8 Key Aspects of ED-12C / DO-178C Compliance

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

8 Key Aspects of ED-12C / DO-178C Compliance

LDRA

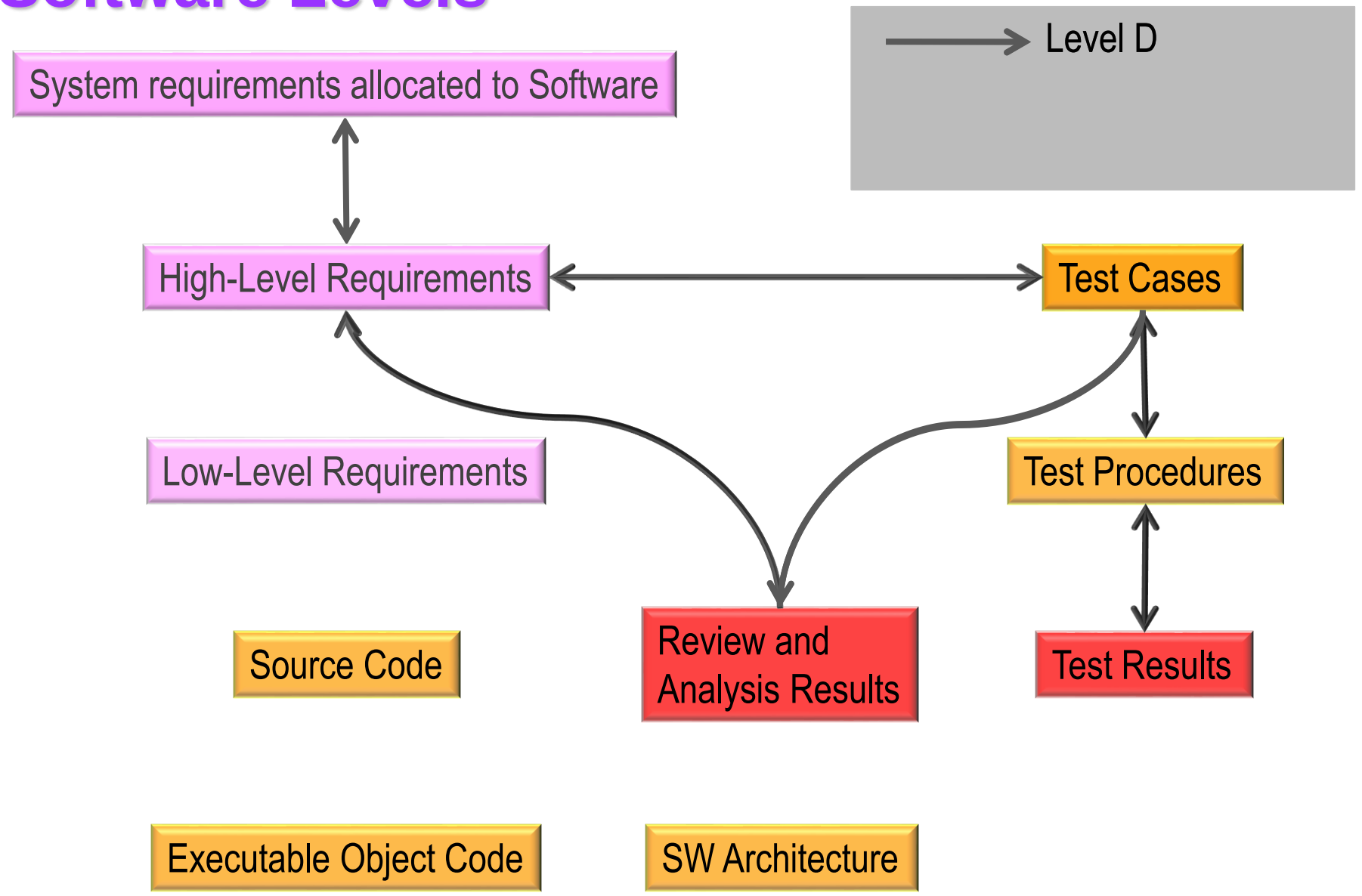
1. Traceability Management
2. Coding Standards
3. Model Based Development and Verification
4. Subclass Verification for Local Type Consistency
5. Data & Control Coupling Coverage
6. Object Code Verification
7. Compliance Knowledgebase
8. Compliance Management System

