Combining Tests

Hypothesis 2(H2): Conformance tests providing requirements UFC coverage in addition to MC/DC over the model are more effective than conformance tests providing only MC/DC over the model

Hypothesis 2 accepted at 5% statistical significance on 3 of 4 systems
Current and Future Work

- **Transformation-Insensitive Structural Coverage Metrics: Observable MCDC,**
  - MCDC focuses on *masking* of conditions
  - Observable MCDC requires no masking along some path to *observable* variable (usually output)

- **Composable Formal Methods**
  - Looking at asynchrony / multiple time scales
  - Combining artifacts from different tools
  - Integration with Plant Models

- **Process Issues**
  - How Many Properties Do I Need?
    - When am I done?
    - Are there coverage metrics like there are for testing?
    - How do I convince the certification authorities I’m done?
Thank You!
Testing:

Formal Methods / Model Checking:
Backup
Can I achieve structural coverage?

Engineer

Verification – Automated Testing

Model or Code
(Simulink, SCADE, C, Java)

TCG Model
(T-VEC, Reactis, …)

Automatic Translation

Failed Test Case

Autotest Tool
(T-VEC, Reactis)

Properties
(CTL, LTL, PSL, …)

Requirements
(Properties)

Tests OK

Test Cases

Automated Check

Can I achieve structural coverage?

The Idea (by example…)

- Empty Sudoku board has $9^{81} \approx 10^{77}$ possible states
  - After filling in given numbers for a Sudoku board (usually 19-22 squares filled), fewer states: $\approx 10^{55}$
  - Valid board with given numbers has exactly 1 solution

- What are Sudoku requirements?
  - Must have exactly one of 1-9 in each row, column, and 3x3 square.
  - Equivalently, a “bad” non-solution will have two (or no) elements along one of these vectors

- Can we trick a model checker into solving Sudoku?
  - State that all solutions for a set of givens are “bad”
  - Model checker will demonstrate that this property is false and provide a counterexample (solution)
Sudoku encoding in SAL

sudoku2: CONTEXT =
BEGIN
  BLOCK : TYPE = [ 0 .. 2 ] ;
  BLOCK_RANGE : TYPE = [ 1 .. 3 ] ;
  RANGE : TYPE = [ 1 .. 9 ] ;
  BOARD : TYPE = array RANGE of array RANGE of RANGE ;

sudoku : MODULE =
BEGIN
  INPUT b : BOARD
  OUTPUT good : BOOLEAN
  DEFINITION
  good =
    % good columns
    (FORALL (x: RANGE, y1: RANGE, y2: RANGE) :
      (y1 /= y2) => (b[x][y1] /= b[x][y2])) AND
    % good rows
    (FORALL (y: RANGE, x1: RANGE, x2: RANGE) :
      (x1 /= x2) => (b[x1][y] /= b[x2][y])) AND
    % good 3x3 blocks
    (FORALL (xb: BLOCK, yb: BLOCK,
      x1: BLOCK_RANGE, y1: BLOCK_RANGE,
      x2: BLOCK_RANGE, y2: BLOCK_RANGE) :
      ((x1 /= x2) OR (y1 /= y2)) =>
      b[3*xb + x1][3*yb + y1] /=
      b[3*xb + x2][3*yb + y2])

END ;

% generate a board that meets the
% sudoku constraints
test_th: THEOREM sudoku |-
  G(not good);

% solve a specific ‘hard’ sudoku
% according to wikipedia

%F

\[
\begin{array}{cccc}
  & b[6][2] = 3 & b[8][2] = 8 & b[9][2] = 5 \\
b[5][9] = 4 & b[9][9] = 9 & &
\end{array}
\]

END ;
Errors are always made during development
~ 1 error / 1,000 SLOC for CMM 5.

Testing can be used everywhere
... it verifies the final product
... but isn’t very good at finding errors.

Model-checking is very good at finding errors
... early in the design process
... but it doesn’t work everywhere.

Use model-checking where it works now
... technology is improving rapidly and will be even better in the future.

But some testing will always be necessary.
CerTA FCS Phase II

- **Sponsored by the AFRL - Wright Patterson VA Directorate**
- Can Model-Checking be Used on Large, Non-linear Systems?
  - Lockheed Martin Adaptive UAV Flight Control System
  - Extensive Use of matrix arithmetic
  - Inputs – 33 floating point inputs (including one 3 x 6 matrix)
  - Outputs – 6 floating point values
  - 166 Simulink subsystems
  - 2000+ basic Simulink blocks
  - Translated to Prover model checker

- **Challenges**
  - Verification of floating point matrix arithmetic
  - Verification of Stateflow flowcharts with cycles
  - Compositional Verification

- **Final Results**
  - Identified five previously unknown errors
  - Identified several implementation errors that were being masked by defensive programming
Verification – Theorem Proving

- Available Since the 1980’s
  - Heavily used on security systems

- Use Rules of Inference to Prove New Properties
  - Also consider all combinations of inputs and states
  - Equivalent to exhaustive testing of the model
  - Generate an unprovable proof obligation if a property is false

- Not Limited by State Space
  - Applicable to almost any formal specification

- Limitations
  - Skill of user – about six months to become proficient
  - Time - constructing proofs is labor intensive
Does the system have property X?

