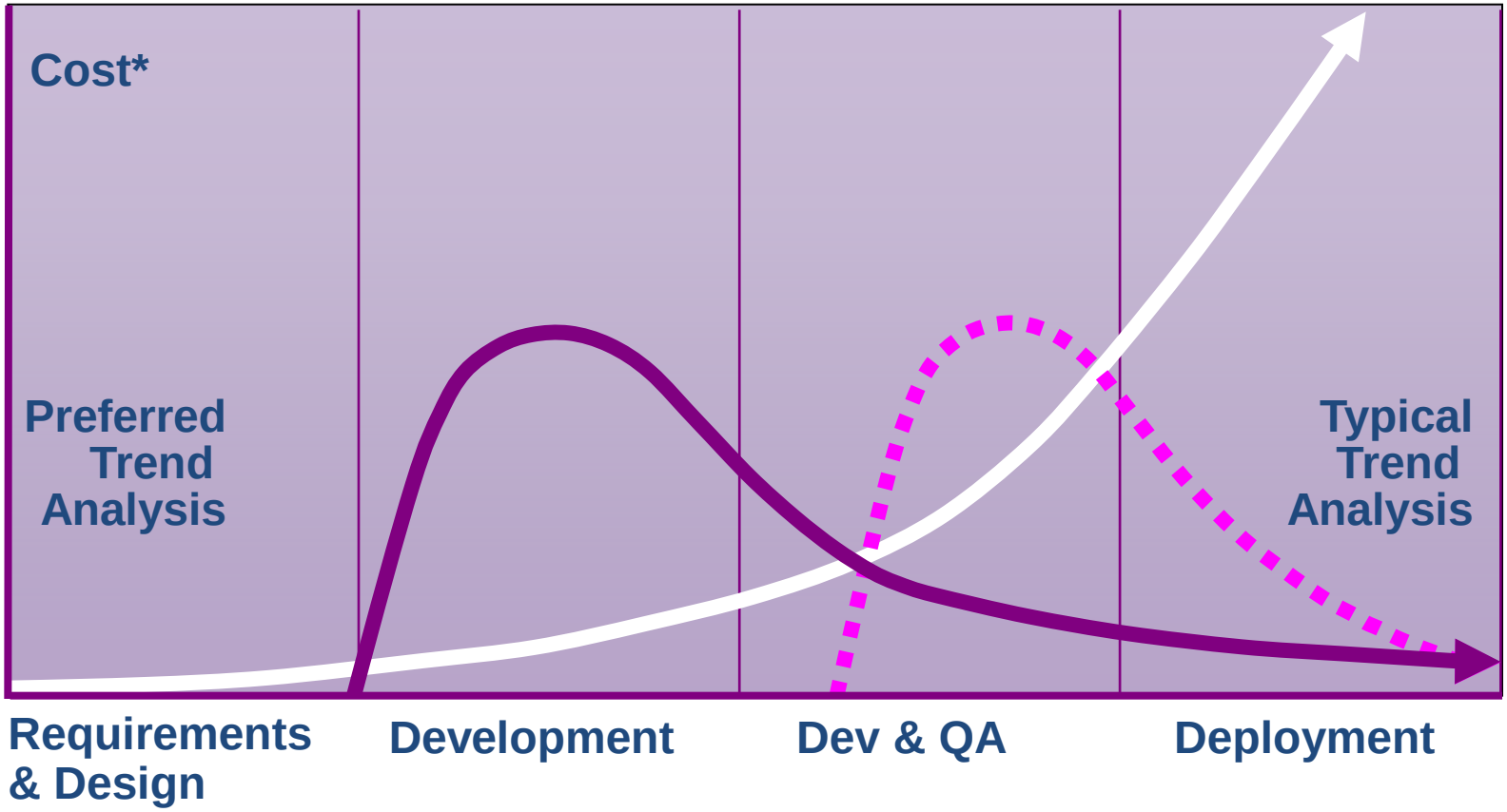
Development and Test Techniques for Large-Scale Spacecraft Projects

