Reduce Medical Device Compliance Costs with Best Practices

mark.pitchford@ldra.com
Agenda

• Medical Software Certification
  – How new is Critical Software Certification?
  – What do we need to do?
  – What Best Practises will help us achieve Certification?

• Questions & Answers
CRITICAL SOFTWARE CERTIFICATION

HOW NEW IS IT?
Where is certification enforced?

Whenever the cost of failure is very high
- Risk of death or injury
- High cost of repair
- High cost of product recall

What software needs to be certified?

- Aircraft
- Nuclear Power Stations
- Trains
- Cars
- Medical Devices
- Industrial Plants
### Leading Safety Critical Standards

<table>
<thead>
<tr>
<th>Sector</th>
<th>Standard</th>
<th>First Published</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avionics</td>
<td>DO-178B (First published 1992) / DO-178C</td>
<td></td>
</tr>
<tr>
<td>Industrial</td>
<td>IEC 61508 (First published 1998)</td>
<td></td>
</tr>
<tr>
<td>Railway</td>
<td>CENELEC EN 50128 (First published 2001)</td>
<td></td>
</tr>
<tr>
<td>Nuclear</td>
<td>IEC 61513 (First published 2001)</td>
<td></td>
</tr>
<tr>
<td>Automotive</td>
<td>ISO/DIS 26262 (Draft)</td>
<td></td>
</tr>
<tr>
<td>Medical</td>
<td>IEC 62304 (First published 2006)</td>
<td></td>
</tr>
<tr>
<td>Process</td>
<td>IEC 61511 (First published 2003)</td>
<td></td>
</tr>
</tbody>
</table>

So, the experience of other sectors is invaluable to the medical device (and automotive) industries.
IEC 62304 AND RELATED IEC 61508 DERIVATIVES
### Safety Integrity Levels

<table>
<thead>
<tr>
<th>Standard</th>
<th>Levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEC 61508 (Industrial)</td>
<td>SIL Level 1 to 4</td>
</tr>
<tr>
<td>ISO/DIS 26262 (Automotive)</td>
<td>ASIL A to ASIL D</td>
</tr>
<tr>
<td>IEC 62304 (Medical)</td>
<td>Class A to Class C</td>
</tr>
<tr>
<td>CENELEC EN 50128 (Railway)</td>
<td>SIL Level 0 to SIL Level 4</td>
</tr>
<tr>
<td>DO-178B / DO-178C (Avionics)</td>
<td>Level E to Level A</td>
</tr>
</tbody>
</table>

So, nothing new here either!
### Functional Safety Assessment

Classes A – C in IEC 62304 are based on the principle of IEC 61508’s SIL levels ...

<table>
<thead>
<tr>
<th>Minimum Level of Independence</th>
<th>Safety Integrity Level</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Independent Person</td>
<td>HR</td>
</tr>
<tr>
<td>Independent Department</td>
<td>-</td>
</tr>
<tr>
<td>Independent Organization</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 2: Assessment independence level for E/E/PE and software life cycle activities

(E/E/PE) : Electrical / Electronic / Programmable Electronic systems
IEC 61508 based standards are

- Primarily process oriented
- Includes Verification and Validation (V&V) guidelines for that process

IEC 61508 based standards define the need for

- Software requirements
- The safety lifecycle for software,
- Validation and verification appropriate for each SIL (or class)

IEC 61508 based standards require V&V activities including:

- Verification of code
- Software module testing
- Software integration testing

Best practices to achieve these aims are long established elsewhere and so can easily be adopted by the medical devices industry in meeting IEC 62304
CRITICAL SOFTWARE CERTIFICATION

WHAT DO WE NEED TO DO?
IEC 62304 : Common Framework

- The set of processes, activities, and tasks described in this standard establishes a common framework for medical device software life cycle processes.
IEC62304 : Clause 5

• IEC 62304 Clause 5 details the software development process of the product. It specifically addresses:

<table>
<thead>
<tr>
<th>Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1 Software development planning</td>
</tr>
<tr>
<td>5.2 Software requirements analysis</td>
</tr>
<tr>
<td>5.3 Software architectural design</td>
</tr>
<tr>
<td>5.4 Software detailed design</td>
</tr>
<tr>
<td>5.5 Software unit implementation and verification</td>
</tr>
<tr>
<td>5.6 Software integration and integration testing</td>
</tr>
<tr>
<td>5.7 Software system testing</td>
</tr>
<tr>
<td>5.8 Software release</td>
</tr>
</tbody>
</table>
IEC62304 : Clause 5.3

- IEC 62304 Clause 5.3 details the Software architectural design:

