An Industry Proof-of Concept Demonstration of Automated Combinatorial Test

Redge Bartholomew
Software Defects Drive Development Cost

...Especially for safety-critical, embedded systems

- NIST/NASA study looked at 15 years of defects
  - Avionics; medical, space, security systems; servers, browsers
- 6-way combinatorial testing could detect most of them
- NIST/UT-Arlington created a toolset to automate this
- Industry proof-of-concept experiment used them
NIST/UT-Arlington Approach

Tool generates minimum set of input test vectors,

User provides test vector generator with input variable definitions & values

... test harness imports, executes, analyzes test cases; identifies failures; measures coverage.

... model checker determines expected outputs for each vector,

NuSMV

... script merges input vectors with their expected outputs,

User provides model of inputs, outputs, properties
Proof-of-Concept Experiment

... reduce the number of defects escaping into system test

- Assess potential cost-effectiveness for unit, integration test
  - Accuracy, coverage, scalability, maturity, ease of learning/using
- Verify parts of a software defined radio (C++, 196 KSLOC)
- Inputs/outputs defined from reverse engineering code
Proof-of-Concept Experiment

... expected output generation deviated from NIST approach

Tool generates minimum set of input test vectors;

User provides test vector generator with input variable definitions & values

... utility finds states containing input vectors, exports them as test cases;

... test harness imports, executes, analyzes test cases; identifies failures; measures coverage.

ACTS

NuSMV

User provides model of inputs, outputs, properties

VectorCAST
Defining the Input Space

... minimizing the vector set

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>256</td>
<td>false</td>
<td>-1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>255</td>
<td>true</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>256</td>
<td>true</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td>255</td>
<td>true</td>
<td>-1</td>
</tr>
<tr>
<td>5</td>
<td>15</td>
<td>256</td>
<td>false</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>15</td>
<td>255</td>
<td>false</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>16</td>
<td>256</td>
<td>true</td>
<td>-1</td>
</tr>
<tr>
<td>8</td>
<td>16</td>
<td>255</td>
<td>false</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>16</td>
<td>256</td>
<td>true</td>
<td>1</td>
</tr>
</tbody>
</table>

For variables a, b, c, d, ACTS condenses all 37 2-way combinations of input values into 9 test vectors.
Generating Expected Outputs

Developer models input-output relationships,

```
MODULE main
VAR
  a : {0,15,16};
  b : {255,256};
  c : {true,false};
  d : {-1,0,1};

DEFINE
  e :=
    case
      ... when c = true, e = a + b,
        (c = true) : a + b;
    TRUE : a * d;
    esac;
  ... otherwise, e = a * d
```

... the model checker generates the state space

```
d = -1
------- State 30 -------
  e = -15
  a = 15
  b = 255
  c = false
  d = -1
------- State 31 -------
  e = 271
  a = 15
  b = 256
  c = true
  d = -1
```
Defining the Input Space
... to maximize defect detection & structural coverage

- More input values = better defect detection & coverage
  - ... and greater risk of combinatorial explosion

- Equivalence classes can reduce the number of values

- Interaction testing can reduce the number of variables
  - Test only those that interact; others set to default values
Verifying the Expected Outputs

... debugging the model – e.g., coding & requirements errors

• 50 test vectors arbitrarily selected for manual verification.

• Model checker compared output model vs. properties
  – After gaining familiarity with tool, language, notation

• False positive/negative detection of seeded defects
  – Initially, most traced back to bugs in the expected output model
Accuracy & Scalability – Unit Test

... number of Input variables & values

- Radio control loop for normal mode: 11 inputs, 15 outputs
- Seeded 100+ errors in code across several versions
- Required interaction testing (avoid combinatorial explosion)
- 47,040 tests generated, executed, analyzed in ~2.6 hrs.
  - All defects detected; 98% branch coverage (~2 hrs.)
Accuracy & Scalability – Unit Test

... complex logic

- Mode control: 34 inputs, 4 outputs of interest
  - 39 if-tests nested up to 8-levels deep, embedded in a 6-case switch
  - 200+ seeded defects

- Used several sets of interaction tests (largest = 19 inputs)

- 2775 tests generated, executed, analyzed in ~1hour
  - All defects detected; 95% MC/DC required ~16 hours
Modified Condition/Decision Coverage

• Independence of outcome required writing test stubs
  
  – … to force the sequence of input vectors in multi-conditionals
  
  – … e.g., in loops that modify the control variable

```
while ((current_radio_state != APPLICATIONS_RUNNING) &&
      (current_radio_state != NO_WAVEFORM_AVAILABLE))
{
    current_radio_state = Radio_State();
}
```
Accuracy & Scalability - Integration Test

• Tested code unit via 6 levels of nested procedure calls
  – Intervening variables set to force execution along required path

• 30,905 tests detected seeded defects
  – Finding intervening variables (30+) & default values: several days

• Scalability for large/nested architecture requires mitigation
Ease of Learning & Using – 1st Test

... 1st test (complex logic): ~84 hrs. total vs. 80 hr. target

• Staff had no prior experience with tools or code
• Learning ACTS (~2 hrs.) + defining 34 inputs (~4 hrs.)
• Learning NuSMV (~20 hrs.) + Debugging model (~12 hrs.)
• Structural coverage: branch (~2 hrs.)
  – MC/DC: another 16 hrs. (includes defining/constructing stubs)
Maturity
... adapted from DoD/NASA Technology Readiness Calculator*

- Level 7
  - Prototype software exists & all key functionality is available for demonstration/test
  - It is well integrated with operational systems
  - Operational feasibility has been demonstrated
  - Most software bugs have been eliminated
  - Some documentation is available

<table>
<thead>
<tr>
<th>Technology Readiness Levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Basic principles observed</td>
</tr>
<tr>
<td>2. Concept/application formulated</td>
</tr>
<tr>
<td>3. Experimental proof-of-concept</td>
</tr>
<tr>
<td>4. Validation in laboratory</td>
</tr>
<tr>
<td>5. Validation in a relevant environment</td>
</tr>
<tr>
<td>6. Prototype demonstration</td>
</tr>
<tr>
<td>7. <strong>Prototype demonstrated in [operational environment]</strong></td>
</tr>
<tr>
<td>8. [Operationally] qualified via test</td>
</tr>
<tr>
<td>9. Proven via mission operations</td>
</tr>
</tbody>
</table>

Summary

... effective enough for unit test

- Significant defect detection, coverage from moderate effort
- May play a role in integration test
- Finding equivalence class values requires experience
  - So does using model checker properties to find expected outputs
- Deployment issues: support, packaging, licensing, training
Acronyms

• ACTS – Advanced Combinatorial Testing System
• DoD – Department of Defense
• GAO – Government Accountability Office
• KSLOC – 1000’s of Source Lines of Code
• MC/DC – Modified Condition/Decision Coverage
• NASA – National Aeronautics and Space Administration
• NIST – National Institute of Standards and Technology
• UT-Arlington – University of Texas at Arlington