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

Introduction

- 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

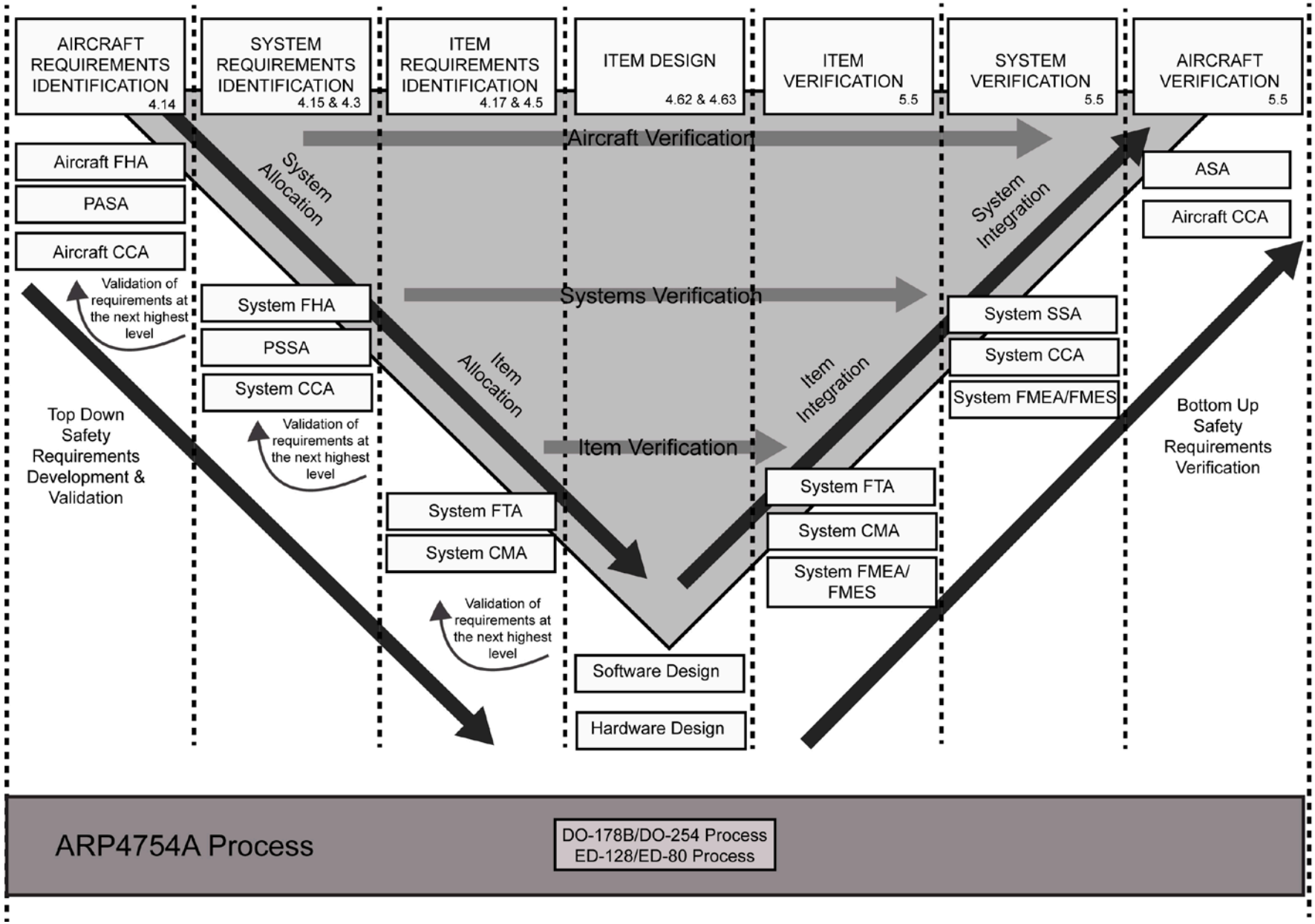
Introduction

LDRA

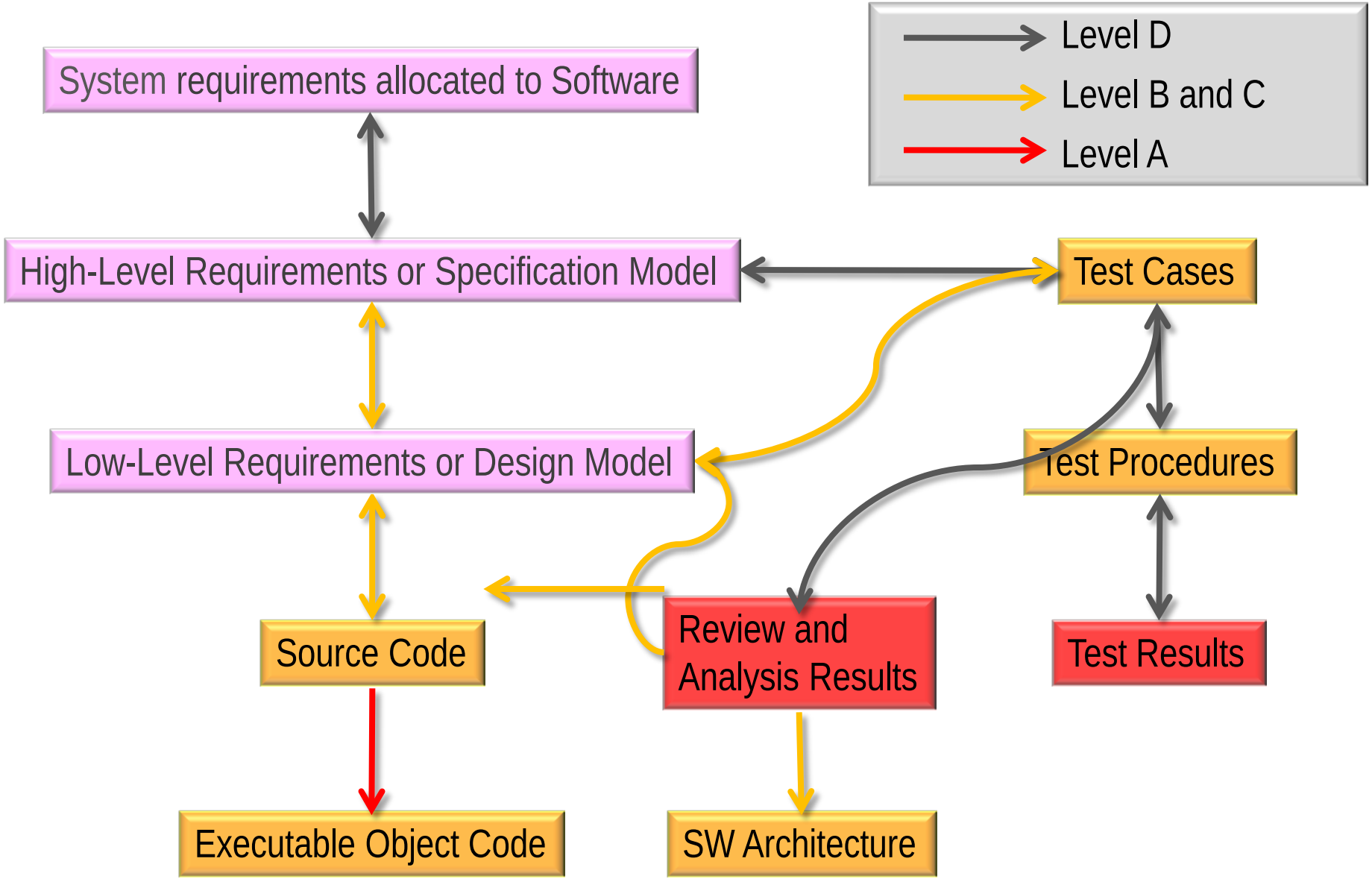
Avionics	DO-178B (First published 1992) / DO-178C
Industrial	IEC 61508 (First published 1998, Updated 2010)
Railway	CENELEC EN 50128 (First published 2001)
Nuclear	IEC 61513 (First published 2001)
Automotive	ISO 26262 (Published 2011)
Medical	IEC 62304 (First published 2006)
Process	IEC 61511 (First published 2003)

