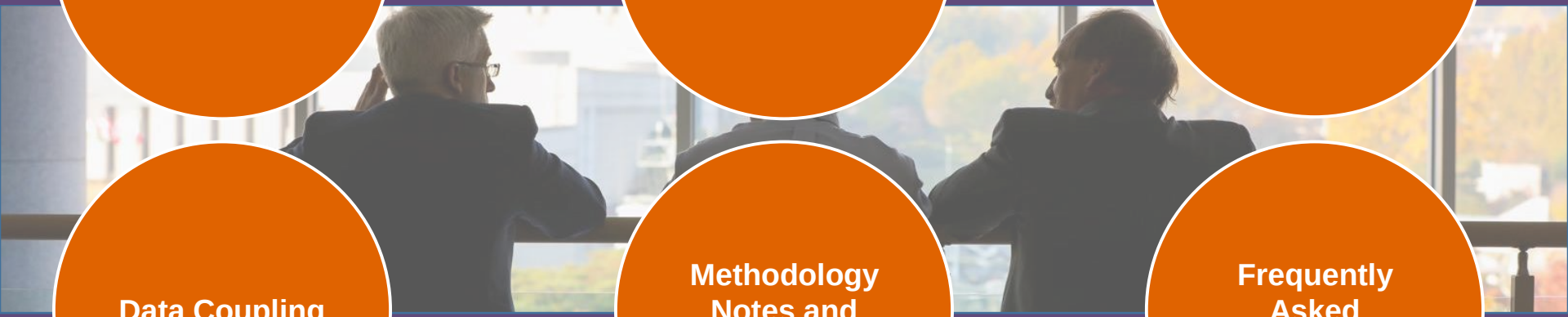


Introduction to Data Coupling and Control Coupling

LDRA Overview

DO-178C Overview

Control Coupling



Data Coupling

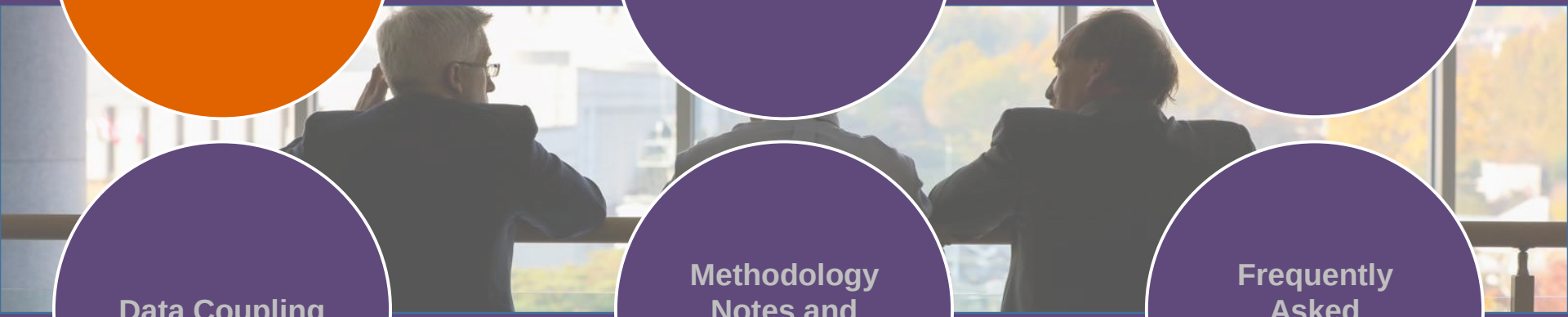
Methodology Notes and Benefits

Frequently Asked Questions

LDRA Overview

DO-178C Overview

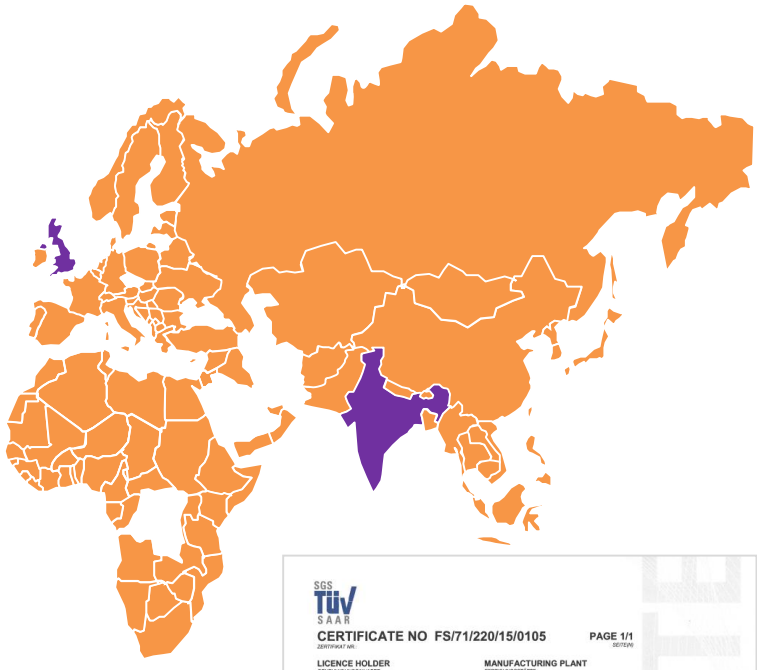
Control Coupling



Data Coupling

Methodology Notes and Benefits

Frequently Asked Questions



Established 1975

ISO 9001 certified company

Certified for use in safety related software development according to IEC 61508, IEC 62304, EN 50128, IEC 60880 and ISO 26262

Provider of Software Quality, Compliance Management and Testing Solutions

Active participants in standards e.g. DO-178C, MISRA C/C++, FACE and CERT

SGS TÜV SAAR

CERTIFICATE NO FS/71/220/15/0105 PAGE 1/1

LICENCE HOLDER	MANUFACTURING PLANT
LDRA LTD PORTSIDE, MONKS FERRY, WIRRAL, CH41 5LH, UNITED KINGDOM	LDRA LTD PORTSIDE, MONKS FERRY, WIRRAL, CH41 5LH, UNITED KINGDOM

PROJECT NO/-ID	LICENSED TEST MARK	CERT. REPORT NO.
K1C2-AU01	SGS TÜV SAAR FUNCTIONAL SAFETY APPROVED	K1C20003

Tested according to	IEC 61508:2010 Ed. 2.0 ISO 26262:2011 EN 50128:2011 IEC 60880:2006 IEC 62304:2006
Certified product(s)	LDRA tool suite LDRAunit LDRArules LDRAcover LDRAlite
Model(s)	Version 9.5
Technical Data and Parameter	Usable in development of safety related software acc. to: • ISO 26262 up to ASIL D, TCL1 can be reached • IEC 61508 up to SIL 4, class T2 tool • EN 50128 up to SW-SIL 4, class T2 tool • IEC 62304 up to SW safety Class C • IEC 60880
Specific Requirements	The certificate is based on voluntarily tests. Any changes to the design, components or processing may require repetition of some parts of the qualification in order to retain the certification. The certificate report is an integral part of this certificate. The safety related application guidelines (refer to K1C20003) shall be maintained.

Certification Body for Functional Safety SGS-TÜV Saar GmbH

Munich, 04.11.2015

Gudrun Neumann

CERTIFICATE

LDRA Standards Experience & Pedigree

Professor Mike Hennell

- Member of SC-205 / WG-71 (DO-178C) formal methods subgroup
- Member of MISRA C committee and MISRA C++ committee
- Member of the working group drafting a proposed secureC annex for the C language definition (SC 22 / WG14)

Bill St Clair

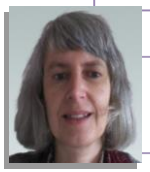
- Member of SC-205 / WG-71 (DO-178C) Object Oriented Technology subgroup

Shan Bhattacharya

- Member of FACE Consortium Technical Working Group Conformance Verification Matrix Subcommittee
- Member of FACE Consortium Integration Workshop Standing Committee

Dr Clive Pygott

- Member of ISO software vulnerabilities working group (SC 22 / WG 23)
- Member of MISRA C++ committee
- Member of the working group drafting a proposed secureC annex for the C language definition (SC 22 / WG14)



Liz Whiting

- Member of MISRA C committee language definition (WG14)

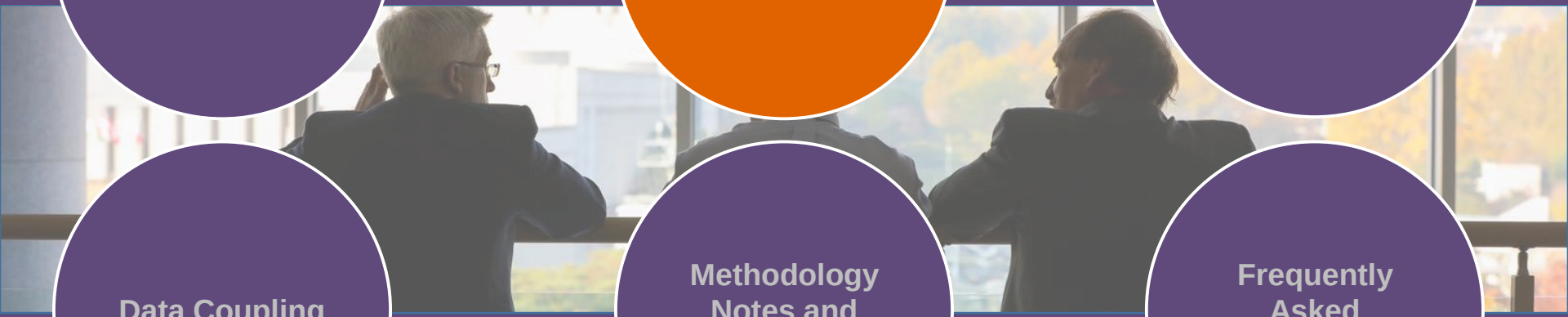
Chris Tapp

- Chairman of MISRA C++ committee
- Member of MISRA C committee language definition (WG14)

LDRA Overview

DO-178C Overview

Control Coupling



Data Coupling

Methodology Notes and Benefits

Frequently Asked Questions

- DO-178C Overview

Delivering Software Quality and Security through Test, Analysis and Requirements Traceability

Challenges on the Path to DO-178B/C

Compliance Management

Tool Qualification

Object Code Verification

Target Testing

Structural Coverage

Data & Control Coupling Analysis

Coding Standards Compliance

Objective Tracking

Requirements Traceability

CERTIFICATE Of QUALITY

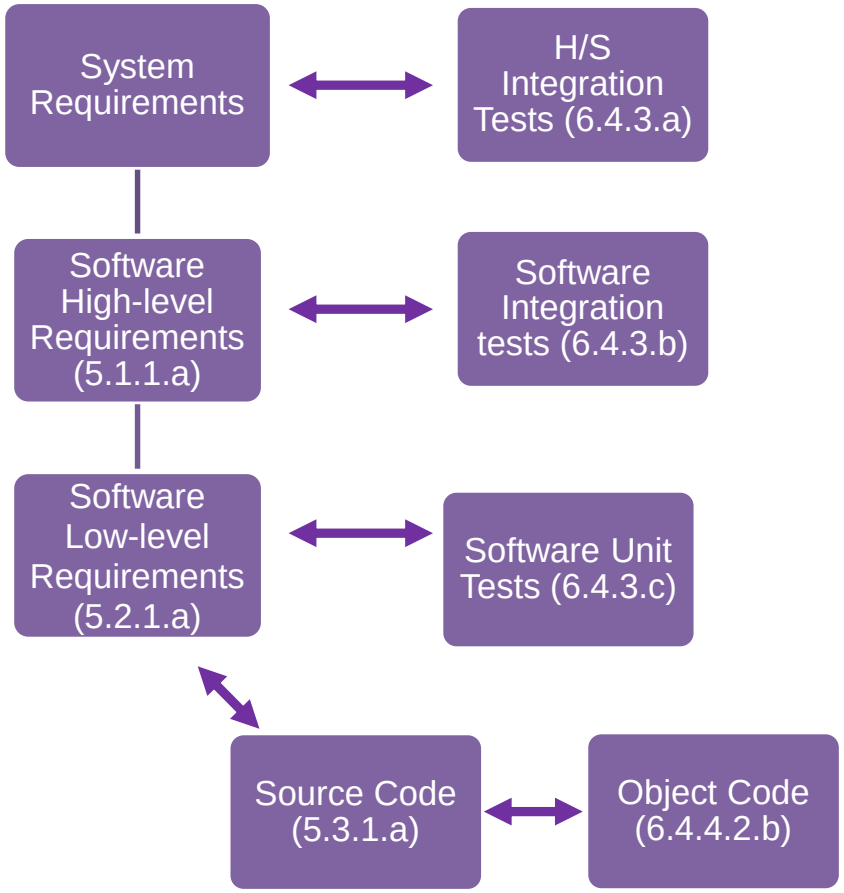
Software has been tested and conforms to **DO-178C**

Reduce Cost of Compliance

Manage Distributed Team

Reduce Time To Compliance And Market

Linking Requirements, Code and Tests



TBmanager Objective Summary Report

Objectives

Summary

- 0 Fulfilled
- 70 Partially Fulfilled
- 11 Unfulfilled

LLR_0320: Mounting Area Number of Lamps

- Verification Status: Verified
- Type: Low Level
- Body: The MountArea class shall provide the number of lamps which will fit within its mounting area
  - **Mapped Procedures**
  - int TunnelData::MountingArea::NumLamps(TunnelData::LampAttributes MyLamp); - C:_LDRA_WorkArea\950\Examples\Toolsuite\Tunnel_5.0\Source_Code\Mountings.cpp

