Development and Test Techniques for Large-Scale Spacecraft Projects
Agenda

• Introduction
  – Key Drivers
  – Certification Standards

• Some key techniques
  – Requirement Traceability
  – Code Review
  – Unit Testing
  – Dynamic Analysis
  – Model based development
  – Simulation Environments
  – Consolidating the results

• Conclusions
• Key Drivers

*Cost to Repair Software, Cost of Lost Opportunities, Cost of Lost Customers
Introduction

• Drivers Specific to Spaceflight
  – Failures are generally catastrophic
  – Systems are complicated
  – Software is a key driver of systems
  – Over half of failures are due to software issues (Cheng, SCSRA Annual Workshop)
  – Costs are high
  – Mitigation costs are low compared to cost of failure
### Introduction

<table>
<thead>
<tr>
<th>Domain</th>
<th>Standard Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avionics</td>
<td>DO-178B (First published 1992) / DO-178C</td>
</tr>
<tr>
<td>Industrial</td>
<td>IEC 61508 (First published 1998, Updated 2010)</td>
</tr>
<tr>
<td>Railway</td>
<td>CENELEC EN 50128 (First published 2001)</td>
</tr>
<tr>
<td>Nuclear</td>
<td>IEC 61513 (First published 2001)</td>
</tr>
<tr>
<td>Automotive</td>
<td>ISO 26262 (Published 2011)</td>
</tr>
<tr>
<td>Medical</td>
<td>IEC 62304 (First published 2006)</td>
</tr>
<tr>
<td>Process</td>
<td>IEC 61511 (First published 2003)</td>
</tr>
</tbody>
</table>
Some Key Techniques
Generally: Process

ARP4754A Process
Requirement Traceability

System requirements allocated to Software

High-Level Requirements or Specification Model

Low-Level Requirements or Design Model

Source Code

Executable Object Code

Review and Analysis Results

SW Architecture

Test Cases

Test Procedures

Test Results

Level A

Level B and C

Level D
Code Review

- Detect potential problems
Code Review

- Use standards to prevent faults
- Verify any coding standards, even company specific ones
- Over 1000 rules can be checked:
  - Static rules
  - Complexity rules
  - Data flow rules
  - User defined rules
Code Review

- Importance in ensuring uniform code
- Project wide definition
- Tailor the rule set for different portions and subcontractors
- C++ can be a challenge for code review
- Use of templates/polymorphism can also be a challenge
- Plan to tailor an existing standards model to fit your program
- Include files matter – portability of structure (can pass or fail based upon analysis of include files)
Unit Testing

- Unit testing is a key enabling technology
- Allows you to test scenarios before the mission
- Breaks testing activities into manageable parts
- Allows the testing process to start early, and continue throughout the development process
- Can be paired with coverage and requirements for more value
Unit Test

• Definition
  – The aim of unit test is to isolate a software system into its most basic constituent elements
  – Each element can be stressed at its interfaces across a wide range of possible inputs to assess the robustness of its implementation

• Types
  – White box – provides full visibility with regard to path coverage through the unit – use of test tools to “instrument” the source code
  – Black box – the “test driver” only has insight into the unit of source code at its interface, the operation of the unit is opaque to the test
  – Grey box – the operation of the unit is partially transparent to the test, greater flexibility is allowed to mix and match white & black box test
Constituents of a Unit Test

• Source code under test

• Test driver
  – This is its own program which will either directly incorporate the source code or link with the source code object library

• Set of test inputs
  – May be directly incorporated into the test driver or read from a file

• Stubs
  – May be file-based code inserted into test driver via a relative path
  – May be a managed stub with test case specific behavior inserted directly into the test driver

• Build environment

• Execution environment
Portability of a Unit Test

- Key Ideas
  - In order to ensure the portability of a unit test all dependencies on the user’s local host machine must be eliminated
  - One way to achieve this is to define a rigid directory structure for the unit test project
  - When it is necessary to reference a file within this defined structure only relative file paths are used
  - The unit test project is stored within the CM system often alongside the source code project
Portability of a Unit Test