Some Key Techniques

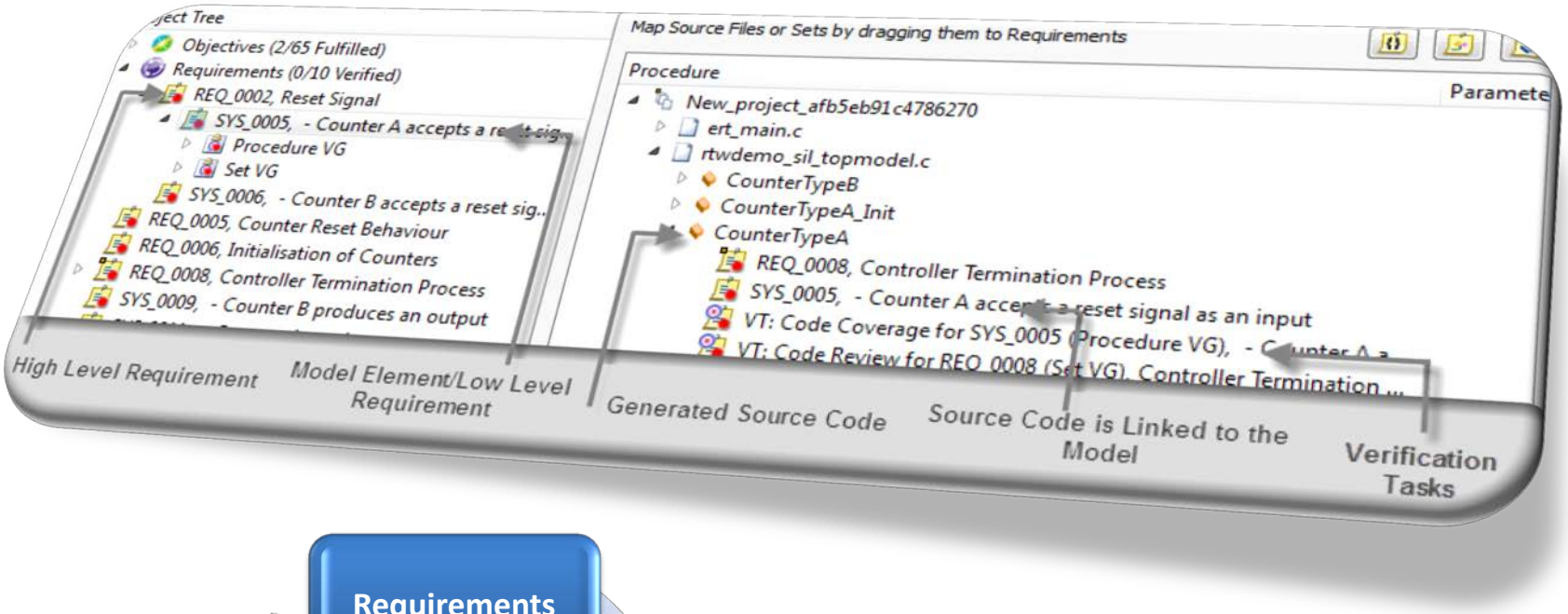
Generally: Process

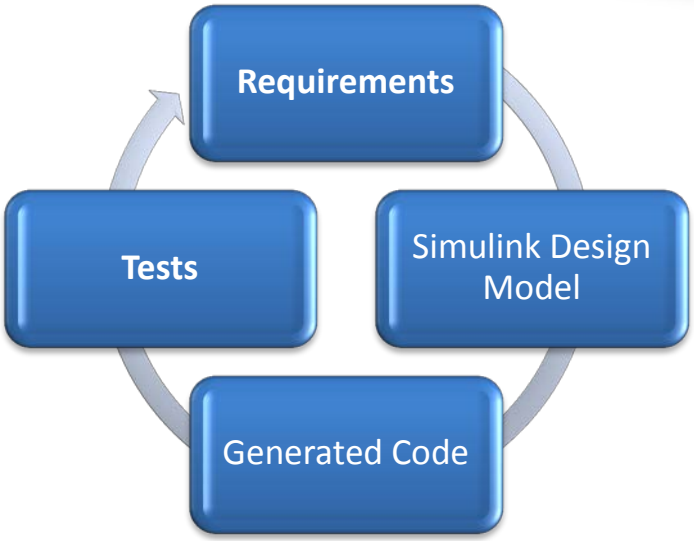


Requirement Traceability



Traceability Management





Code Review

- Detect potential problems

LDRA TBvision 9.0.0 © 2012 LDRA Ltd

Source View Configure Website Links Help

File View

File Explorer

- Eclipse_C_Array
  - Array.c
    - Global Variables
    - Global Typedefs
    - Global Macros
    - init
    - print
    - forwardPrint
    - backwardPrint
    - main