LLR_0330: System Data Instantiation

- Verification Status: Verified
- Type: Low Level
- Body: System data shall be instantiated only once as this data will be global for the system
  - **Mapped Procedures**
  - TunnelData::SystemData::SystemData(const TunnelData::SystemData & dummy1); - C:_LDRA_WorkArea\950\Examples\Toolsuite\Tunnel_5.0\Source_Code\Systemdata.cpp
  - TunnelData::SystemData * TunnelData::SystemData::Instance(); - C:_LDRA_WorkArea\950\Examples\Toolsuite\Tunnel_5.0\Source_Code\Systemdata.cpp
  - void TunnelData::SystemData:: =(const TunnelData::SystemData & dummy1); - C:_LDRA_WorkArea\950\Examples\Toolsuite\Tunnel_5.0\Source_Code\Systemdata.cpp
  - TunnelData::SystemData::SystemData(); - C:_LDRA_WorkArea\950\Examples\Toolsuite\Tunnel_5.0\Source_Code\Systemdata.cpp

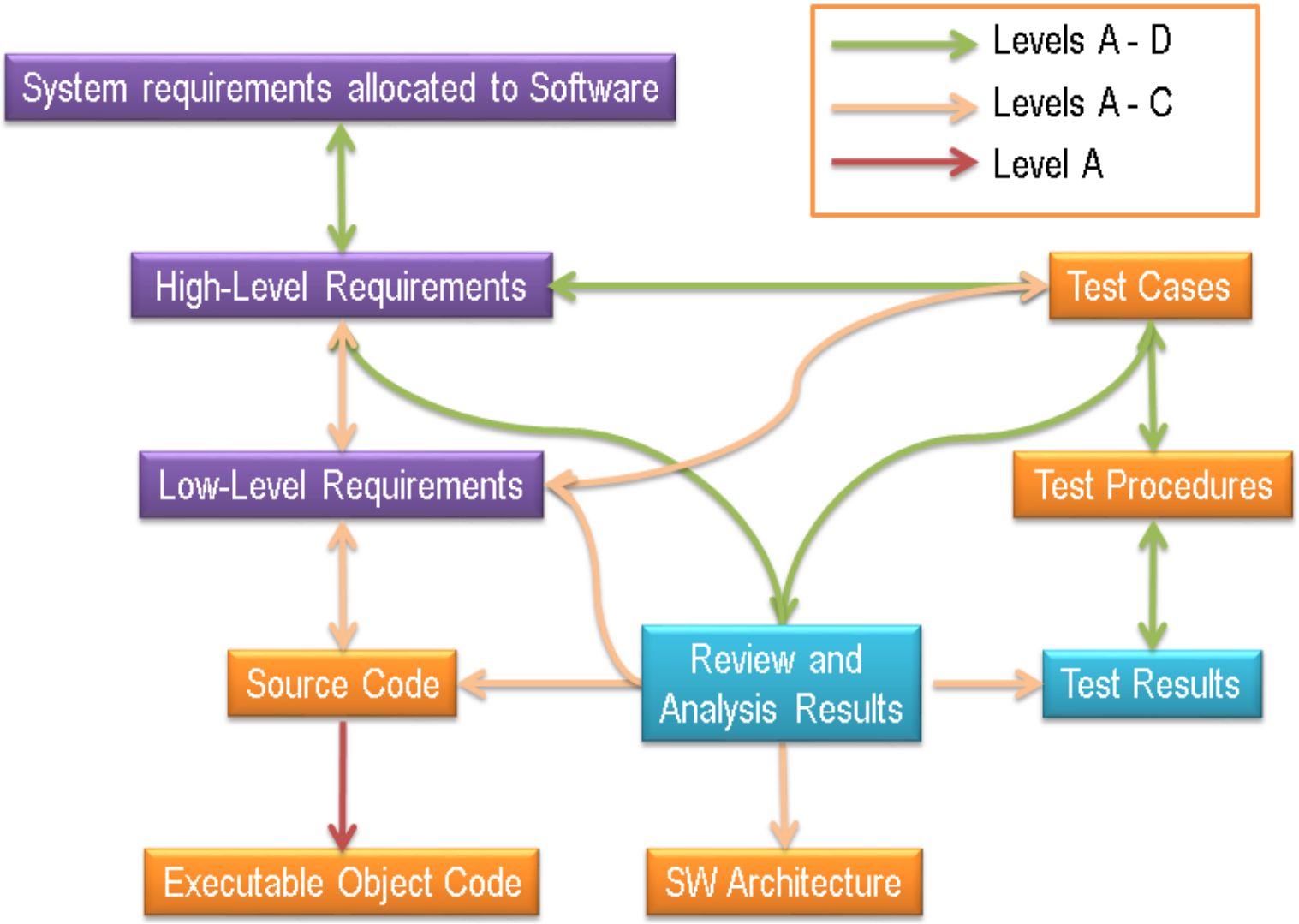
LLR_0340: System Data Initialisation

- Verification Status: Not Verified
- Type: Low Level
- Body: All system data tied to the tunnel lighting system as a whole shall be managed to provide data needed for system level data acquisition and decision making
  - **Mapped Procedures**
  - void TunnelData::SystemData::InitialiseParams(Sint_32 * pSystemDataArray); - C:_LDRA_WorkArea\950\Examples\Toolsuite\Tunnel_5.0\Source_Code\Systemdata.cpp

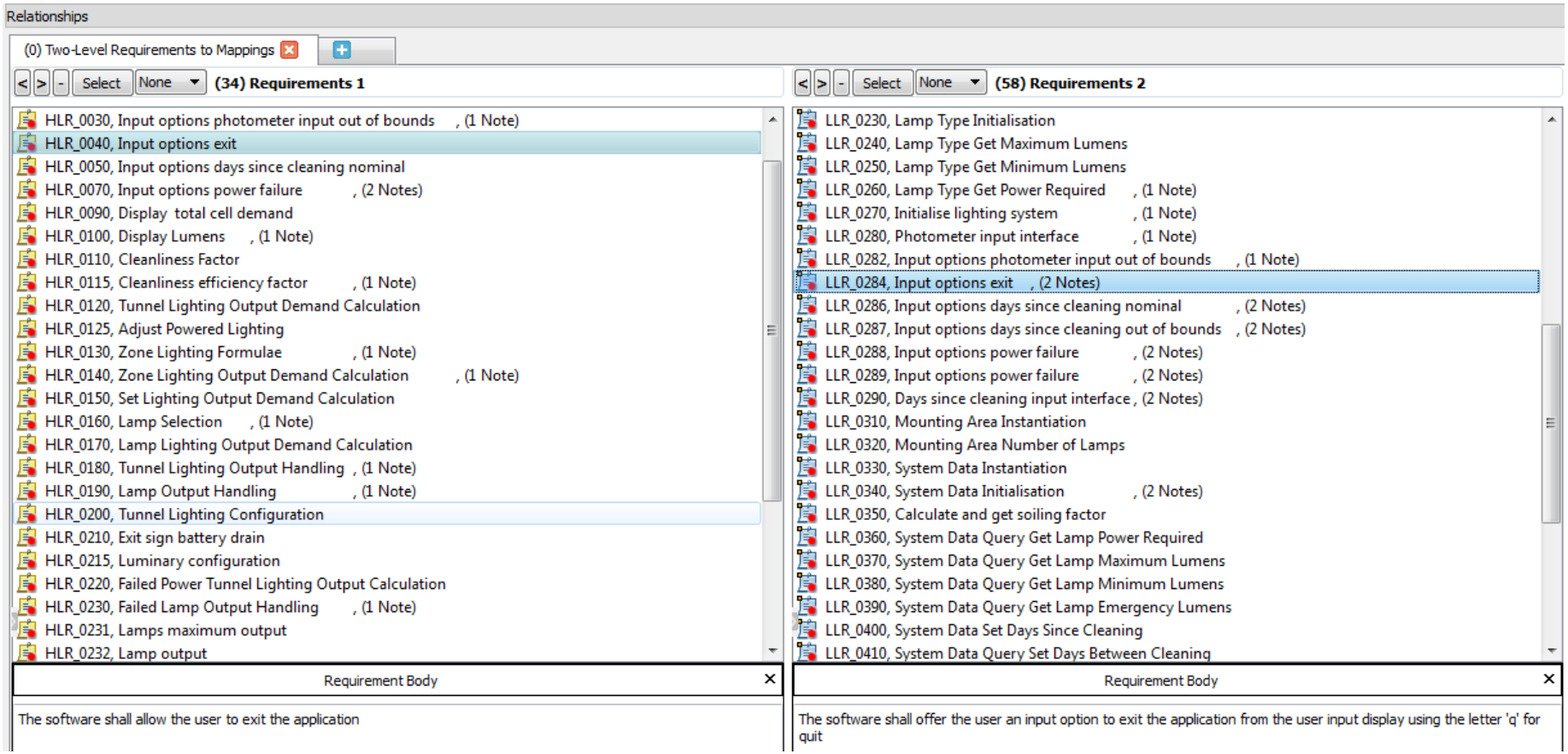
	TCI Description	Text case data n...	Name: Soiling f...	Name: Soiling ...	Verify that soili...	Low Level Requ...	Verify that for a ...
Requirement Body	**Any Item x TCI**	TCI_0345: Text c...	TCI_5080: Nam...	TCI_5090: Nam...	TCI_5100: Verify...	TCI_5140: Low L...	TCI_5180: Verify...
The Cell output shall be calculated as the n...	LLR_0050					X	
A Duo lamp model shall be applied if a lam...	LLR_0060						X
Get Data shall parse the tunnel lighting syst...	LLR_0100						
When power is sent to a lamp, the number ...	LLR_0130						
When queried, a lamp object shall be provi...	LLR_0140						
A lampmodel shall be initiliased by a provi...	LLR_0170						
A lampmodel shall provide its dimensions, ...	LLR_0180						

Traceability from Requirements through Code and Tests

Traceability Across DAL Levels

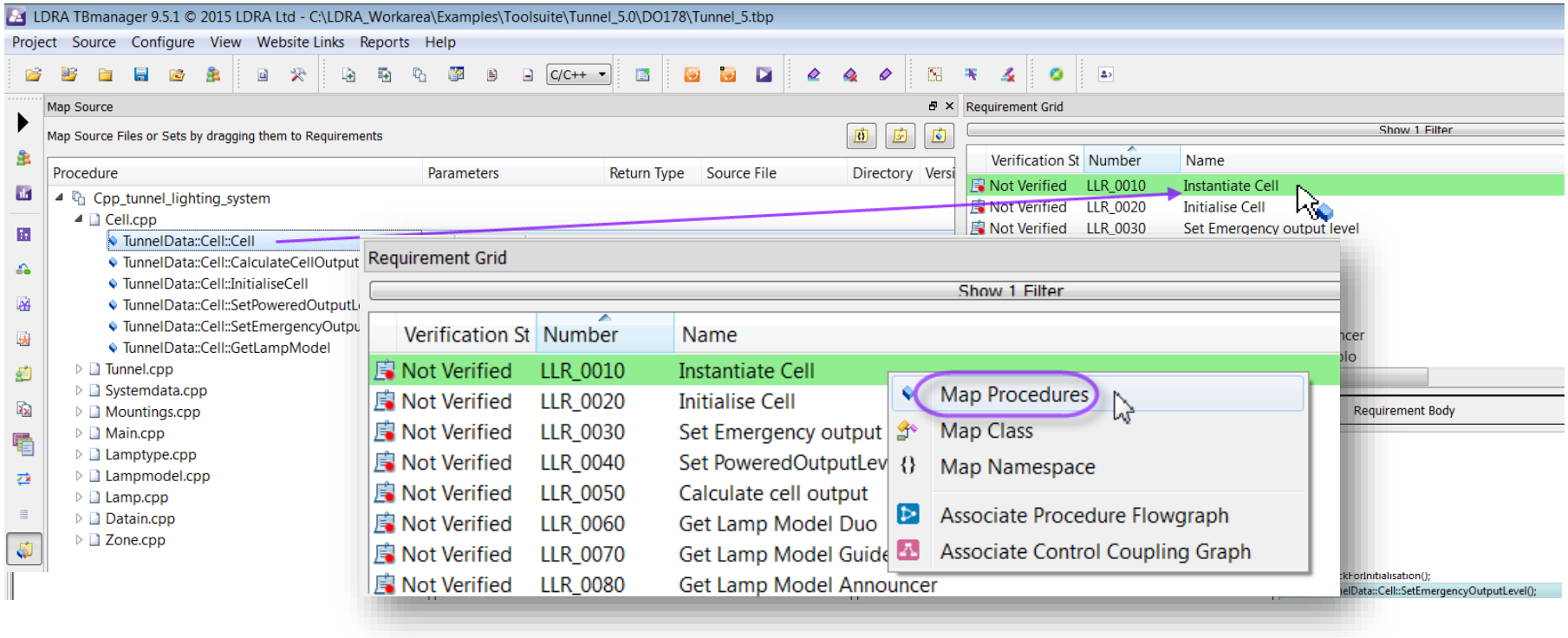


Reviewing Requirements



Reviewing requirements and traceability data

Linking Requirements to Code



Mapping low-level requirements to source code

- Ensure low-level requirements are implemented
- Identifies implemented functionality missing in requirements
- Key to structural coverage analysis
- Correlates expected behavior and/or design details to test design and test measurement

Safety and Security with Coding Standards

Code

```
int32_t a[ 10 ];
uint32_t i;

for ( i = 0; i < 20; ++i )
{
   a[ i ] = 0;
}
```

Result

Depending on the runtime environment (OS, etc), this will result in an exception or overwrite unrelated memory.

