Testing Complex Safety-Related Systems

Mike Bartley
TVS, Founder and CEO
Agenda

- Some background on your speaker and testing safety related systems
- Can directed testing scale with complex systems?
- Ensuring new techniques fit within safety standards
Your speaker: Mike Bartley

- PhD in Mathematical Logic
- MSc in Software Engineering
- MBA

- Worked in software testing and hardware verification for over 25 years
  - Praxis, IPL, ST-Micro, Infineon, Panasonic, ARM, NXP, nVidia, ClearSpeed, Gnodal, DisplayLink, Dialog, ...
  - Worked in formal verification of both software and hardware

- Started TVS in 2008
  - Software testing and hardware verification products and services
  - Offices in India, UK, France and Germany
TVS - Global Leaders in Test and Verification

Continuous geographical expansion…

Number of Employees by quarter

<table>
<thead>
<tr>
<th>EMPLOYEES</th>
<th>Q3-13</th>
<th>Q4-13</th>
<th>Q1-14</th>
<th>Q2-14</th>
<th>Q3-14</th>
<th>Q4-14</th>
<th>Q1-15</th>
<th>Q2-15</th>
<th>Q3-15</th>
<th>Q4-15</th>
<th>Q1-16 (Est.)</th>
<th>Q2-16 (Est.)</th>
<th>Q3-16 (Est.)</th>
<th>Q4-16 (Est.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALENDAR YEAR</td>
<td>Q3-13</td>
<td>Q4-13</td>
<td>Q1-14</td>
<td>Q2-14</td>
<td>Q3-14</td>
<td>Q4-14</td>
<td>Q1-15</td>
<td>Q2-15</td>
<td>Q3-15</td>
<td>Q4-15</td>
<td>Q1-16 (Est.)</td>
<td>Q2-16 (Est.)</td>
<td>Q3-16 (Est.)</td>
<td>Q4-16 (Est.)</td>
</tr>
</tbody>
</table>
Safety Standards

- **IEC61508**: Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems
- **DO178C**: Software considerations in airborne systems and equipment certification
- **EN50128**: Software for railway control and protection systems
- **IEC60880**: Software aspects for computer-based systems performing category A functions
- **IEC62304**: Medical device software -- Software life cycle processes
- **ISO26262**: Road vehicles – Functional safety
Safety-Critical: Verification and Testing

- Depend on integrity level/class
- Software Verification Plan
- Reviews of Specifications and Code
- Testing (against specifications)
  - Unit
  - Software Integration
  - Software System
- Test Coverage Criteria
- Requirements and Test Traceability
- Independence
- Audit trails
## Dynamic analysis and testing

<table>
<thead>
<tr>
<th>Technique</th>
<th>SIL 1</th>
<th>SIL 2</th>
<th>SIL 3</th>
<th>SIL 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structural test coverage (entry points) 100%</td>
<td>HR</td>
<td>HR</td>
<td>HR</td>
<td>HR</td>
</tr>
<tr>
<td>Structural test coverage (statements) 100%</td>
<td>R</td>
<td>HR</td>
<td>HR</td>
<td>HR</td>
</tr>
<tr>
<td>Structural test coverage (branches) 100%</td>
<td>R</td>
<td>R</td>
<td>HR</td>
<td>HR</td>
</tr>
<tr>
<td>Structural test coverage (conditions, MC/DC) 100%</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>HR</td>
</tr>
<tr>
<td>Test case execution from boundary value analysis</td>
<td>R</td>
<td>HR</td>
<td>HR</td>
<td>HR</td>
</tr>
<tr>
<td>Test case execution from error guessing</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>Test case execution from error seeding</td>
<td>-</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>Test case execution from model-based test case generation</td>
<td>R</td>
<td>R</td>
<td>HR</td>
<td>HR</td>
</tr>
<tr>
<td>Performance modelling</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>HR</td>
</tr>
<tr>
<td>Equivalence classes and input partition testing</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>HR</td>
</tr>
</tbody>
</table>
All the Right Stages but Not Necessarily in the Right Order

- Software Requirements
- High Level Design
- Unit Level Design
- Coding
- Integration Testing
- Unit Testing
- Software System Testing
Shift-Left “Sequential” Development Flow

1. Product Reqs → Requirements Verif Spec → Acceptance Verif
2. System Spec → System Verif Spec → System Verif
3. Unit Spec → Unit Verif Spec → Integration Verif Spec → Integration Verif
4. Unit Build → Static Analysis
5. Unit Verif Spec → Unit Verif
6. Requirements Verif Spec → System Verif Spec
The Unit Test Foundation

Integration of Pre-tested Components
How do we do the system level testing?