Class View | File View

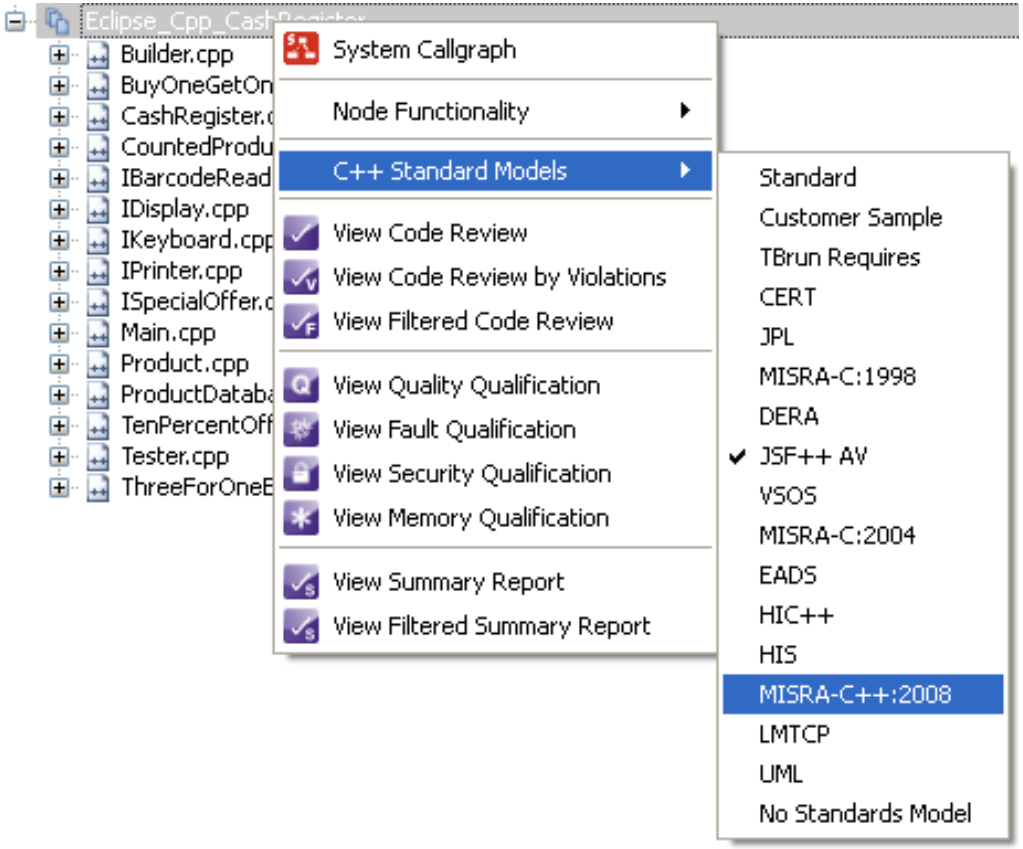
Results View

Available Results | Code Review : (By Violation)

	Number Violated	Standard Code	Standard Code
Violations			
Array Bound exceeded.	3	Required	MISRA-C:1998 4 | MISRA-C:2004 21.1 | D
Array Bound exceeded. : anArray[*]; accessed=6, range=0-5		Required	MISRA-C:1998 4 | MISRA-C:2004 21.1 | D
Array Bound exceeded. : theArray[*]; accessed=6, range=0-5		Required	MISRA-C:1998 4 | MISRA-C:2004 21.1 | D
Array Bound exceeded. : theArray[*]; accessed=6, range=0-5		Required	MISRA-C:1998 4 | MISRA-C:2004 21.1 | D
Array bound exceeded at call. : theArray		Required	MISRA-C:2004 21.1 | CERT ARR30-C | CW
Signed/unsigned conversion without cast. : (int and unsigned char): it...		Required	MISRA-C:1998 43,48 | MISRA-C:2004 10.1
Prototype and Definition name mismatch. : aValue		Required	MISRA-C:1998 74 | MISRA-C:2004 16.4 | D
Included file is not permitted.		Required	MISRA-C:1998 121,123,124,127 | MISRA-
Identifier reuse: proc param vs proc param.	2	Advisory	MISRA-C:2004 5.7 | MISRA-AC 5.7
Identifier reuse: variable vs variable.	3	Advisory	MISRA-C:2004 5.7 | MISRA-AC 5.7
Scope of variable could be reduced	2	Advisory	MISRA-C:1998 22 | MISRA-C:2004 8.7 | D
Basic type declaration used.		Advisory	MISRA-C:1998 13 | MISRA-C:2004 6.3 | D
Value is not of appropriate type. : (int and unsigned char): iter = 6U		Advisory	MISRA-C:1998 18,48 | MISRA-C:2004 6.1,