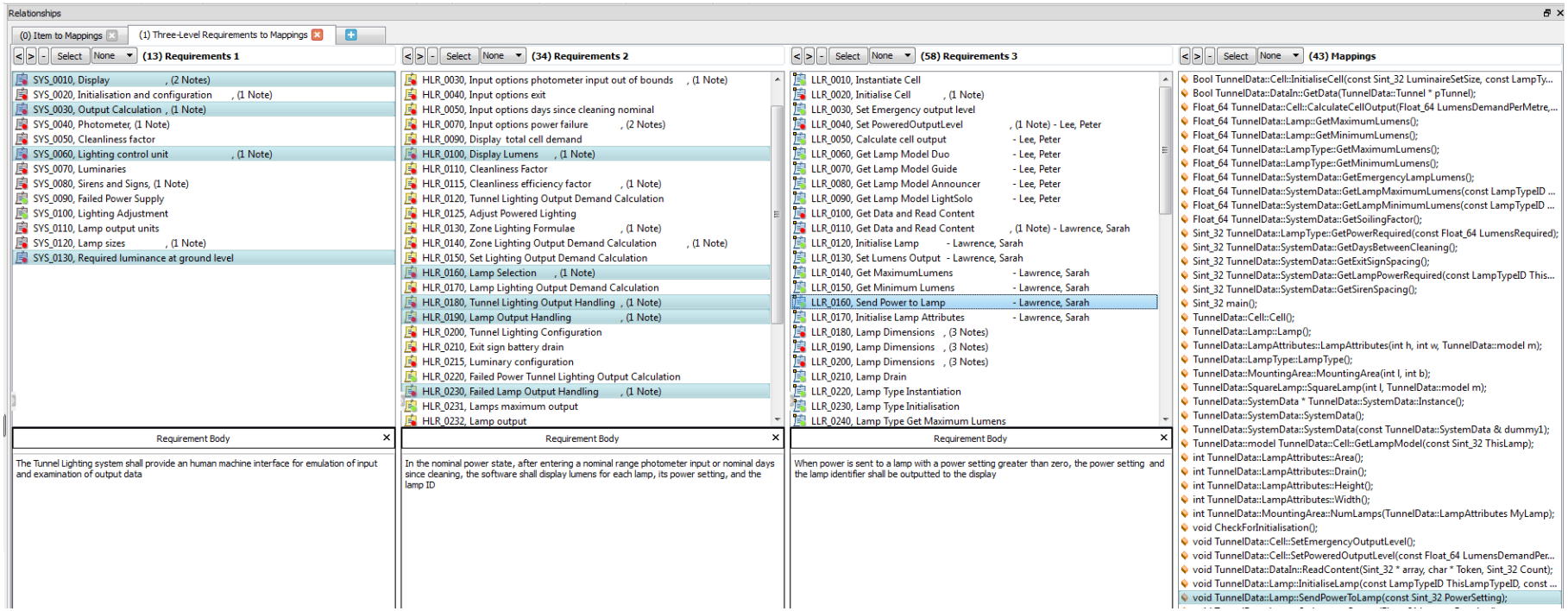
Pre-empt security and reliability issues early in the life-cycle

- Reduce cost of verification early in the process by developing more verifiable software
- Eliminate defects statically, either by pre-empting or detection is vastly cheaper
- Consistent coding style and form across teams
- Portability & reusability across varying architecture and compiler/tool chains

Security vs Reliability

- Standards families such as MISRA focus on reliability
- CERT C / CWE were developed to address security vulnerabilities
- Both categories of standards can be used in conjunction
- Tailorable for new development, legacy code, and runtime error checking

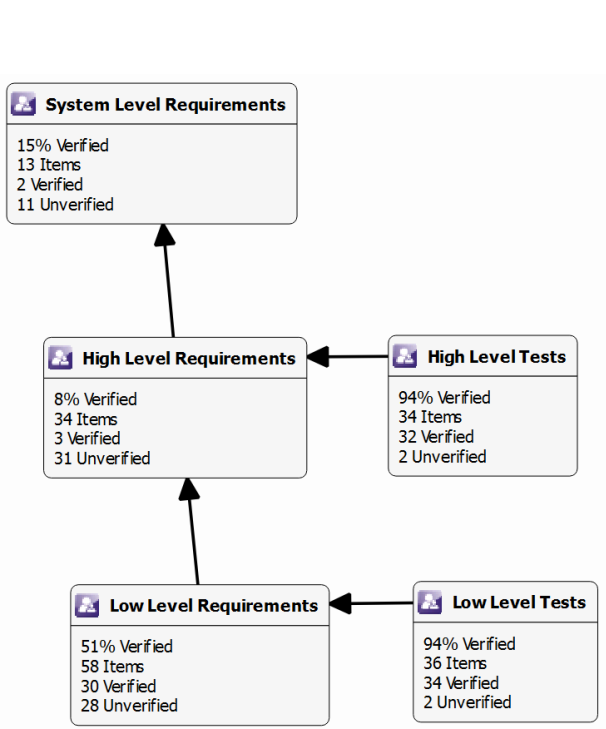
Considering Traceability Scenarios

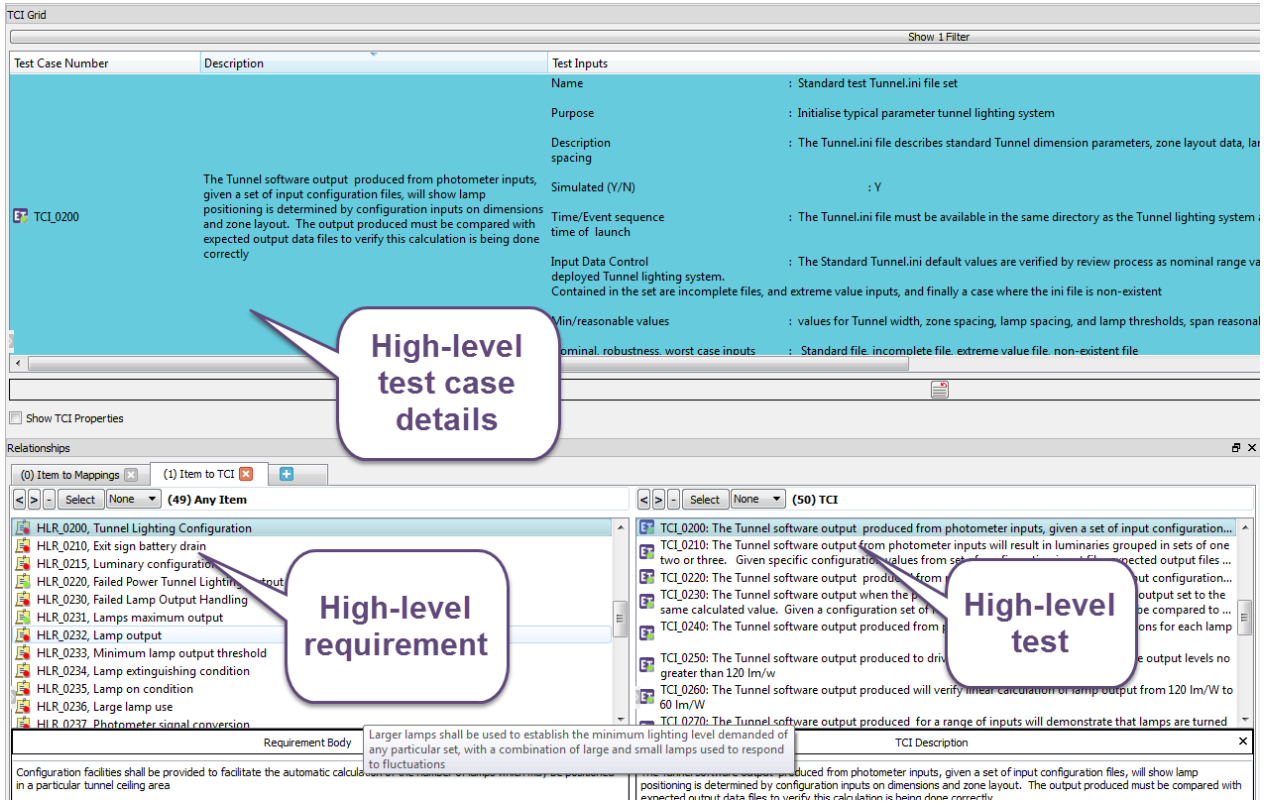


Identifying potential traceability issues

- Improper decomposition
- The all encompassing requirement
- Many to one/few could reveal poor traceability analysis

Author Link and Review Functional Tests

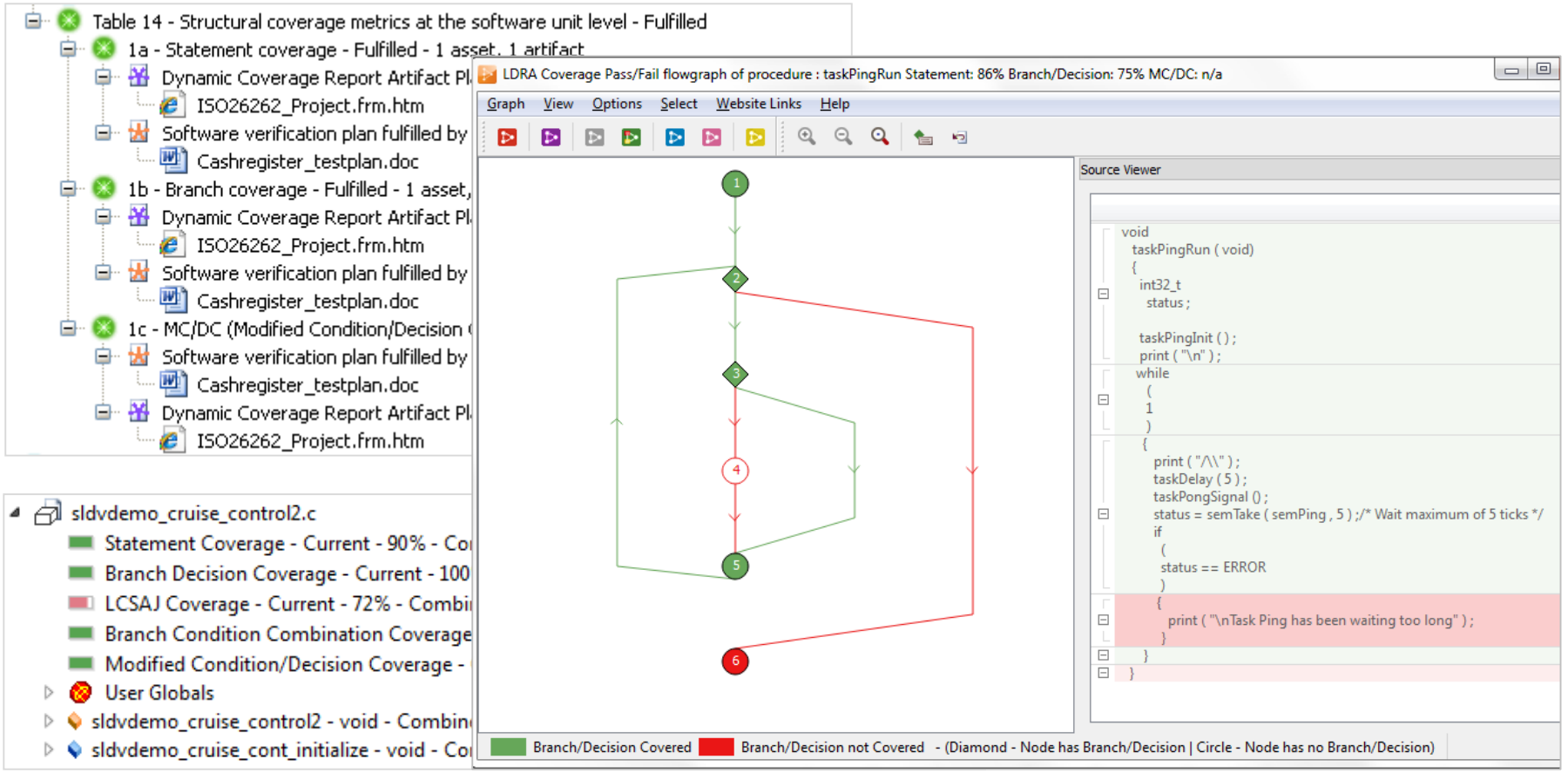




Developing and documenting test cases

- Must be linked to requirements they verify
- Test case details must be reviewed against requirements
- Function testing at the system or integration scope can managed from TBmanager

Measure Structural Coverage



Measure of Test Effectiveness

- Coverage through requirements based tests cases (functional and low-level)
- Identifies unreachable/infeasible code and gaps in requirements and test design

What is Structural Coverage?