Aspect #1:
Traceability Management

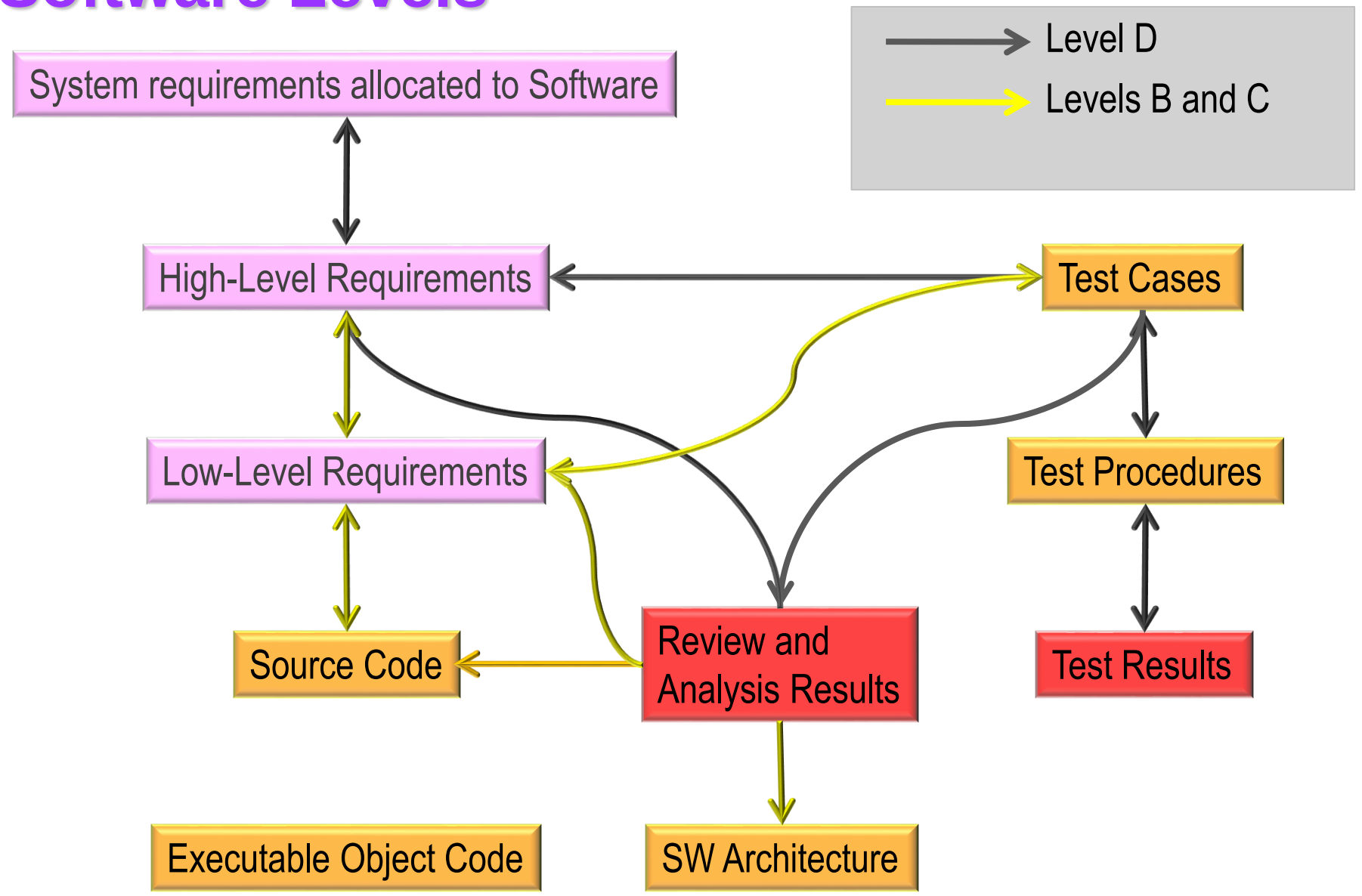
Traceability Management

- Production and maintenance of a traceability matrix that shows:
  - The bidirectional relationship between requirements and their immediate higher-level requirements
  - The bidirectional relationship between source code and its associated low-level requirements
  - The bidirectional relationship between requirements and the test cases and procedures that verify those requirements
  - The relationship of objectives with development assets and verification artifacts