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

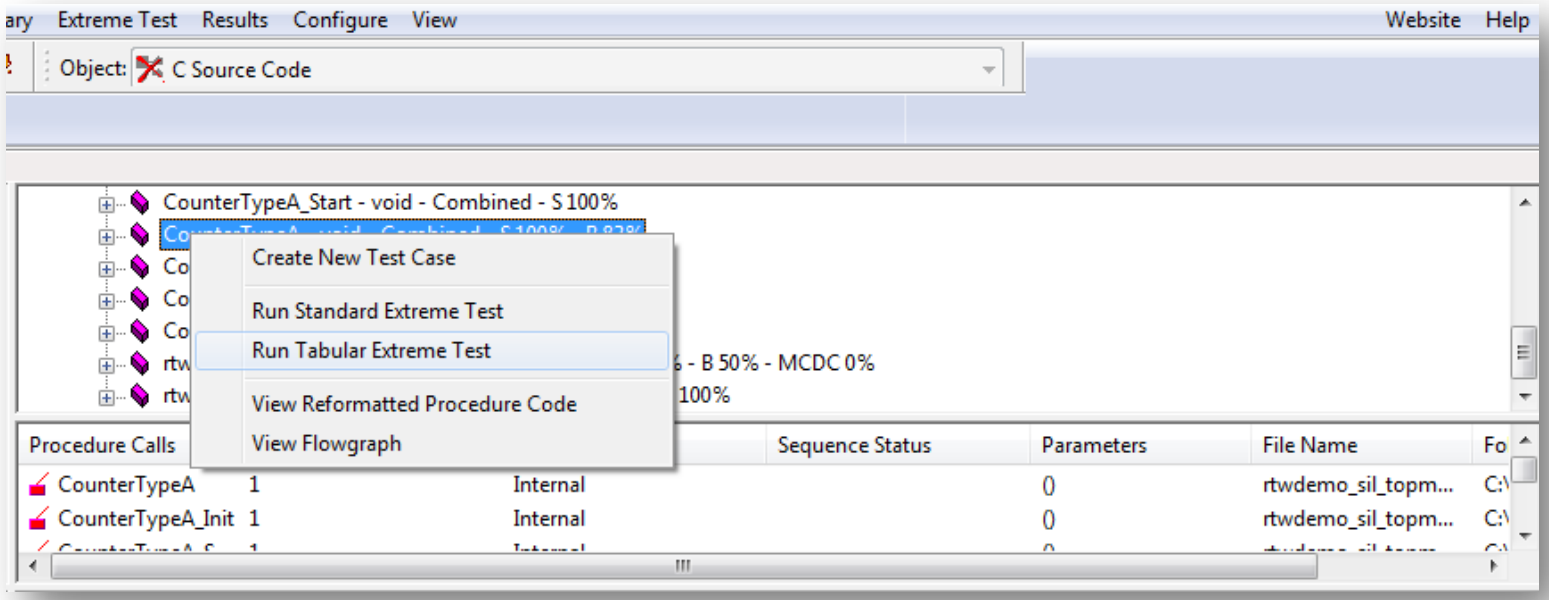


Code Review

- 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

LDRA

- 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

LDRA

- Key Idea - It is important to establish a common “unit test configuration standard” in order to ensure the re-creation 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 targets 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

LDRA

- 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 processers 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