Measurement of Test Effectiveness

- How effectively did tests exercise code?
- Exercised, entry points, statements, branches, compound conditionals, execution paths
- Systems requirement reliability levels up with one defect per 10^9 operating hours
- Metric that helps determine when a system is adequately tested

Structural Coverage is Often Mandated

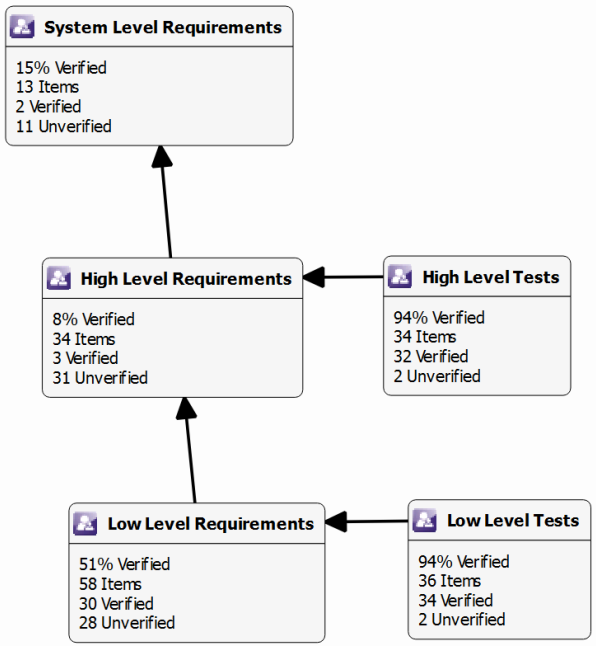
- DO-178B/C, DO-278(A) for Commercial/Defense avionics and ground systems
- IEC 61508 for industrial controls
- ISO 26262 for automotive
- IEC 62304 for medical devices
- EN 50128 for rail
- Company based standards (in-house)

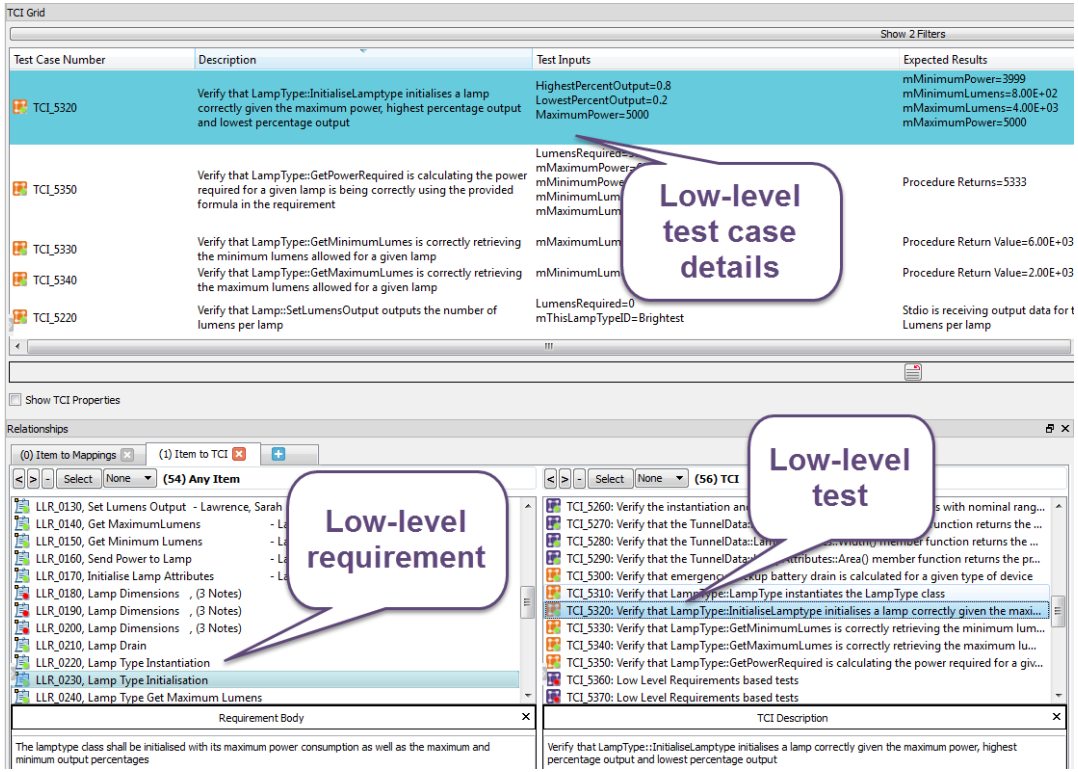
Types of Coverage

Depending on the SIL or DAL level and functional safety standard being followed, coverage requirements and required methodology varies

- Statement Coverage
- Branch Decision Coverage
- Modified Condition / Decision Coverage (MC/DC)
- Data Coupling and Control Coupling Coverage
- Object Code Coverage
- Linear Code Sequence And Jump Coverage – Test Path (LCSAJ)

Author Link and Review Low-Level Unit Tests



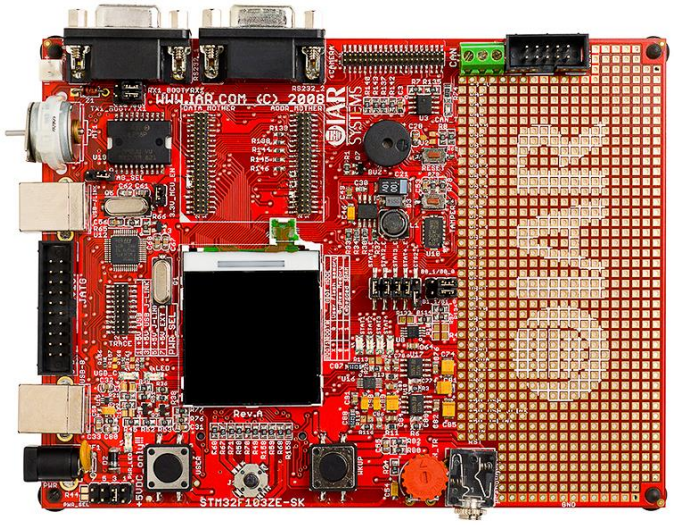


Developing and documenting low-level test cases

- Must be linked to requirements they verify
- Test case details must be reviewed against requirements
- Typically written at the function interface

Unit Testing – Host / Target

- 5.5.5 - SOFTWARE UNIT VERIFICATION
  - Input Assets
    - Software Verification Cases and Procedures fulfilled by 1 item
      - Low_Level_Tests.xlsx
  - Output Artifacts
    - TBrun Regression Report Artifact Placeholder fulfilled by 1 item
      - Cpp_tunnel_lighting_system.frm.htm
- 5.6.1 - Integrate SOFTWARE UNITS
  - Input Assets
    - Software Integration Plan fulfilled by 50 items
      - int TunnelData::LampAttributes::Width();
      - void TunnelData::Zone::InitialiseZone(Sint_32 * pZoneData, const Sint_3...
      - void TunnelData::Tunnel::AdjustPoweredLighting(const Float_64 Photo...
      - Float_64 TunnelData::SystemData::GetEmergencyLampLumens();
      - int TunnelData::LampAttributes::Height();



Test Case View

Test Case	Regression P / F	Procedure
1	PASS	integer_to_ascii
2	PASS	integer_to_ascii
3	PASS	integer_to_ascii
4	PASS	integer_to_ascii
5	PASS	integer_to_ascii
6	PASS	integer_to_ascii
7	PASS	integer_to_ascii
8	PASS	integer_to_ascii
9	PASS	integer_to_ascii

Variable I/O View

Value	Name	Type	Use	Regression Analysis
123	value	u32	Input parameter applied through local	Assigned
5	digits	u8	Input parameter applied through local	Assigned
0	blanks	u8	Input parameter applied through local	Assigned
'0'	itoa_str[0]	s8	Output global	Compare + Write
'0'	itoa_str[1]	s8	Output global	Compare + Write
'1'	itoa_str[2]	s8	Output global	Compare + Write
'2'	itoa_str[3]	s8	Output global	Compare + Write
'3'	itoa_str[4]	s8	Output global	Compare + Write
'0'	itoa_str[5]	s8	Output global	Compare + Write
'0'	itoa_str[6]	s8	Output global	Compare + Write
0	itoa_str[7]	s8	Output global	Compare + Write

Testing at the function interface

- Achieve structural coverage objectives
- Verify low level requirements

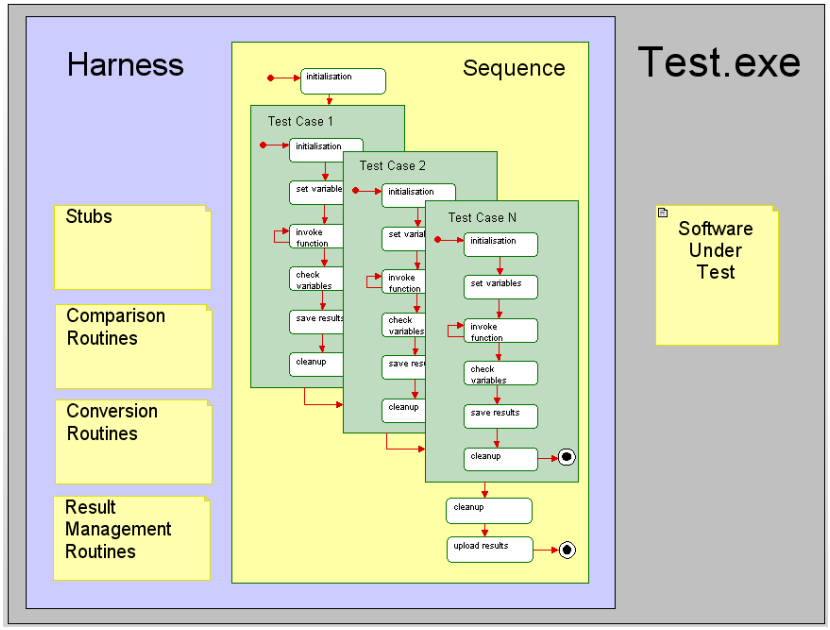
Why Unit Testing

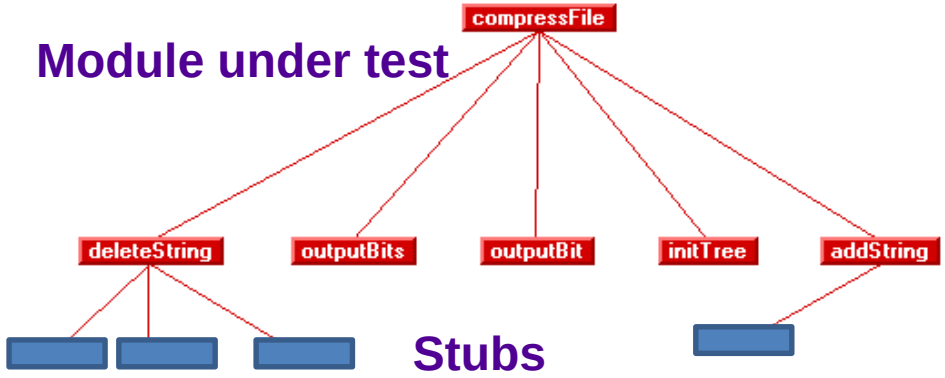
Why Unit test?

- Allows you to find faults earlier
- Allows you to characterize the behavior of components as you develop – and make sure they behave the way they are supposed to
- Allows you to verify components before they are put into systems
- Gives you confidence the system will work correctly when it all fits together

Why Unit testing with structural coverage?

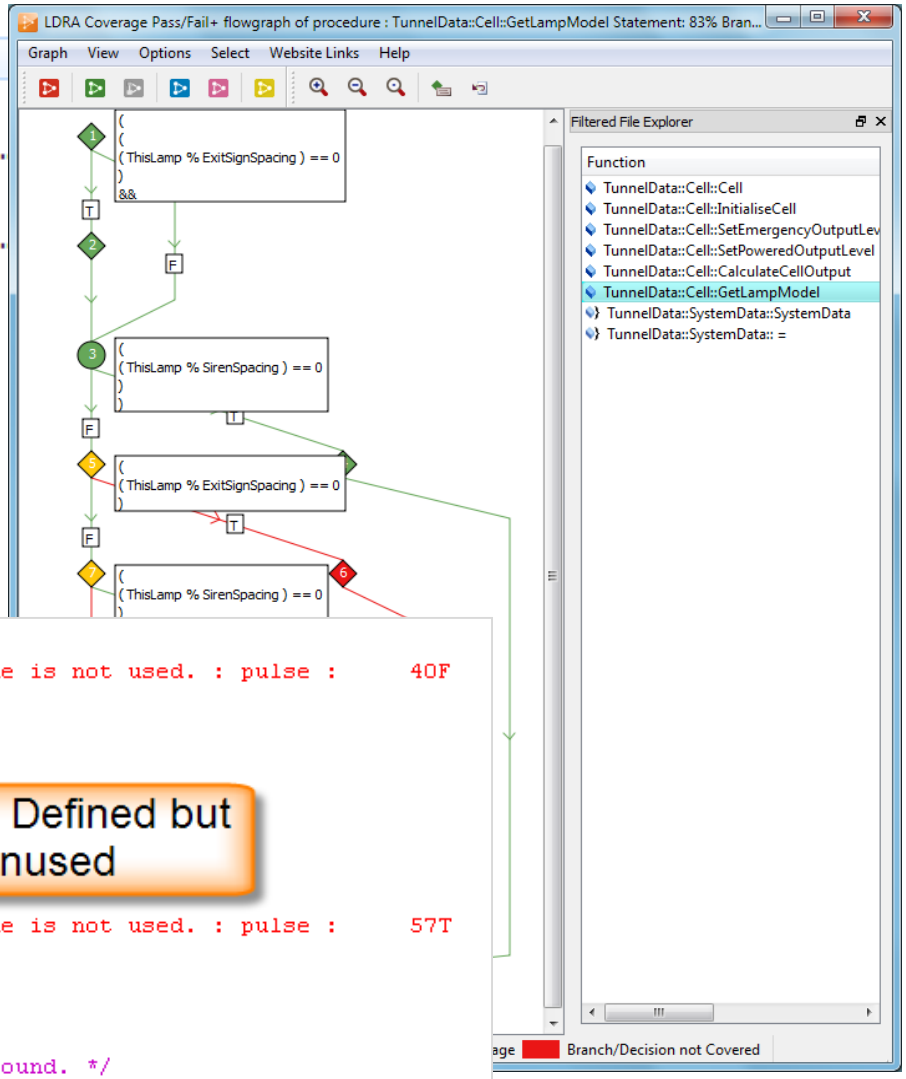
- Entry point, Statement, Branch, MC/DC, Path, assembly level coverage
- Provides a basis to know you have adequately tested





Data Flow & Control Flow Analysis

- 1e - Control flow analysis
  - Output Artifacts
    - Code Coverage Flowgraphs fulfilled by 1 item
      - Callgraph - Pass/Fail Coverage for Lampmodel...
    - Code Coverage Callgraphs fulfilled by 1 item
      - Callgraph - Pass/Fail Coverage for Lampmodel...
  - Input Assets
- 1f - Data flow analysis
  - Output Artifacts
    - Cpp_tunnel_lighting_system.frm.htm (Artifact)




```
   uint32_t pulse=0U;
/* (M) DATAFLOW VIOLATION    : 70 D : DU anomaly, variable value is not used. : pulse :     40F
/*    See also line 17 DataFlow.c(DATAFLOW) */
   uint32_t iter;

   if ( cycles > 0U )
   {
      for ( iter=0U; iter<cycles; iter++ )
      {
         pulse++;
/* (M) DATAFLOW VIOLATION    : 70 D : DU anomaly, variable value is not used. : pulse :     57T
/*    See also line 17 DataFlow.c(DATAFLOW) */
      }
   }
}
/* (O) DATAFLOW VIOLATION    :  7 D : DU  data flow anomalies found. */
```

pulse is Defined but Unused

- Data Coupling and Control Coupling

Delivering Software Quality and Security through Test, Analysis and Requirements Traceability

DC/CC from DO-178B to DO-178C

DO-178B A7-8 points to section 6.4.4.2c, states

- “The analysis should confirm the data coupling and control coupling between the code components.”