Yes!

Engineer

Requirements
(Properties)

Analysis Model
(PVS, ACL2, HOL, …)

Theorem Prover
(PVS, ACL2, HOL, …)

Automatic Translation

Properties
(Lemmas, Theorems, …)

Why not?

Translation

Guru

Requirements as Shall Statements

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.

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.

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.
Encoding Requirements as Properties

If the mode is COOKING, then the microwave door shall be CLOSED

SPEC AG(mode = COOKING -> door_closed) ;

assert (!(mode == COOKING) || door_closed);
### 1 Mode Annunciations

#### 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 onboard FD is turned 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.

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

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 onboard FD is on, or the off-side FD is on, or the AP is engaged.

#### 1.3 Operation

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

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

SPEC AG((Mode_Annunciations_On & IsThis_Side_Active = 1 & Onside_FD_On) => Mode_Annunciations_On))

SPEC AG((Mode_Annunciations_On & Offside_FD_On = FALSE) => AX((IsThis_Side_Active = 1 & Offside_FD_On = TRUE) => Mode_Annunciations_On))

SPEC AG((Mode_Annunciations_On & IsThis_Side_Active = 1 & Onside_FD_On) => Mode_Annunciations_On))

SPEC AG(Mode_Annunciations_On => AX((IsThis_Side_Active = 1 & !Onside_FD_On & Offside_FD_On = FALSE & !Is_AP_Engaged) => !Mode_Annunciations_On)))

SPEC AG(Mode_Annunciations_On => AX((IsThis_Side_Active = 1 & !Onside_FD_On & Offside_FD_On = TRUE & !Is_AP_Engaged) => !Mode_Annunciations_On)))

SPEC (Mode_Annunciations_On)

SPEC AG((IsThis_Side_Active = 1) => (Mode_Annunciations_On <= (Onside_FD_On | Offside_FD_On = TRUE & !Is_AP_Engaged)))
Presentation Overview

Overview

Formalizing Requirements

Formal Methods

Testing

Challenges and Future Directions
Rockwell Collins Translation Framework

- Support a wide variety of back end tools
  - Very straightforward to add new tools
  - E.g. Prover support: 4 days effort
- Allows “the right tool for the right job”
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
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

<table>
<thead>
<tr>
<th></th>
<th>Subsystem/Blocks</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>$6 \times 10^{13}$</td>
<td>48</td>
</tr>
<tr>
<td>Failure processing</td>
<td>7 / 42</td>
<td>0 / 0 / 0</td>
<td>$2.1 \times 10^{4}$</td>
<td>6</td>
</tr>
<tr>
<td>Reset manager</td>
<td>6 / 31</td>
<td>2 / 26 / 0</td>
<td>$1.32 \times 10^{11}$</td>
<td>8</td>
</tr>
<tr>
<td>Totals</td>
<td>23 / 169</td>
<td>5 / 61 / 198</td>
<td>N/A</td>
<td>62</td>
</tr>
</tbody>
</table>

... for each of ten control surfaces

Phase I Results

<table>
<thead>
<tr>
<th></th>
<th>Effort (% total)</th>
<th>Errors Found</th>
</tr>
</thead>
<tbody>
<tr>
<td>Testing</td>
<td>60%</td>
<td>0</td>
</tr>
<tr>
<td>Model-Checking</td>
<td>40%</td>
<td>12</td>
</tr>
</tbody>
</table>
Verification – Automated Test Generation

- Available Since the 1990s
  - Supported by several commercial tools
  - T-VEC, Reactive Systems, LDRA, The Mathworks, etc.

- Generate tests up to some level of coverage
  - In Avionics: MCDC is common
  - Not sound (no guarantee all errors found)!

- Not Limited by State Space
  - Applicable to source code, object code, or design models
  - Low cost to generate thousands or millions of tests

- Limitations
  - No oracle! Nothing to describe whether test “passes”.
    - Properties can define notion of correctness
  - May be unable to generate all tests for given level of coverage
  - (Again) not a proof! Only possible to refute properties.
Acknowledgements

- University of Minnesota (Dr. Mats P. E. Heimdahl)
- NASA Langley Research Center (Ricky Butler)
- Air Force Research Labs (RD Directorate)
- Lockheed Martin (Walter Storm)
- Rockwell Collins (Dr. Steven Miller, Dr. Darren Cofer, Lucas Wagner)