<table>
<thead>
<tr>
<th>5.3 Software architectural design</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5.3.1 Transform software requirements into an ARCHITECTURE</td>
<td></td>
</tr>
<tr>
<td>5.3.2 Develop an ARCHITECTURE for the interfaces of SOFTWARE ITEMS</td>
<td></td>
</tr>
<tr>
<td>5.3.3 Specify functional and performance requirements of SOUP item</td>
<td></td>
</tr>
<tr>
<td>5.3.4 Specify SYSTEM hardware and software required by SOUP item</td>
<td></td>
</tr>
<tr>
<td>5.3.5 Identify segregation necessary for RISK CONTROL</td>
<td></td>
</tr>
<tr>
<td>5.3.6 Verify software ARCHITECTURE</td>
<td></td>
</tr>
</tbody>
</table>

SOUP = Software Of Unknown Pedigree
Safety Integrity Levels

• The IEC 62304 standard expects the manufacturer to assign a safety class to the software system as a whole
• This classification is based on the potential to create a hazard that could result in an injury to the user, the patient or other people
• There are three software classes:

<table>
<thead>
<tr>
<th>Class</th>
<th>Failure Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>No injury or damage to health is possible</td>
</tr>
<tr>
<td>B</td>
<td>Non serious injury is possible</td>
</tr>
<tr>
<td>C</td>
<td>Death or serious injury is possible</td>
</tr>
</tbody>
</table>
Impact of Software Safety Classification

- The safety classification has a significant impact on the software development life cycle
MEDICAL SOFTWARE CERTIFICATION

WHAT BEST PRACTICES SHOULD WE APPLY?
Recommended Best Practices

• Requirements
  – Trace Requirements

• Static Analysis
  – Coding Standard
  – Check Complexity
  – Control Flow Analysis
  – Data Flow Analysis

• Dynamic Analysis & Unit Testing
  – Structural Coverage

• Test independence
Avoid the Requirement Gap

• Process must be “right weight”
  – Not too heavy, not too light
  – Help rather than hinder
  – No bias to particular disciplines or phases

• Focus on requirements
  – Don’t ignore them once construction begins
  – Implement what the stakeholder wants

• Manage requirements
  – Continually refine
  – Apply quality criteria

• Trace requirements
Requirements Drive Development

**Manage requirements; assign verification & debug tasks**

**Map requirements to design and source code**

---

**Software Requirements & Defect Reports**

**Project Managers**

**Requires Traceability Matrix (RTM)**

**Test Cases**

**Test Engineers**

**Verifies requirements against test cases**

**Implement requirements & verify design**

**Model or Design Specification**

**Software Engineers**

**Code Base**

**Development & Build Engineers**
Traceability Across Development Tiers

- **Tier 1**: High-Level Requirements
- **Tier 2**: Modelling Tool, Formal Methods, Software Specs
- **Tier 3**: Implementation (Source Code / Assembly)
- **Tier 4**: Host Tier (Node 1 – n)
- **Tier 5**: Target Tier (Node 1 – n)

- **Design Review Defects**
- **Code & Quality Review Defects**

- LL Reqs to HL Reqs
- Code to LL Reqs
- Test Results & Defects

Test Cases to LL Reqs
Requirements Traceability

SVP  LDRA Verification Plan
- Populated_project
  - Subprojects
    - Billcode (Developer)
      - Tbench
        - Tunnel I/O - Unit Test
          - Unit Test for LL_1 (Tunnel I/O - Unit Test)
          - Unit Test for REQ_0002 (Tunnel I/O - Unit Test)
        - Tunnel Power Failure - Code Review
          - Code Review for LL_2 (Tunnel Power Failure - Code Review)
          - Code Review for REQ_0009 (Tunnel Power Failure - Code Review)
        - Tunnel I/O - System Test
          - System Test for REQ_0016 (Tunnel I/O - System Test)
      - Code Review - JSF++ AV
        - Code Review for REQ_0003 (Code Review - JSF++ AV)

-s_filesCell.cpp
Requirements Traceability - Minimizing the overhead

• Traditionally a labour intensive process
  – even if static & dynamic analysis are automated.

• Automation improves quality and reduces costs through
  – Less room for human error
  – Automatic analysis of the “knock on” effects of changes
  – Reference point when changes are requested
  – A maintained RTM even when the pressure is on
STATIC ANALYSIS
Coding Standards for new developments

Quality
The best way to avoid having defects in the code is not to put them in.

Roughly 80% of C/C++ software defects are attributable to issues with 20% of the language constructs.

Standards such as MISRA-C:2004 and MISRA C++:2008 avoid this subset to improve quality.

Security
Standards such as Cert C avoid language constructs that can lead to exploitable vulnerabilities.

Style
Ensure that code is written in a particular style.
Automating “peer review” improves quality and reduces costs through
• Consistency of interpretation
• Consistency of application (no “Friday afternoon” effect!)
• Removal of potential for tension between participants.
• Speed of review process

When this is a new concept, initial resistance is likely... But it soon becomes a learning tool...