DO-178C section 6.4.4.2c states

- “Analysis to confirm that the requirements-based testing has exercised the data and control coupling between code components”

Analytical exercise vs measurement exercise

- The shift in emphasis from *confirm the coupling* (DO-178B) to *confirm the exercising of the coupling* (DO-178C) changes the DCCC objective from an analytical exercise against the test design to a measurement exercise against the test execution. This was a specific topic of conversation when the committee discussed this issue

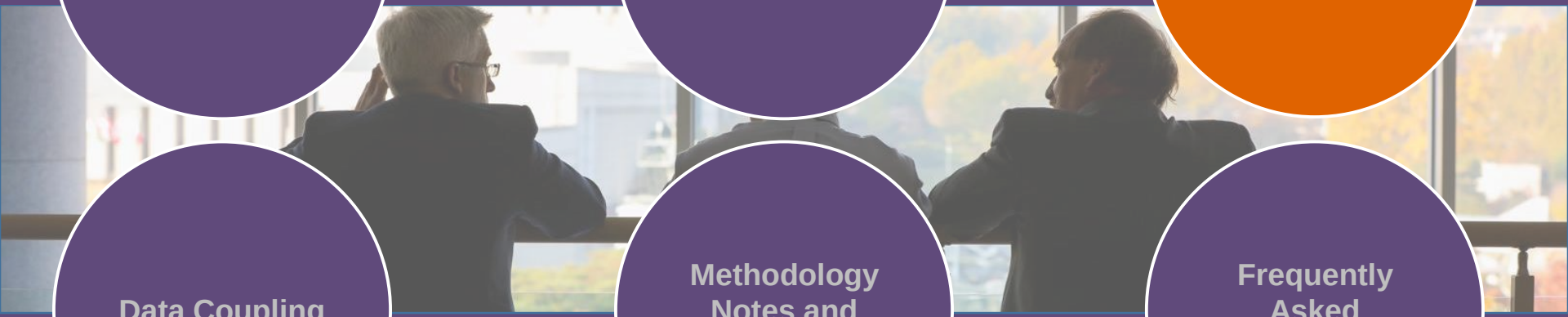
Confirming coupling

- Confirm control coupling by reviewing **procedure call coverage** achieved by requirements based tests across software component boundaries
- Confirm data coupling by reviewing **dynamic data flow coverage report** generated from requirements based tests

LDRA Overview

DO-178C Overview

Control Coupling

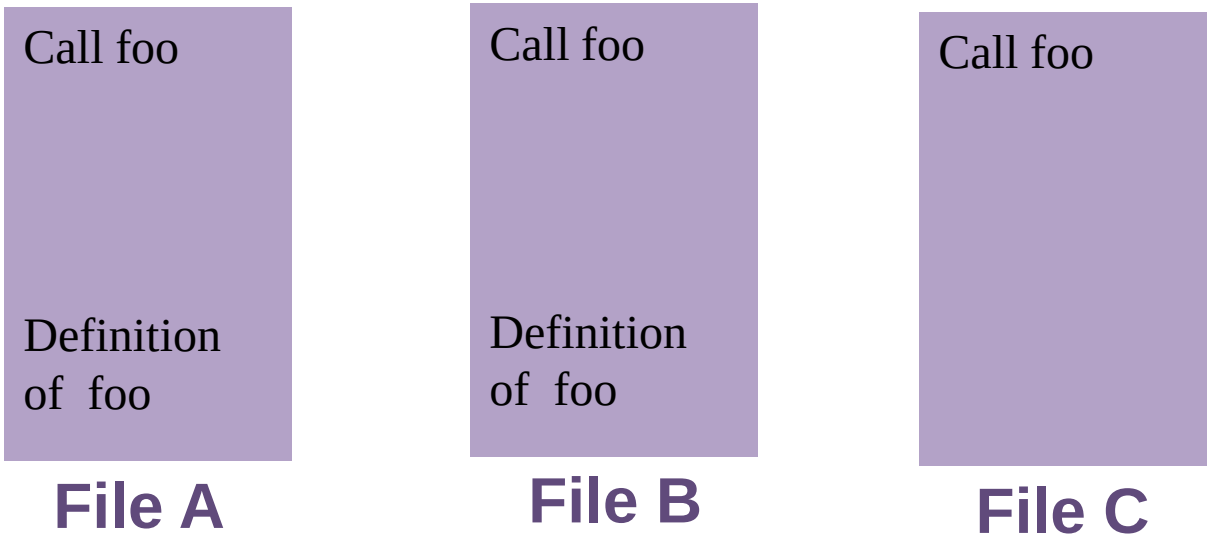


Data Coupling

Methodology Notes and Benefits

Frequently Asked Questions

Control Coupling Examples – Linking Issues (CC1)



Two of the files contain a definition of a function foo

- May resolve all calls to definition in file A
- Or calls in File A to file A, calls in file B to file B, and calls in file C to either

Control Coupling defect as user may be unaware of ambiguity

Weak Linking – Linking Issues Continued

Test1.c

```
#pragma weak g
int g()
{ return 5; }
```

Test2.c

```
#pragma weak g
int g()
{ return 3; }
```

Weak symbols are often useful in defining library functions that can be overridden

- GCC supports for Elf targets

In the example above one of the two functions will be linked

The symbol that is linked is decided arbitrarily

- Essentially picks up the symbol that is “closest”
- Procedure call coverage is the ideal way to ensure the right instance is linked.

https://gcc.gnu.org/onlinedocs/gcc/Weak-Pragmas.html

#pragma weak *symbol*

This pragma declares symbol to be weak, as if the declaration had the attribute of the same name. The pragma may appear before or after the declaration of symbol. It is not an error for symbol to never be defined at all

#pragma weak symbol1 = symbol2

This pragma declares symbol1 to be a weak alias of symbol2. It is an error if symbol2 is not defined in the current translation unit.

C++ Challenges and potential Linking Issues

LDRA

Test1.cpp

```
class c {
public:
   int getint() { return 3; }
};

int g() {
   return 5;
}
```

Test2.cpp

```
class c {
public:
   int getint() { return 5; }
};

int g();
```

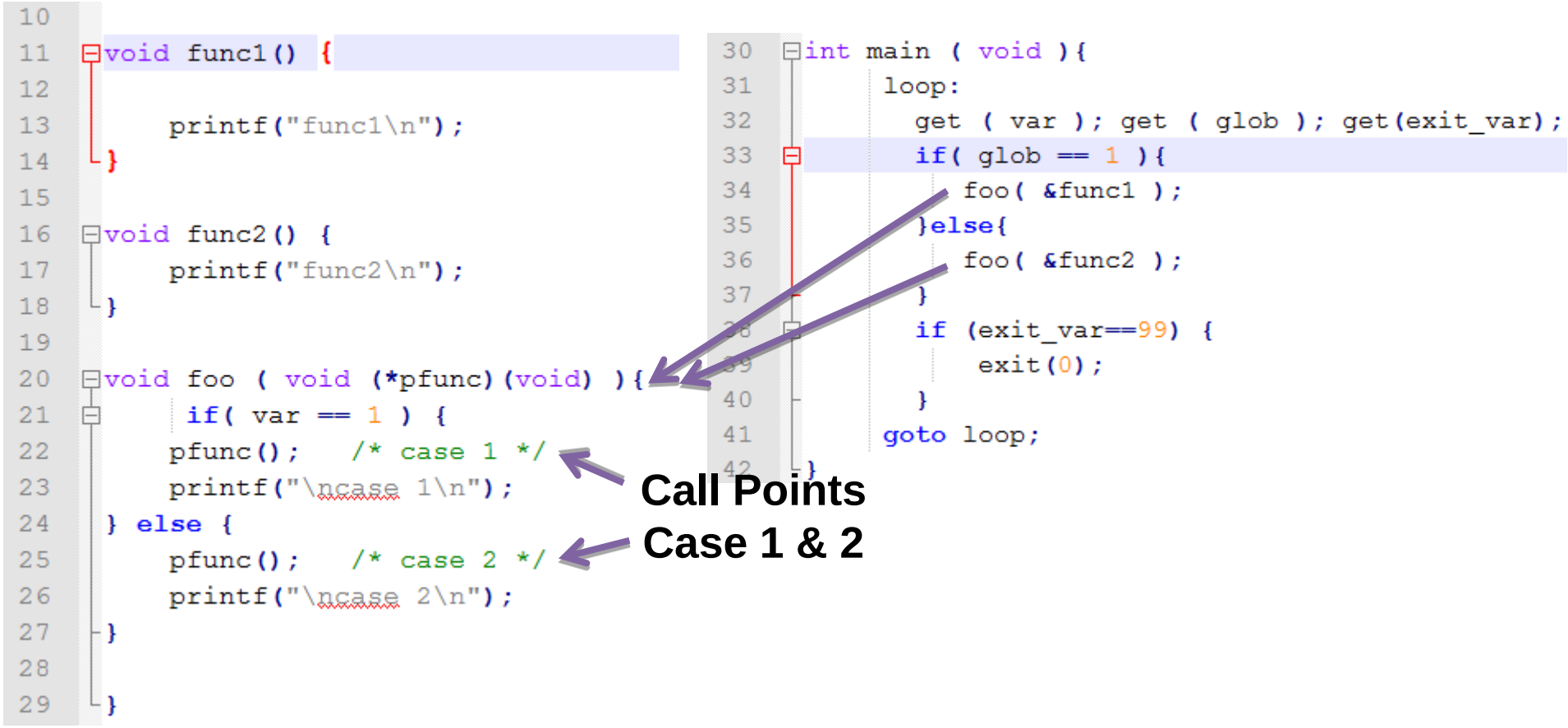
Test2.cpp

```
#include <stdio.h>
int main() {
   g();
   c a;
   printf("value of c.getint()
%d\n", a.getint());
```

Inline function defined in a class with two different implementation (Cpp example)

- Arbitrary and essentially picks up the symbol that is “closest”
- Procedure call coverage is the ideal way to ensure the right instance is linked.

Control Coupling Example – Procedure Call Coverage (CC2)



- Control Coupling requires that all potential calls be executed at each point

CC2 Data Sets

Inputs	Var value	Glob value	Call Point
Data Set 1	0	1	foo (&func1) at case 2
Data Set 2	1	0	foo (&func2) at case 1
Data Set 3	0	0	foo (&func2) at case 2
Data Set 4	1	1	foo (&func1) at case 1

```
30  int main ( void ){
31        loop:
32          get ( var ); get ( glob ); get(exit_var);
33          if( glob == 1 ){
34              foo( &func1 );
35          }else{
36              foo( &func2 );
37          }
```

```
20  void foo ( void (*pfunc)(void) ){
21        if( var == 1 ) {
22          pfunc();   /* case 1 */
23          printf("\ncase 1\n");
24      } else {
25          pfunc();   /* case 2 */
26          printf("\ncase 2\n");
```

CC2 Data Sets

```
30  int main ( void ){
31        loop:
32          get ( var ); get ( glob ); get(exit_var);
33          if( glob == 1 ){
34              foo( &func1 );
35          }else{
36              foo( &func2 );
37          }
```

```
20  void foo ( void (*pfunc)(void) ){
21        if( var == 1 ) {
22          pfunc();   /* case 1 */
23          printf("\ncase 1\n");
24        } else {
25          pfunc();   /* case 2 */
26          printf("\ncase 2\n");
```

Procedure Call/Return Trace

Data Set 1 - Case 3

REF. (SOURCE)	LEVEL	LINE CALL/RETURN	PROCEDURE NAME
65 (34)	1	CALL	foo
47 (25)	2	CALL	func1
23 (14)	2	RETURN	func1
50 (29)	1	RETURN	foo

Procedure Call/Return Trace

Data Set 3 - Case 2

REF. (SOURCE)	LEVEL	LINE CALL/RETURN	PROCEDURE NAME
71 (36)	1	CALL	foo
47 (25)	2	CALL	func2
29 (18)	2	RETURN	func2
50 (29)	1	RETURN	foo

Procedure Call/Return Trace

Data Set 2 - Case 1

REF. (SOURCE)	LEVEL	LINE CALL/RETURN	PROCEDURE NAME
71 (36)	1	CALL	foo
41 (22)	2	CALL	func2
29 (18)	2	RETURN	func2
50 (29)	1	RETURN	foo

Procedure Call/Return Trace

Data Set 4 - Case 1

REF. (SOURCE)	LEVEL	LINE CALL/RETURN	PROCEDURE NAME
65 (34)	1	CALL	foo
41 (22)	2	CALL	func1
23 (14)	2	RETURN	func1
50 (29)	1	RETURN	foo

- Streaming information captured as code executes

CC1 Data Sets 1 and 2

Procedure Call/Return Trace

REF. (SOURCE)	LINE LEVEL	CALL/RETURN	PROCEDURE NAME
65 (34)	1	CALL	foo
47 (25)	2	CALL	func1
23 (14)	2	RETURN	func1
50 (29)	1	RETURN	foo

Procedure	Statement	Branch
main	+100	+100
func1	100	[No Branches]
func2	100	[No Branches]
foo	100	100

Data Set 1

Procedure Call/Return Trace

REF. (SOURCE)	LINE LEVEL	CALL/RETURN	PROCEDURE NAME
71 (36)	1	CALL	foo
41 (22)	2	CALL	func2
29 (18)	2	RETURN	func2
50 (29)	1	RETURN	foo

Procedure/Function Call/Return Coverage (Fail)