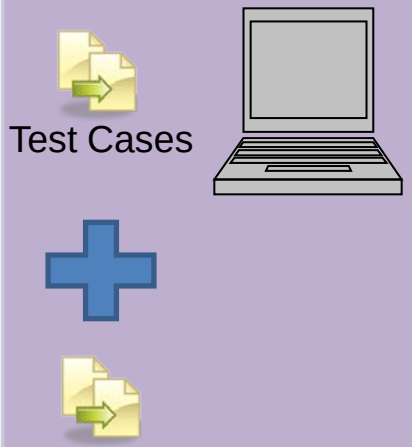
LDRA

Simulation

C Level Coverage

≤ 100 %

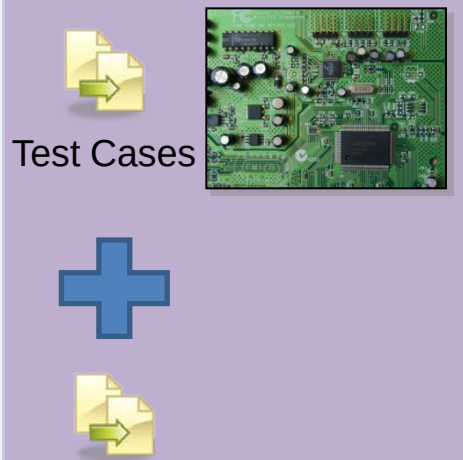
Host Computer



C Level Coverage

+ 100 %

Target H/W



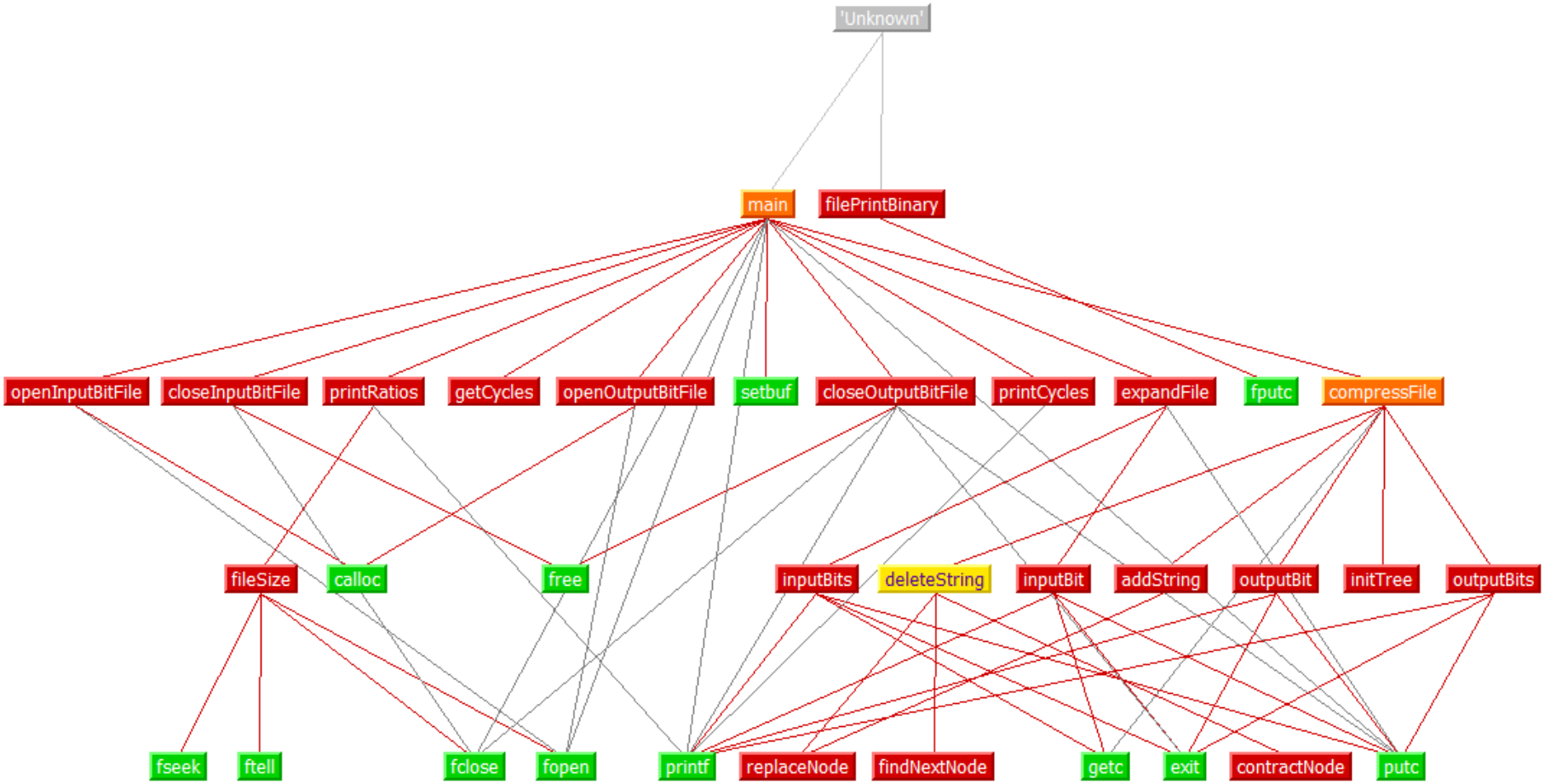
C Level Coverage

+ 100 %

ASM Level (Level A)