... And ultimately merely confirms that the standard is being met
Why use Complexity Metrics?

• Code is sometimes complicated.

• Sometimes complicated code is addressing a complex problem.
  – That is unavoidable!

• Sometimes complicated code is not addressing a complex problem. That code:
  – Is prone to costly error at the point of release
  – Is prone to costly error during modification
  – Will demand disproportionately extensive tests whenever changes are made
Complexity Metrics

• The principal Complexity Metrics are:
  – Knots
  – Cyclomatic Complexity

• Additional complexity metrics are:
  – Essential Knots
  – Essential Cyclomatic Complexity
Complexity Analysis - Minimizing the overhead

• Sometimes complicated code is not addressing a complex problem. That code:

  – Is prone to costly errors at the point of release
  – Is prone to costly errors during modification
  – Will demand disproportionately extensive tests whenever changes are made
Control Flow & Data Flow Analysis

• Control flow analysis
  – Control Flow Analysis is performed both on the program calling hierarchy and on the individual procedures. The rules of structured programming are applied and defects reported

• Static data flow analysis
  – Follows variables through the source code and reports any anomalous use. This is performed at procedure level and also as part of the system wide analysis

<table>
<thead>
<tr>
<th>Technique/Measure</th>
<th>Ref</th>
<th>SIL1</th>
<th>SIL2</th>
<th>SIL3</th>
<th>SIL4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Boundary value analysis</td>
<td>C.5.4</td>
<td>R</td>
<td>R</td>
<td>HR</td>
<td>HR</td>
</tr>
<tr>
<td>2 Checklists</td>
<td>B.2.5</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>3 Control flow analysis</td>
<td>C.5.9</td>
<td>R</td>
<td>HR</td>
<td>HR</td>
<td>HR</td>
</tr>
<tr>
<td>4 Data flow analysis</td>
<td>C.5.10</td>
<td>R</td>
<td>HR</td>
<td>HR</td>
<td>HR</td>
</tr>
<tr>
<td>5 Error guessing</td>
<td>C.5.5</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>6 Fagan inspections</td>
<td>C.5.15</td>
<td>---</td>
<td>R</td>
<td>R</td>
<td>HR</td>
</tr>
<tr>
<td>7 Sneak circuit analysis</td>
<td>C.5.11</td>
<td>---</td>
<td>---</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>8 Symbolic execution</td>
<td>C.5.12</td>
<td>R</td>
<td>R</td>
<td>HR</td>
<td>HR</td>
</tr>
<tr>
<td>9 Walk-throughs/design reviews</td>
<td>C.5.16</td>
<td>HR</td>
<td>HR</td>
<td>HR</td>
<td>HR</td>
</tr>
</tbody>
</table>

Table B.8 – Static analysis
Call Graph : Control Flow Visualisation

![Call Graph Image]
DYNAMIC ANALYSIS – UNIT TEST & CODE COVERAGE
Why use Unit/Module Test?

- **Unit testing** focuses on the behaviour of execution of a subset of application code.
  - Code is compiled and executed in a similar environment to that used by the application under development.
- Unit testing traditionally employs a bottom-up testing strategy in which units are tested and then integrated with other test units.
- There is clearly no complete code set to hand to initiate tests such as these, which implies the need for “harness” code to allow the code to build.
Why use Structural Coverage?

- Consistent coverage produces software ready for all eventualities.
- Code coverage data from Unit and System testing can be combined.
Unit Testing and Code Coverage - Minimizing the overhead

• Automated Unit test tools are designed to automatically generate the harness code.
  – This means that tests focus on the application code, and there is no need to debug the harness code itself!

• Unit test sequences can be stored and re-executed at will, from batch files if desired.

• Code Coverage from Unit Test or System Test can be used in isolation or combination.

• The “test-modify-retest” process cycle can be undertaken even under version control.
TEST TOOLS & TEST INDEPENDENCE
Test tools and test independence

• Static analysis
  – The interpretation of coding rules is consistent and repeatable.

• Unit test
  – Ideally, dynamic tests should be carried out independently.
  – Where that is not practical, test tools provide a framework which itself lends an element of independence.
  – Traditional Unit testing demands a certain knowledge of the code in order to write the harness.
  – Robustness tests through automatic vector generation.

• System test
  – Code coverage confirms the extent to which code has been exercised.
SUMMARY
Summary

• How new is Software Standards Compliance?

• What do we need to do?
  – IEC 62304 and other standards
  – Class levels

• How do we apply best practice?
  – Requirements Traceability
  – Coding Standards
  – Control Flow and Data Flow Analysis
  – Software Module Testing
  – Structural Coverage & Unit Test
  – Test tools & test independence
For further information:

www.ldra.com

mark.pitchford@ldra.com

jonathan.kelly@ldra.com