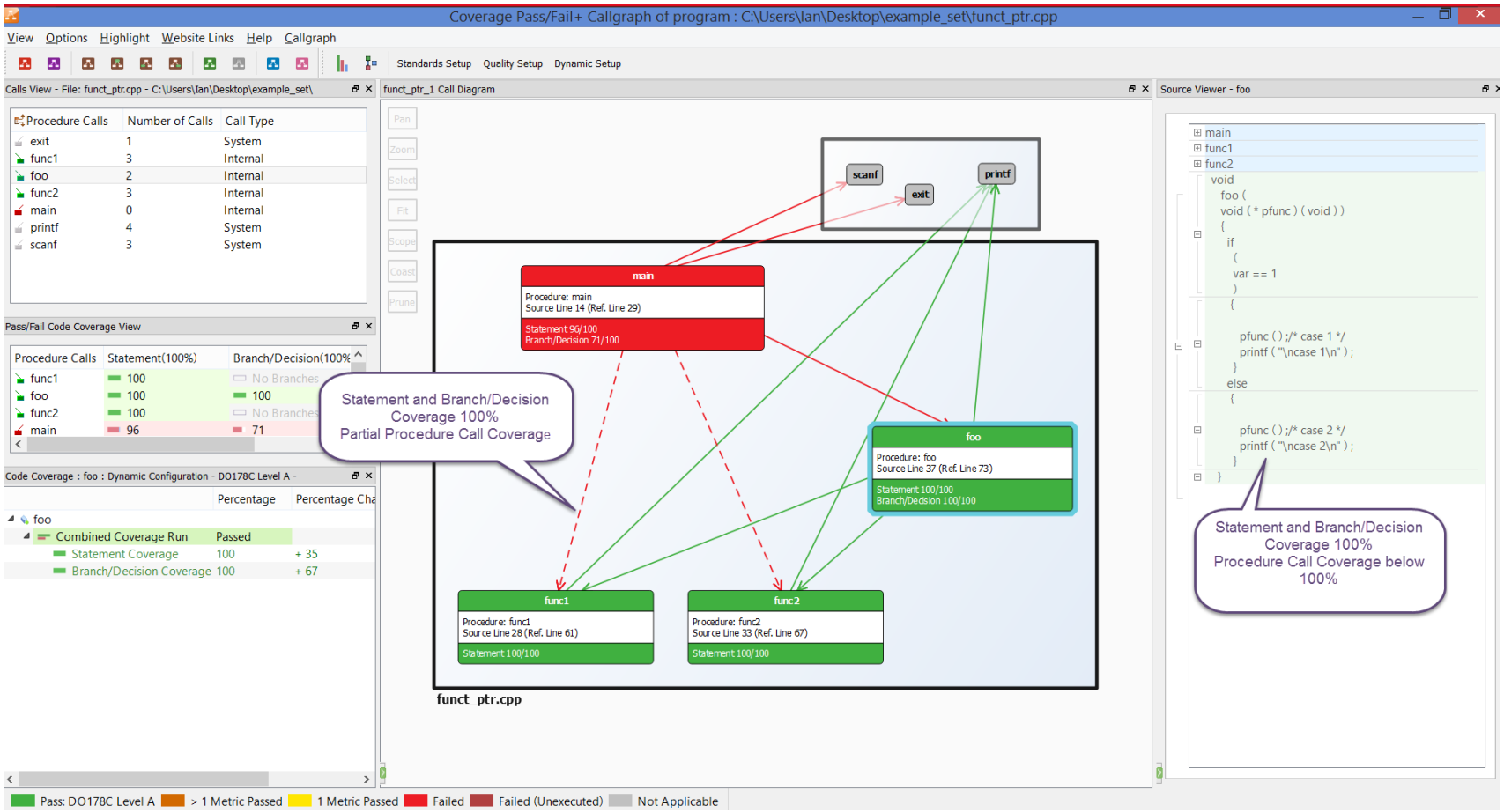
FROM LINE REF. (SOURCE)	PROCEDURE	TO LINE REF. (SOURCE)	PROCEDURE	PREVIOUS RUNS	CURRENT RUN	COMBINED
23 (14)	func1	42 (23)	foo	0 ***	0 ***	0 ***
23 (14)	func1	48 (26)	foo	0 ***	1 NNN	1
29 (18)	func2	42 (23)	foo	1	0 ***	1
29 (18)	func2	48 (26)	foo	0 ***	0 ***	0 ***
41 (22)	foo	19 (11)	func1	0 ***	0 ***	0 ***
41 (22)	foo	25 (16)	func2	1	0 ***	1
47 (25)	foo	19 (11)	func1	0 ***	1 NNN	1
47 (25)	foo	25 (16)	func2	0 ***	0 ***	0 ***
50 (29)	foo	66 (34)	main	0 ***	1 NNN	1
50 (29)	foo	72 (36)	main	1	0 ***	1
65 (34)	main	31 (20)	foo	0 ***	1 NNN	1
71 (36)	main	31 (20)	foo	1	0 ***	1
79 (39)	main	-1		1	1	2

Data Set 2

Summary	Prev. Runs	Current run	Combined
Number of Calls/Returns	13		
Number Executed	5	5	9
Number not Executed	8	8	4
Call/Return coverage (%)	**38**	**38**	**69**

- Data Sets 1 and 2 yield 100% Statement and Branch/Decision Coverage
- Less than 100% Procedure Call Coverage Achieved

Visualising Procedure Call Coverage



- Function foo has 100% Statement and Branch/Decision Coverage but all potential calls have not been executed at each point
- Dashed Red arrows from main to func1 and func2 indicate incomplete Procedure Call Coverage

CC1 Data Sets 3 and 4

Procedure Call/Return Trace

REF. (SOURCE)	LEVEL	LINE CALL/RETURN	PROCEDURE NAME
71 (36)	1	CALL	foo
47 (25)	2	CALL	func2
29 (18)	2	RETURN	func2
50 (29)	1	RETURN	foo

Data Set 3

Procedure Call/Return Trace

REF. (SOURCE)	LEVEL	LINE CALL/RETURN	PROCEDURE NAME
65 (34)	1	CALL	foo
41 (22)	2	CALL	func1
23 (14)	2	RETURN	func1
50 (29)	1	RETURN	foo

Data Set 4

Procedure	Statement	Branch
main	+100	+100
func1	100	[No Branches]
func2	100	[No Branches]
foo	100	100

Procedure/Function Call/Return Coverage (Fail)

FROM LINE REF. (SOURCE)	PROCEDURE	TO LINE REF. (SOURCE)	PROCEDURE	PREVIOUS RUNS	CURRENT RUN	COMBINED
23 (14)	func1	42 (23)	foo	0 ***	1 NNN	1
23 (14)	func1	48 (26)	foo	0 ***	0 ***	0 ***
29 (18)	func2	42 (23)	foo	0 ***	0 ***	0 ***
29 (18)	func2	48 (26)	foo	1	0 ***	1
41 (22)	foo	19 (11)	func1	0 ***	1 NNN	1
41 (22)	foo	25 (16)	func2	0 ***	0 ***	0 ***
47 (25)	foo	19 (11)	func1	0 ***	0 ***	0 ***
47 (25)	foo	25 (16)	func2	1	0 ***	1
50 (29)	foo	66 (34)	main	0 ***	1 NNN	1
50 (29)	foo	72 (36)	main	1	0 ***	1
65 (34)	main	31 (20)	foo	0 ***	1 NNN	1
71 (36)	main	31 (20)	foo	1	0 ***	1
79 (39)	main	-1		1	1	2

Summary	Prev. Runs	Current run	Combined
Number of Calls/Returns	13		
Number Executed	5	5	9
Number not Executed	8	8	4
Call/Return coverage (%)	**38**	**38**	**69**

- Data Sets 3 and 4 yield 100% Statement and Branch/Decision
- Less than 100% Procedure Call Coverage achieved

All Data Sets Combined (1 – 4)

Procedure/Function Call/Return Coverage (Pass)

FROM LINE REF. (SOURCE)	PROCEDURE	TO LINE REF. (SOURCE)	PROCEDURE	PREVIOUS RUNS		CURRENT RUN		COMBINED
23 (14)	func1	42 (23)	foo	0	***	1	NNN	1
23 (14)	func1	48 (26)	foo	1		0	***	1
29 (18)	func2	42 (23)	foo	1		0	***	1
29 (18)	func2	48 (26)	foo	1		0	***	1
41 (22)	foo	19 (11)	func1	0	***	1	NNN	1
41 (22)	foo	25 (16)	func2	1		0	***	1
47 (25)	foo	19 (11)	func1	1		0	***	1
47 (25)	foo	25 (16)	func2	1		0	***	1
50 (29)	foo	66 (34)	main	1		1		2
50 (29)	foo	72 (36)	main	2		0	***	2
65 (34)	main	31 (20)	foo	1		1		2
71 (36)	main	31 (20)	foo	2		0	***	2
79 (39)	main	-1		3		1		4

Procedure	Statement	Branch
main	+100	+100
func1	100	[No Branches]
func2	100	[No Branches]
foo	100	100

Procedure Call/Return Trace

Data Set 1 - Case 3

REF. (SOURCE)	LEVEL	CALL/RETURN	PROCEDURE NAME
65 (34)	1	CALL	foo
47 (25)	2	CALL	func1
23 (14)	2	RETURN	func1
50 (29)	1	RETURN	foo

Procedure Call/Return Trace

Data Set 3 - Case 2

REF. (SOURCE)	LEVEL	CALL/RETURN	PROCEDURE NAME
71 (36)	1	CALL	foo
47 (25)	2	CALL	func2
29 (18)	2	RETURN	func2
50 (29)	1	RETURN	foo

Procedure Call/Return Trace

Data Set 2 - Case 1

REF. (SOURCE)	LEVEL	CALL/RETURN	PROCEDURE NAME
71 (36)	1	CALL	foo
41 (22)	2	CALL	func2
29 (18)	2	RETURN	func2
50 (29)	1	RETURN	foo

Procedure Call/Return Trace

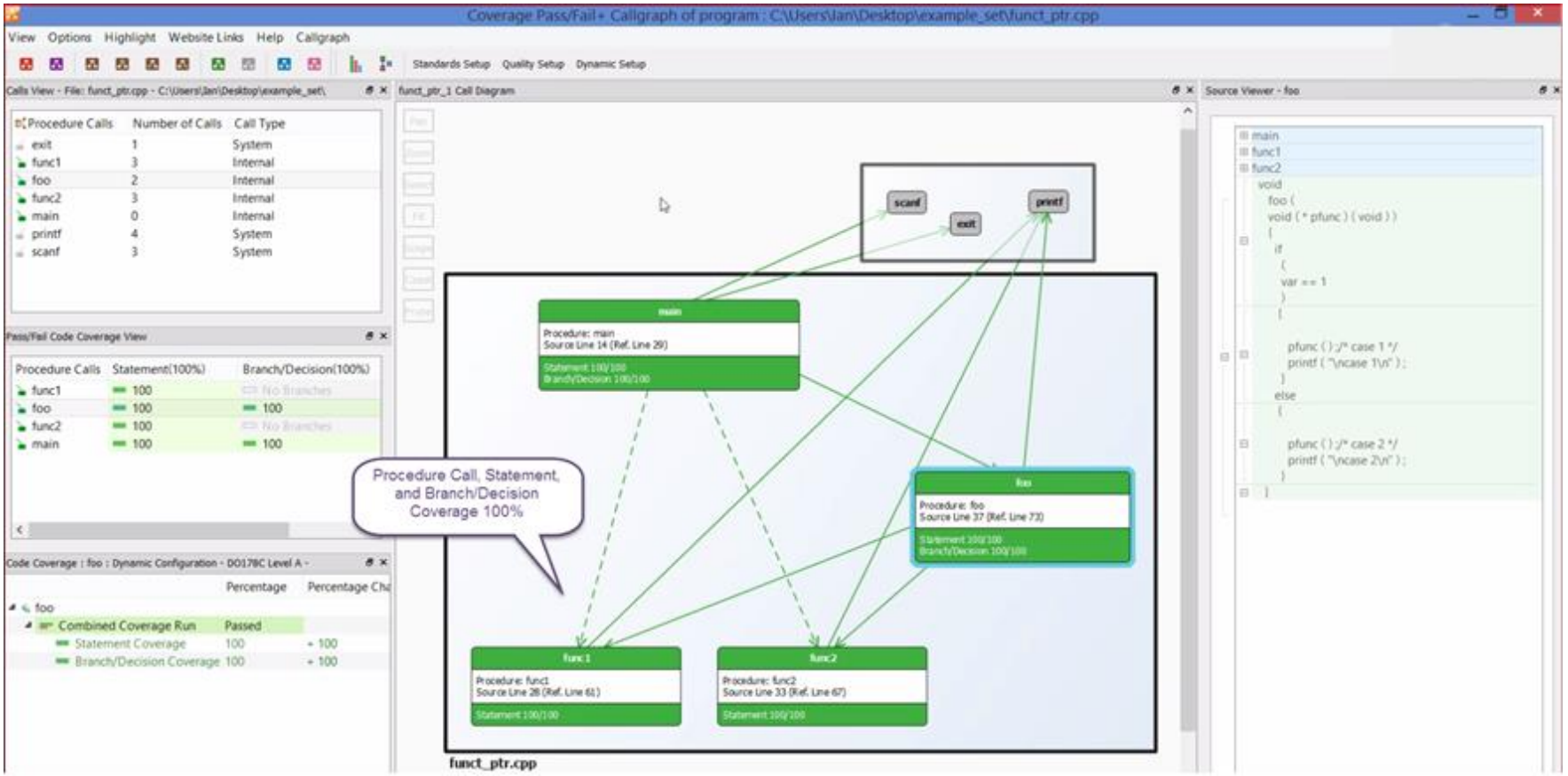
Data Set 4 - Case 1

REF. (SOURCE)	LEVEL	CALL/RETURN	PROCEDURE NAME
65 (34)	1	CALL	foo
41 (22)	2	CALL	func1
23 (14)	2	RETURN	func1
50 (29)	1	RETURN	foo