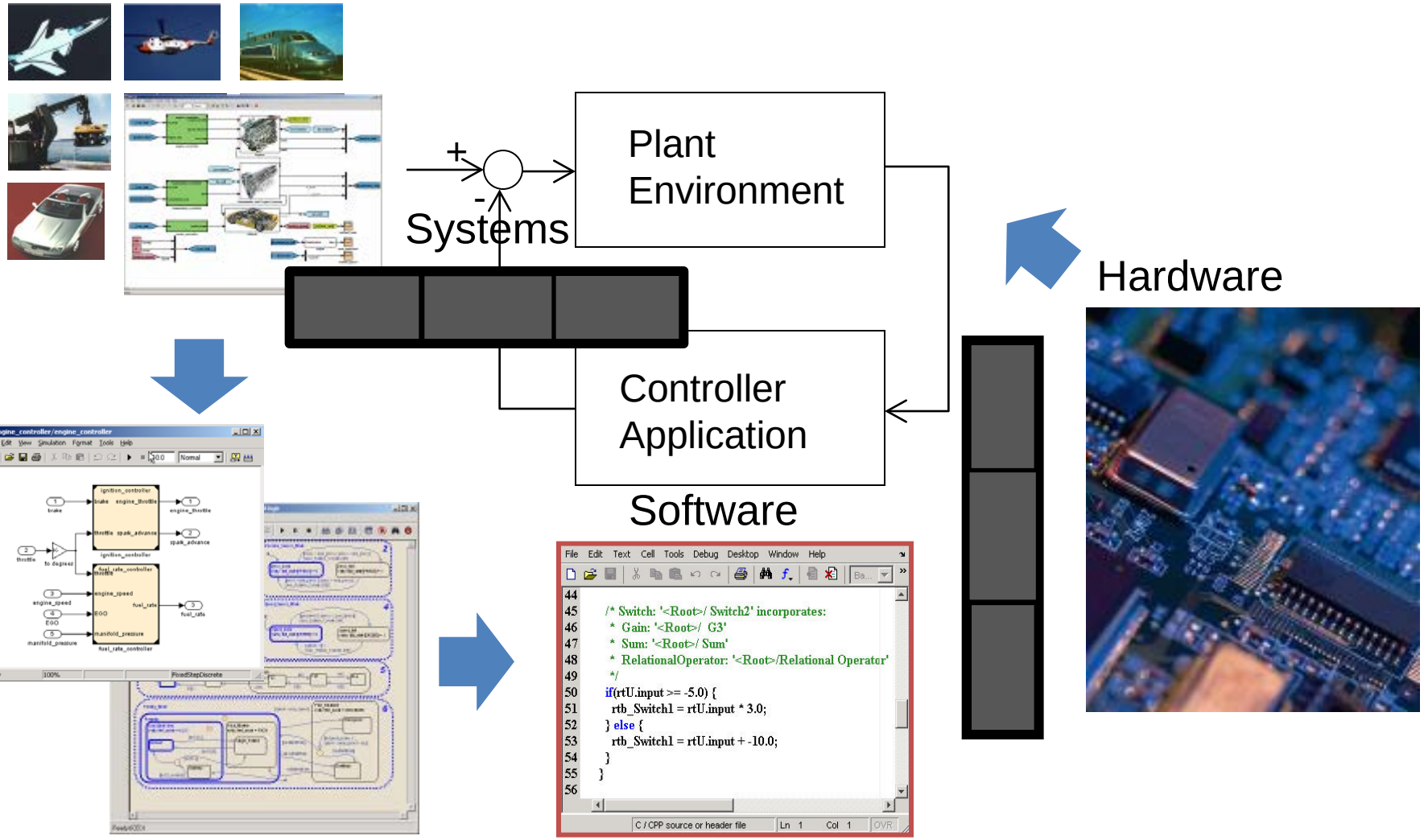
+ 100 %

Call Graph Coverage for Integration



Model Based Development

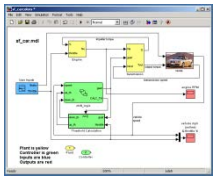
With Modeling, Simulation, Code Generation, and Verification

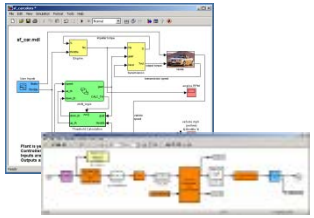


Model Based Development

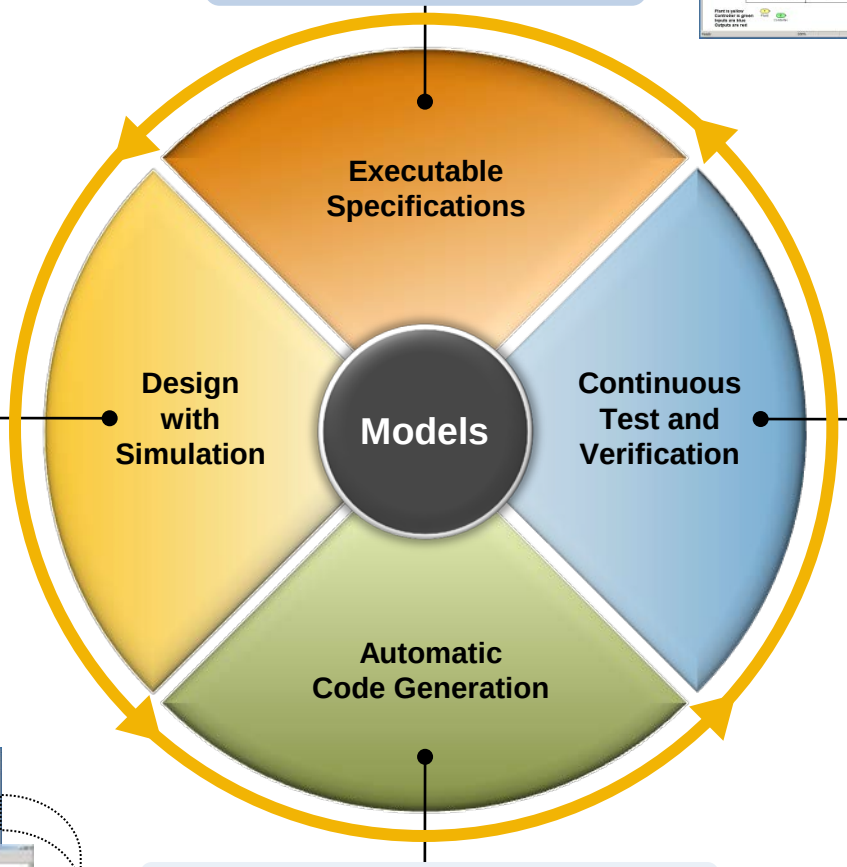
With Integrated Model-Based Design & Test

Executable models
-unambiguous
-only "one truth"

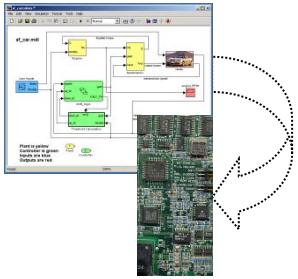


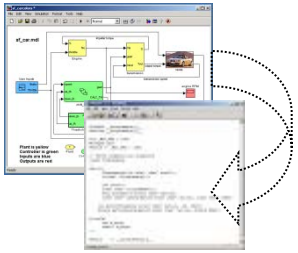


Simulation
-reduces "real" prototypes
-systematic "what-if" analysis



Test with Design
- detects errors earlier





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

- 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