• Key Idea - It is important to establish a common “unit test configuration standard” in order to ensure the recreation and successful regression of the unit test
  – Paths and macros are important
  – Any configuration settings which define execution behavior
Unit Test Execution Environments

- A spacecraft flight software system will typically execute on an embedded radiation hardened microprocessor.
- Hardware often is not available until late in the development cycle which necessitates the use of some kind of simulator to execute the unit test.
- Several possibilities exist in this regard:
  - Generic target simulation – no RTOS, basic I/O services with host, essentially a simulation of the target's instruction set.
  - High fidelity target simulation – simulator emulates all aspects of the target CPU, the unit test executes with the RTOS used by the flight system.
Simulation Environments

• Traditional View
  – Validity of simulation is based on nominal inputs & outputs

• New View
  – Validity of simulation is based on nominal and off-nominal inputs & outputs
  – Simulations need to be bug for bug compatible
  – Important to include failure modes in simulations

• Simulations of processors and environments need to be coupled

• System Simulation and Coverage work well together

• RAD hardened processors are good targets for simulation
Dynamic Analysis

• Statement Coverage
• Branch Coverage
• MC/DC Coverage
  – ISO 26262 ASIL D
  – DO-178B Level A
• Object Code Coverage
  – DO-178B Level A
• LCSAJ Coverage

MC/DC – Multiple Condition/Decision Coverage
LCSAJ – Linear Code Sequence And Jump
Accruing Coverage Credit

Simulation Test Cases

Host Computer

Target H/W

Test Cases

Additional Test Cases

C Level Coverage

≤ 100 %

C Level Coverage

+ 100 %

C Level Coverage

+ 100 %

ASM Level (Level A)

+ 100 %
Call Graph Coverage for Integration
Model Based Development
With Modeling, Simulation, Code Generation, and Verification
Model Based Development
With Integrated Model-Based Design & Test

Executable models
- unambiguous
- only “one truth”

Simulation
- reduces “real” prototypes
- systematic “what-if” analysis

Design with Simulation

Executable Specifications

Continuous Test and Verification

Test with Design
- detects errors earlier

Automatic Code Generation

Automatic code generation
- minimizes coding errors
Model-based Development

- Two most common in spaceflight software development
  - Matlab Simulink (best suited for GN&C algorithms, direct auto-generation of C/C++ code from the model)
  - Rhapsody (structure generation with C++ framework)
  - xUML (executable UML)

- Advantages
  - G&C diagrams directly transition to code
  - Conformance of code is easy to verify
  - Qualification of code generation is available
Challenges

- Modification of generated code for compliance (there is the ability with Matlab/Simulink to customize behavior but may not be completely documented)
- MISRA or other standards compliant code generation
- Connecting models to high level requirements
Consolidating the Results

• Systemwide requirement traceability
• System wide coverage
• System integration coverage
• Common Formats
• Management User Interfaces
• Project Management Spreadsheets
Conclusions

• Software process in spacecraft flight software development is extremely important.

• Understanding your software as components and understanding the connections between those components is key part of reducing risk.

• Investing in process early can save your mission.
About the Authors

• **Jay Thomas** is a Technical Development Manager at LDRA. He currently helps customers get complex projects through certification, launch and beyond. His prior experience includes work on flight software development at SpaceX and space vehicle simulation at Space Systems/Loral.

• **Trevor Tidwell** has 16 years of experience working for Lockheed Martin and United Space Alliance in human rated flight software development and testing for NASA’s Space Shuttle and Orion MPCV programs.

• **Lou Abney** is a Field Application Engineer at LDRA. He has over twenty years of aerospace experience in the manned flight systems and software working on the International Space Station, the MIR space station, and the Orion crew capsule.
For further information:

www.ldra.com

info@ldra.com