Summary	Prev. Runs	Current run	Combined
Number of Calls/Returns	13		
Number Executed	11	5	13
Number not Executed	2	8	0
Call/Return coverage (%)	**85**	**38**	**100**

- Control Coupling requires that all potential calls be executed at each point
- 100% Statement, Branch/Decision, and Procedure Call Coverage achieved

Visualising Procedure Call Coverage

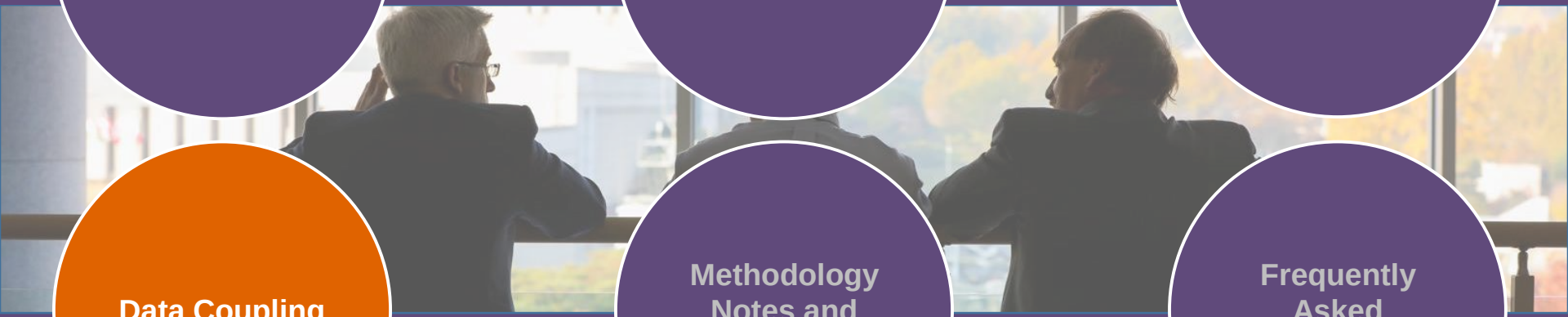


- Function foo has 100% Statement and Branch/Decision Coverage
- Callgraph arrows indicate 100% Procedure Call Coverage has been met

LDRA Overview

DO-178C Overview

Control Coupling



Data Coupling

Methodology Notes and Benefits

Frequently Asked Questions

Data Coupling Analysis Scenarios and Examples

RTCA, Inc.
1150 18th Street, NW, Suite 910
Washington, D.C. 20036

Supporting Information for DO-178C and DO-278A

RTCA DO-248C
December 13, 2011

Prepared by: SC-205

- *"Data coupling - The dependence of a software component on data not exclusively under the control of that software component."*
- *"Control coupling - The manner or degree by which one software component influences the execution of another software component."*

The verification of data and control coupling involves a combination of the following:

- Reviews and analysis of software architecture, as stated in DO-178C/DO-278A section 6.3.3.b.
- Reviews and analysis of Source Code, as stated in DO-178C/DO-278A section 6.3.4.b.
- Requirements-based testing, confirmed by structural coverage analysis, as stated in DO-178C/DO-278A section 6.4.4.d.

The focus of this FAQ is on the objective stated in DO-178C/DO-278A section 6.4.4.d, which is:

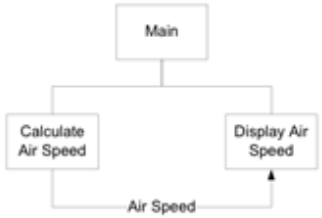
- *"Test coverage of software structure, both data coupling and control coupling, is achieved."*

The related activity described in DO-178C/DO-278A section 6.4.4.2.c is:

- *"Analysis to confirm that the requirements-based testing has exercised the data and control coupling between code components."*

The intent behind this objective is to ensure that applicants perform a sufficient amount of hardware/software integration testing and/or software integration testing.

The example below shows three components: a subprogram "Main", which calls two subprograms "Calculate Air Speed" and "Display Air Speed". The calculated air speed is passed between the two subprograms using a global variable "Air Speed".



ain" and "Calculate Air Speed", Speed". There also exists data Speed".

tion 6.4.4.d, all of the following

peed".

ed".

l" to "Display Air Speed".

ame test.

h of section 3.2 of DO-178C ***ly developed software used in a*** **'8A states that "*Component X* *in an approved system.*") Is** **l in the context of a previously**

DS; reuse; software life cycle

llustrate a sequence of software software product with different use of software, refer to DO- previously developed software

sign process feedback to the 278A, where feedback to the ocess seems adequate?

planning process; software

or incorrect inputs detected o the system life cycle process, nning process as feedback for d are added for emphasis.)

o the software planning process

Data Coupling Analysis by Test Case Example

- CalculateAirspeed and DisplayAirspeed are both invoked by runAirspeedCommand
- All three are in different files and represent different software components
- Test cases are created to verify commands are being set and achieve structural coverage

```
static S_U32 airspeed;

#define CALCULATE_CMD 1
#define DISPLAY_CMD   2

/*
 * Data Coupling defects:
 * 1) It is possible to call displayAirspeed (a use operation of 'airspeed')
 *    before calculateAirspeed (a set operation of 'airspeed').
 *    Demonstrated with Test Case 1
 *
 * 2) It is possible to call calculateAirspeed without a subsequent call to
 *    displayAirspeed
 *    Demonstrated with Test Case 2
 */

void runAirspeedCommand  (S_U16 command)
{
   switch(command)
   {
      case CALCULATE_CMD:
      calculateAirspeed (airspeed);
      break;
      case DISPLAY_CMD:
      displayAirspeed (airspeed);
      break;
   }
}
```

The requirements states that CALCULATE_CMD is called before DISPLAY_CMD every time

Test Case View

Test Case	Procedure	Regression P / F	Name / Description
Tc 1	runAirspeedCommand	PASS	Inovke DISPLAY_CMD so that the current value of airs...
Tc 2	runAirspeedCommand	PASS	Inovke CALCULATE_CMD so that the airspeed is calc...
Tc 3	runAirspeedCommand	PASS	Inovke a command value outside of DISPLAY_CMD an...

Testing Requirements to Achieve Coverage

- In order to get 100% statement and 100% decision coverage of the “C” code, we needed to create three test cases that verify the requirement

Variable I/O View

Value	Name	Type
CALCULATE_CMD	command	S_U16
*** Value Retained ***	airspeed	S_U32
0	airspeed	S_U32

Variable I/O View

Value	Name	Type
DISPLAY_CMD	command	S_U16
*** Value Retained ***	airspeed	S_U32

Variable I/O View

Value	Name	Type
*** Value Retained ***	airspeed	S_U32
0	command	S_U16
0	airspeed	S_U32

- AirspeedCommands.cpp
  - Statement Coverage - Current - 100% - Combined - 100%
  - Branch Decision Coverage - Current - 100% - Combined - 100%
  - LCSAJ Coverage - Current - 100% - Combined - 100%
  - Global Variables
  - runAirspeedCommand - void - Combined - S 100% - B 100% - L 100%

Test Case View

Test Case	Procedure	Regression P / F	Name / Description
Tc 1	runAirspeedCommand	PASS	Inovke DISPLAY_CMD so that the current value of airs...
Tc 2	runAirspeedCommand	PASS	Inovke CALCULATE_CMD so that the airspeed is calc...
Tc 3	runAirspeedCommand	PASS	Inovke a command value outside of DISPLAY_CMD an...

Statement, Branch, and Data Coverage Achieved

LINE NUMBER REF. (SOURCE)	STATEMENT	PREVIOUS RUNS	CURRENT RUN	COMBINED
94 (31)	void	-	-	-
95	runAirspeedCommand (	6	3	9
96	S_U16 command)	-	-	-
97 (32)	{	-	-	-

LINE NUMBERS: FROM	REFORMATTED (SOURCE) TO	PREVIOUS RUNS	CURRENT RUN	COMBINED	CODE PRECEDING DECISION POINT
98 (33)					
99					
100					
101 (34)					
102 (35)	101 (34) 102 (35)	2	1	3	command) {
103 (36)	101 (34) 105 (38)	2	1	3	
104 (37)	101 (34) 109 (42)	2	1	3	
105 (38)					
106 (39)	104 (37) 109 (42)	2	1	3	break ;
107 (40)					
108 (41)	107 (40) 109 (42)	2	1	3	break ;
109 (42)	}				

	Call Depth / Parameter Name					
Variable Name	Alias	File	Procedure	Type Code	Attribute Code	Used on lines...
airspeed		AirspeedCommands.cpp	runAirspeedCommand	G	R	39
				G	D Definition	36
command		AirspeedCommands.cpp	runAirspeedCommand	P	E	31
				P	R	33
factor		AirspeedCalculate.cpp	calculateAirspeed	G	R	16
sensorReading		AirspeedCalculate.cpp	calculate		R	16
speed		AirspeedCalculate.cpp	calcul			14
						16
		AirspeedDisplay.cpp	displ			12
						14
						14

In aggregate the Dynamic Data Flow Coverage Report shows all data elements have been read and written to as expected

DDFC By Test Case Reveals Control Flow Issues

Test case

Variable I/O View

Value	Name	Type
CALCULATE_CMD	command	S_U16
*** Value Retained ***	airspeed	S_U32
0	airspeed	S_U32

Unexecuted code for the given test case

```
31  void runAirspeedCommand  (S_U16 command)
32  {
33      switch(command)
34      {
35          case CALCULATE_CMD:
36          calculateAirspeed (airspeed);
37          break;
38          case DISPLAY_CMD:
39          displayAirspeed (airspeed);
40          break;
41      }
42  }
```

```
void
 runAirspeedCommand (
 S_U16 command )
 {
  switch (
   command
   )
   {
   case 1 :
     calculateAirspeed ( airspeed ) ;
     break ;
   case 2 :
     displayAirspeed ( airspeed ) ;
     break ;
   }
 }
```

Unexecuted data reference for the given test case

	Call Depth / Parameter Name					
Variable Name	**Alias**	**File**	**Procedure**	**Type Code**	**Attribute Code**	**Used on lines...**
airspeed		AirspeedCommands.cpp	runAirspeedCommand	G	R	39 *****
						36
command		AirspeedCommands.cpp	runAirspeedCo			31
						33
factor		AirspeedCalculate.cpp	calculateAirspeed			16

On line 39 the reference to airspeed by displayAirspeed is not executed with this test case

DDFC By Test Case Reveals Control Flow Issues

Test case

Variable I/O View

Value	Name	Type
DISPLAY_CMD	command	S_U16
*** Value Retained ***	airspeed	S_U32
0	airspeed	S_U32

Unexecuted code for the given test case

```
31  void runAirspeedCommand  (S_U16 command)
32  {
33      switch(command)
34      {
35          case CALCULATE_CMD:
36          calculateAirspeed (airspeed);
37          break;
38          case DISPLAY_CMD:
39          displayAirspeed (airspeed);
40          break;
41      }
42  }
```

```
void
    runAirspeedCommand (
    S_U16 command )
    {
        switch (
            command
            )
            {
            case 1 :
                calculateAirspeed ( airspeed ) ;
                break ;
            case 2 :
                displayAirspeed ( airspeed ) ;
                break ;
            }
    }
```

Unexecuted data reference for the given test case

	Call Depth / Parameter Name					
Variable Name	Alias	File	Procedure	Type Code	Attribute Code	Used on lines...
airspeed		AirspeedCommands.cpp	runAirspeedCommand	G	R	39
				G	D	36 *****
command					E	31
				P	R	33
factor				G	R	16 *****

On line 36 the define of airspeed by calculateeAirspeed is not executed with this test case

Dynamic Data Flow Coverage

DO-178C Text

- “Analysis to confirm that the **requirements-based testing has exercised the data and control coupling** between code components”
- “**Test Coverage** of software structure, both **data and control coupling**, is achieved”
- **A measurement exercise against the test execution**

A Form of Test Measurement

- Analogous to structural coverage but focus on data set and usage

Scope and Granularity

- Measured in aggregate or by test case
- Measured after execution of high level and/or low-level tests

Data Coupling for Non-parametric Globals

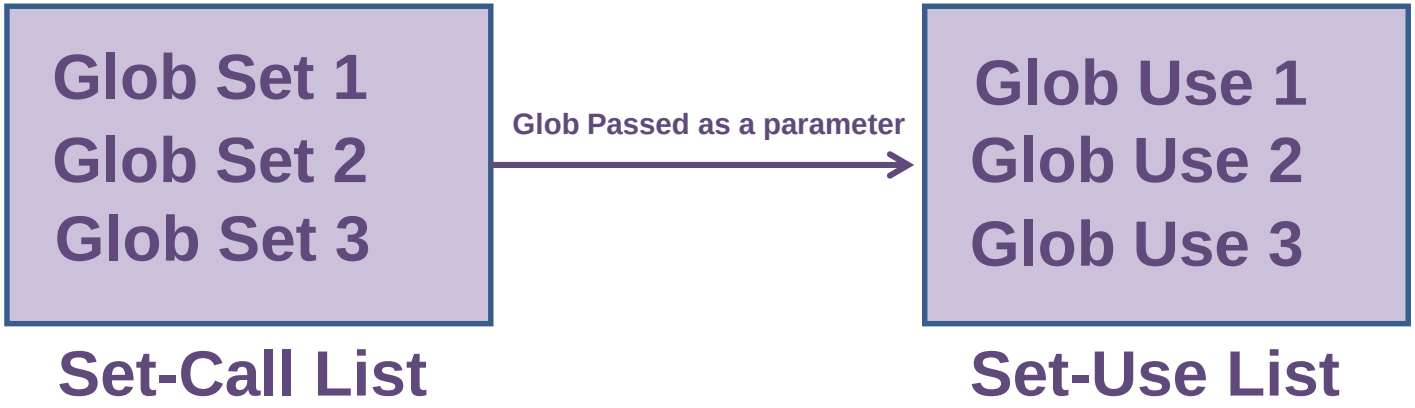
```
-------------------------------------------------------------------------------

Global Variable Usage Summary
-----------------------------

=======================================================
NAME            FILE          ATTRIB      USED ON LINE
=======================================================
 NumZones       CELL.CPP        G  D
                                G  E
                SYSTEMDATA.CPP  G  D
                                G  E
                TUNNEL.CPP      G  D
                                G  R
                                G  E
                ZONE.CPP        G  D
                                G  E
 NumZoneParams  CELL.CPP        G  D
                                G  E
                SYSTEMDATA.CPP  G  D
                                G  E
                TUNNEL.CPP      G  D
                                G  E
                ZONE.CPP        G  D
                                G  E
 NumSystemParams
                CELL.CPP        G  D
                                G  E
                SYSTEMDATA.CPP  G  D
                                G  E
                TUNNEL.CPP      G  D
                                G  E
                ZONE.CPP        G  D
                                G  E
 NumLampTypes   CELL.CPP        G  D
                                G  R
                                G  R
                                G  E
                SYSTEMDATA.CPP  G  D
                                G  R
                                G  E
                TUNNEL.CPP      G  D
                                G  R
                                G  E
                ZONE.CPP        G  D
                                G  R
                                G  E
 LampsPerLuminaire
                CELL.CPP        G  D
                                G  R
                                G  E
```