- Robotic vacuum cleaner?
- Mars rover?
- Drone landing on ship in rough seas?
- Automotive
  - Automated parking?
  - Lane keeping assistance?
  - Driverless cars

- Do we continue to perform directed testing?
  - Hardware verification faced the same issue 20 years ago
  - Hardware adopted constrained random verification
  - And ensure requirements tracing for safety standards compliance
The mechanics of an advanced test bench

Test

Driver

Functional Coverage

Coverage

Checker

Monitor

Assertions

Stimulus generator

constraint

addr

data

Active

Passive

Design Under Test

assert

Code Coverage
Some hardware verification examples

CPU Verification

Instruction Stream Generator → assembler → CPU RTL → Compare

CPU C Model

Accuracy?
Some hardware verification examples

USB Verification

- Packet Generator
- Driver
- DUT
- Scoreboard
- Response

- Assertions
- Coverage
- Assertions
- Coverage
Bubble Sort “Proof of Concept” for SW Testing

• **Program Specification**
  – Input lists of integers, floats, ascii, etc.
  – Reject lists of mixed types
  – Convert unsorted lists to sorted lists

• **Can we test the program with constrained input generation?**
  – Generate valid and invalid inputs
  – Direct generation towards corner cases
  – Check outputs for correctness
    • Without re-writing an identical checker program
  – Measure what we have tested
Results of Bubble Sort “Proof of Concept”

Lists of
- Integers
- Floats
- Ascii
- etc

List Generator

Checkers
- Check output list is ordered
- Output list contents == input list contents

Software Under Test

Coverage Metrics
- Empty List
- Reverse ordered
- Error cases (mix integers, floats, ascii)
- Etc.

Lists

Constrain towards
- Empty lists
- Equal values
- Reverse ordering
Virtual System Level Checkers

- Assert “never do anything wrong”
  - Always fail safe

- Assert “always respond correctly”
  - If A&B&C occur then check X happens
    - Assertion coverage “check A&B&C occurs” for free

- Analyse log files
  - Look for anomalies
    - Did the actuator outputs occur in the correct order
Functional Coverage

From Kerstin Eder of the University of Bristol

- Requirements coverage
- “Cross-product” coverage


A cross-product coverage model is composed of the following parts:
1. A semantic description of the model (story)
2. A list of the attributes mentioned in the story
3. A set of all the possible values for each attribute (the attribute value domains)
4. A list of restrictions on the legal combinations in the cross-product of attribute values

A functional coverage space is defined as the Cartesian product over the attribute value domains.

- Situation coverage

Safety compliance (asureSIGN)

- Managing Requirements
  - Importing and editing requirements

- Decomposing requirements to verification goals

- Tracking test execution
  - Automating import of test results
  - Automate accumulation and aggregation of test results

- Impact analysis
  - Managing changes in requirements and tests

- Demonstrating compliance to DO254 & DO178C

- Managing multiple projects
asuresIGN™ at the heart of HW/SW V&V

Requirements
- Excel
- Doors
- Jira
- etc

XML API

SystemC Simulation

Hardware Simulation
- Coverage
- Assertions
- Cadence
- Mentor, Aldec
- Etc.

UCIS API

Directed test results

Manual API

Automated SW Test Tool

Run API

Formal Verification
- OneSpin

Lab Results

Matlab

SW Test Tools

Requirements Engineering tools
Decomposing requirements to features and tests

Import Requirements (Doors, Excel, Word, etc)

Edit Requirements

Map requirements to verification goals

The mapped verification goal

Sign off a requirement with a manual test (e.g. in the lab)
Safety compliance (asureSIGN)

- Managing Requirements
  - Importing and editing requirements

- Decomposing requirements to verification goals

- Tracking test execution
  - Automating import of test results
  - Automate accumulation and aggregation of test results

- Impact analysis
  - Managing changes in requirements and tests

- Demonstrating compliance to DO254 & DO178C

- Managing multiple projects
Tracking test execution:
Automating import of test results

Accellera standard

ASURESIGN

UCIS

XML

API

Manual Entry

Manual Testing

Software Test Tools

Formal Verification

Hardware Simulation

Hardware Verification Results

Hardware Simulation

Capturing test execution:
Automating import of test results

Accellera standard

ASURESIGN

UCIS

XML

API

Manual Entry

Manual Testing

Software Test Tools

Formal Verification

Hardware Simulation

Hardware Verification Results
Automate accumulation and aggregation of test results

- Record results from each test
- Accumulate results over multiple regressions
- Aggregate results through the hierarchy
- Define and track against interim milestones (based on % of requirements tested)
Safety compliance (asureSIGN)

- Managing Requirements
  - Importing and editing requirements
- Decomposing requirements to verification goals
- Tracking test execution
  - Automating import of test results
  - Automate accumulation and aggregation of test results
- Impact analysis
  - Managing changes in requirements and tests
- Demonstrating safety compliance – for example
  - DO254/178C, ISO26262, IEC 60601, IEC 61508, EN 50128, IEC 61513
- Managing multiple projects
Demonstrating compliance to DO254 & DO178C

- Export XML for import back into Doors, etc.
- Export PDF report for audit

Select level of detail

Pid = unique reference to requirement in external tool

Export Metadata such as
- Tool version numbers
- Configuration data
- Data owners
Summary

- Current system testing techniques will not scale with new complex systems

- Lessons from hardware verification
  - Constrained random inputs
  - Automated checking
  - Functional coverage

- Ensuring safety compliance
  - Requirements driven testing

- Questions