	Call Depth / Parameter Name						
Variable Name	Alias	File	Procedure	Type Code	Attribute Code	Used on lines...	
MaxCellsPerZone		Config.h	main	G	E	19	
				G	R	37	
				G Global	D	19	
MaxLuminairesPerCell		Config.h	main	G	E	20	
				G	R	24	31
				G	D	20	
NumLampTypes		Cell.cpp	TunnelData::Cell::Cell	G	R	24 *****	
			TunnelData::Cell::InitialiseCell	G	R	43 *****	
						73 *****	
			TunnelData::Cell::SetPoweredOutputLevel	G	R	112	113
						122 *****	
						137 *****	
		Config.h	main	G	E	17	
				G	R	20	21 27 28 29
				G	D	17	
		Systemdata.cpp	TunnelData::SystemData::InitialiseParams	G	R	52 *****	
			TunnelData::SystemData::SystemData	G	R	34 *****	
NumSystemParams		Datain.cpp	TunnelData::DataIn::GetData	G	R	56	
						65 *****	
		Config.h	main	G	E	16	
				G	D	16	
NumZoneParams		Datain.cpp	TunnelData::DataIn::GetData	G	R	82	
						87 *****	
			TunnelData::DataIn::ReadContent	G	R	122 *****	
		Config.h	main	G	E	15	
				G	D	15	
NumZones		Datain.cpp	TunnelData::DataIn::GetData	G	R	115	
		Config.h	main	G	E	14	
				G	R	17	
				G	D	14	
		Tunnel.cpp	TunnelData::Tunnel::AdjustEmergencyLighting	G	R	53 *****	
			TunnelData::Tunnel::AdjustPoweredLighting	G	R	43 *****	

```
                                G  E          17 (Config.h)
                                G  D          17 (Config.h)
                                G  R         20 (Systemdata)    21 (Systemdat
                                G  E          17 (Config.h)
                                G  D          18 (Config.h)
                                G  R         24 (Cell.h)    31 (Cell.h)     5
                                G  E          18 (Config.h)
```


- Generated global variables usage data can be used to filter DC/CC artifacts
- Results can be analysed in the context of requirements based tests

Data Coupling for Parametric Globals



Variable Fan-in/Fan-out Pairs

Set 1	Set 2	Set 3
Use 1	Use 2	Use 3

Variable Fan-in/Fan-out coverage (3 Sets and 3 Use)

- **Execute every Set of the Set-Call list and every Use in the Set-Use list**
- **Variable Fan-in/Fan-out coverage is practically approachable much like MC/DC coverage (n+1 number of test cases)**

Review of Fan-in/Fan-out Coverage in the context of requirements based tests to meet objective A-7.8

Parametric Global

Dynamic Data Flow Table

	Call Depth / Parameter Name					
Variable Name	Alias	File	Procedure	Type Code	Attribute Code	Used on lines...
choice		param_use.c	call_setter	P	E	38
				P	R	40
			call_use	P	E	54
				P	R	56 *****
glob_var		param_use.c	call_setter	G	D	42
						45 *****
						48 *****
	1 *(param)	param_use.c	param_setter1	P	D	7

```
38  void call_setter(int choice)
39  {
40  switch(choice) {
41  case 1:
42  param_setter1(&glob_var);
```

```
5   void param_setter1(int *param)
6   {
7   *param = 1;
8   }
9
```

glob_var		param_use.c	call_use	G	R	58
						61 *****
						64 *****
	1 param	param_use.c	param_use1	P	E	20
				P	R	22

```
54  void call_use(int choice)
55  {
56  switch(choice) {
57  case 1:
58  param_use1(glob_var);
```

```
20  void param_use1(int param)
21  {
22  int display_value = param;
23  printf("Global Variable Use 1 : %d\n",display_value);
24  }
```

- Test Case 1
  - Glob Var is passed in call_setter to param_setter1 for setting
  - Glob Var is set in param_setter1
- Test Case 2
  - Glob Var is passed in call_user to param_use1 and is referenced
  - Glob Var is set in param_setter1

- Variable Fan-in/Fan-out Coverage of parametric global glob_var
  - By reference for set and by value for use
  - NOTE: Add test cases in hidden slide

Parametric Global

Test Case View

Test Case	Regression P / F	Procedure	Name / Description	File Name
Tc 1	PASS	call_setter		param_use.c
Tc 2	PASS	call_use		param_use.c
Tc 3	PASS	call_setter		param_use.c
Tc 4	PASS	call_use		param_use.c
Tc 5	PASS	call_setter		param_use.c
Tc 6	PASS	call_use		param_use.c

- Execution of the six text cases above achieve 100% Variable Fan-In/Fan-Out Coverage

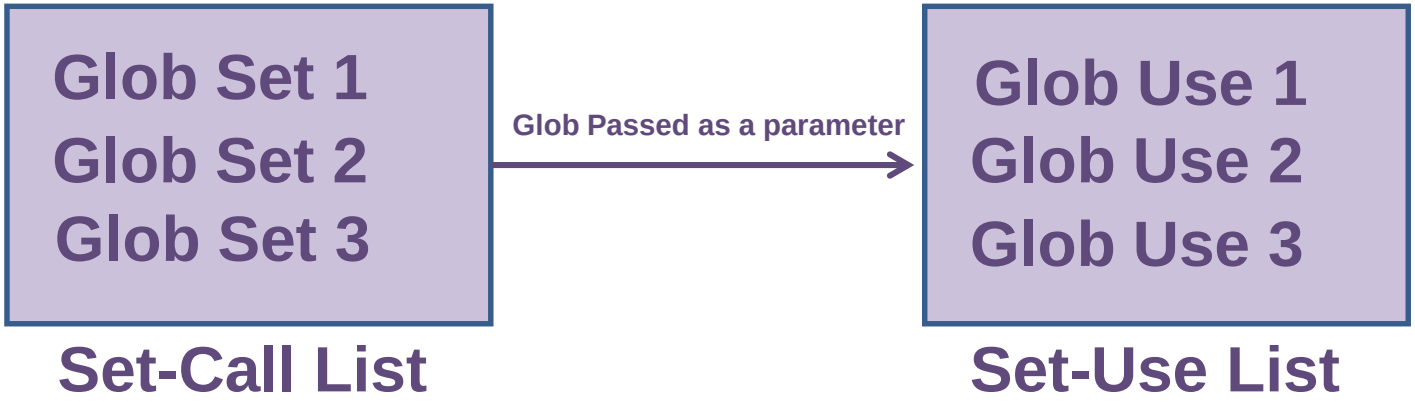
glob_var		param_use.c	call_setter	G	D	42
	1 *(param)	param_use.c	param_setter1	P	D	7
	1 *(param)		param_setter2	P	D	12
	1 *(param)		param_setter3	P	D	17

- Glob_var being set in param_setter (1..3), the Set-Call list

glob_var		param_use.c	call_use	G	R	58
	1 param	param_use.c	param_use1	P	E	20
				P	R	22
	1 param		param_use2	P	E	26
				P	R	28
	1 param		param_use3	P	E	32
				P	R	34

- Glob_var being used in param_use (1..3), the Set-Use list

Data Coupling for Parametric Globals



Set-Use combinations

Set 1	Set 2	Set 3	Set 1	Set 2	Set 3	Set 1	Set 2	Set 3
Use 1	Use 1	Use 1	Use 2	Use 2	Use 2	Use 3	Use 3	Use 3

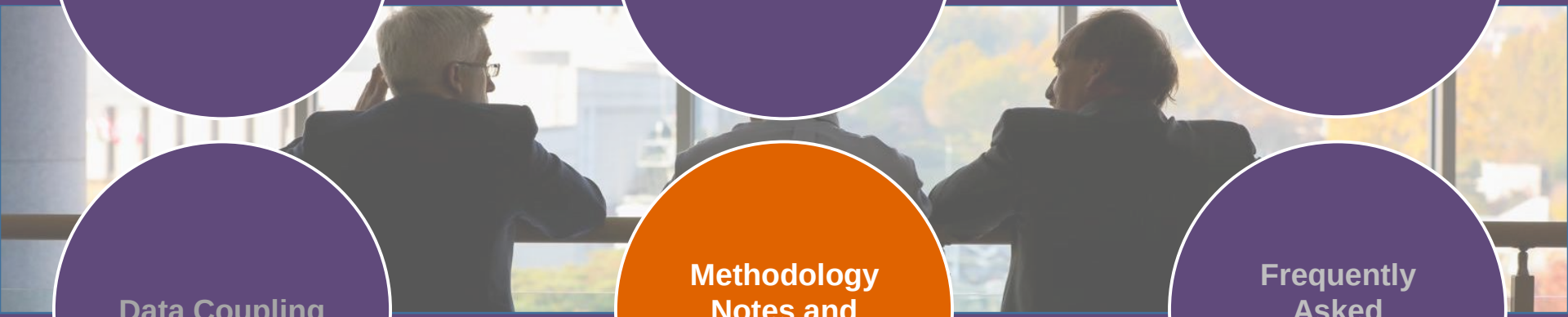
Set-Use combinations (9 Set and 9 Use)

- Ensure that the all the combinations of the Set-Use pairs in these lists are executed (9 in total)
- Set-Use combinations results in a combinatorial explosion much like Branch Combination Coverage (2^n number of test cases)

LDRA Overview

DO-178C Overview

Control Coupling



Data Coupling

Methodology Notes and Benefits

Frequently Asked Questions

The Benefits of Using LDRA Tools for DDFC

Dramatic reduction of time necessary for DCCC analysis

Clear, repeatable, methodology for DCCC that has been reviewed and accepted by DERs

- Reduces risks of methodology ambiguities during SOI audits
- Consistent with the expectations DO-178C as a test measurement exercise

Defined artifact set for archival and review

By utilising a DO-178C harmonised DDFC qualification package, the review burden for this process is vastly reduced

Reduced cost of DDCC activities during incremental releases

LDRA Overview

DO-178C Overview

Control Coupling

Data Coupling

Methodology Notes and Benefits

Frequently Asked Questions

Data Coupling Objective A-7.8 Evidence

Question: From Certification perspective, what are the artifacts to be produced as an evidence to the A-7.8 Data Coupling and Control Coupling objective?

- *Data Coupling - Dynamic Data Flow Coverage Report and methodology as described in the software verification plan (SVP), showing that results are reviewed against requirements based tests to ensure global and parametric variable set/use pairs are clearly understood and exercised.*

- *Control Coupling - Procedure Call Coverage Report – Ensuring 100% procedure call coverage. Review of the procedure call coverage report:*
  - *To ensure the linking phase has resolved all calls correctly. Potential duplicate definitions and other linking issues should be considered*
  - *Ensure all potential calls are executed at each point. Function pointers can be particularly problematic.*

Structural Coverage and DC/CC

Question: If one achieves 100% Test Coverage for High Level Requirements and Low Level Requirements, and 100% Structural Coverage for Statement and Branch/Decision Coverage from Requirements Based Tests, is this not sufficient to take credit for A-7.8?

- *Data Coupling – "Test coverage of software structure, both data coupling and control coupling…" (6.4.4 d), requires a test measurement of data coupling. 100% Structural Coverage Data doesn't ensure that all relevant set/use pairs of data elements are exercised per requirements, especially global variables passed as parameters and "used" further downstream in the call tree.*

- *Additionally 100% Statement and Branch/Decision coverage doesn't necessarily imply 100% Procedure Call Coverage, which is necessary to ensure that all potential calls are executed at each point.*

- *The specifics of the methodology used to meet objective A-7.8 should be documented in the SVP, and typically communicated and agreed upon up front with your DER.*

DER FAA/EASA Feedback

Question: Did you present your approach to any FAA/EASA or Certification DER representative, if so, what was their feedback?

- *Feedback has been consistently positive. This is an area where tooling was lacking when DO-178B was written and this resulted in a lot of manual effort and confusion. DO-178C's changes, more clearly reflect the committees original intentions. When the Dynamic data flow coverage report has been shown to DERs, ACOs, and other FAA/EASA leadership, LDRA's has been told that this type of transparent reporting of test measurement of data definitions and references (set/use operations) aligns very well with the intent of the standard. Additionally we received many comments that this approach and technology saves both the applicant and DER significant time and effort to meet the objective. The qualification package was added to cost effectively qualify the DC/CC analysis and reporting produced by the LDRA tool suite and further reduce the review effort required.*

Are there any

Questions?

For further information:

www.ldra.com info@ldra.com

@ldra_technology

LDRA Software Technology

LDRA Limited