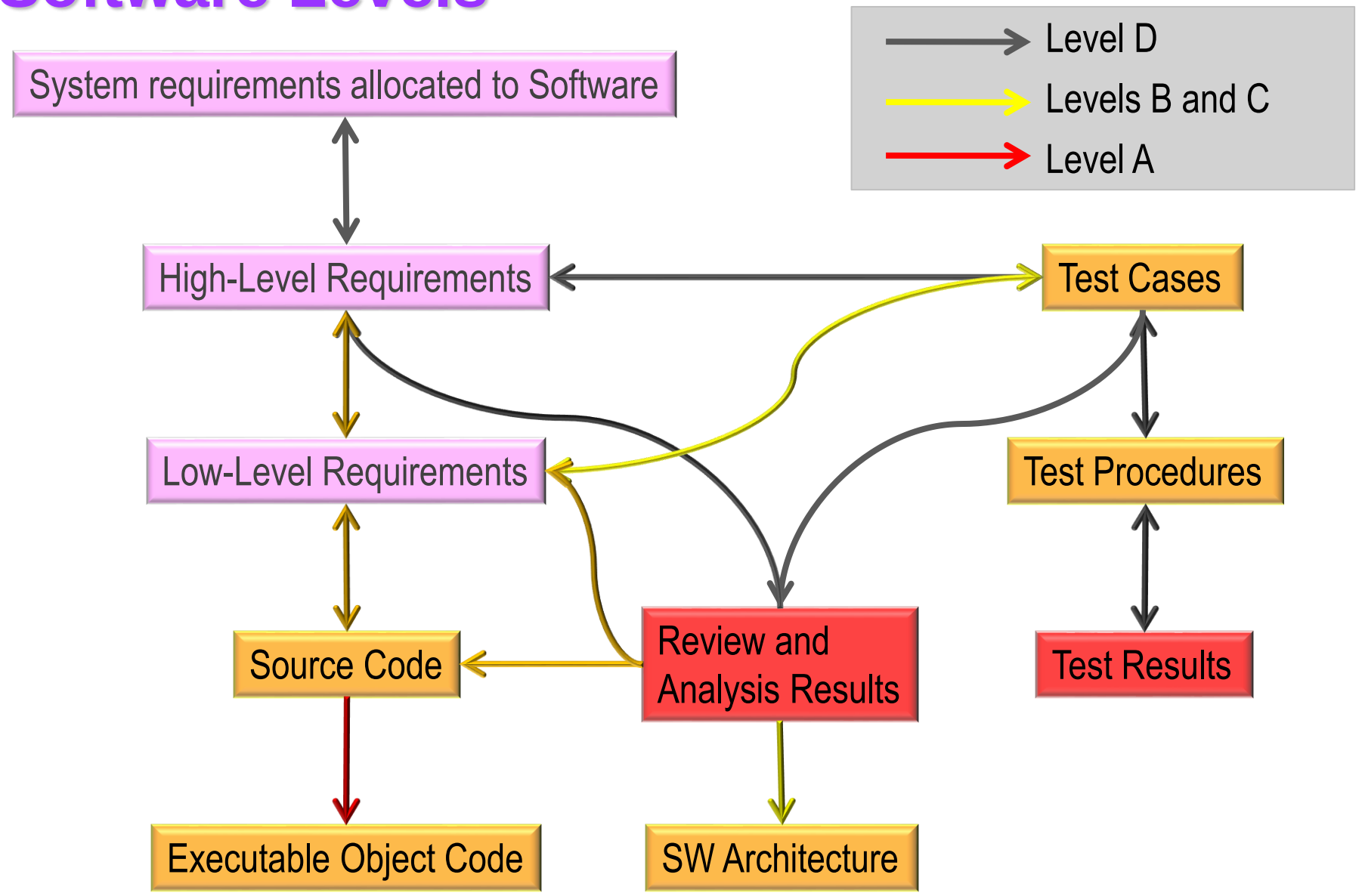
Requirements Traceability Across Software Levels



Requirements Traceability Across Software Levels

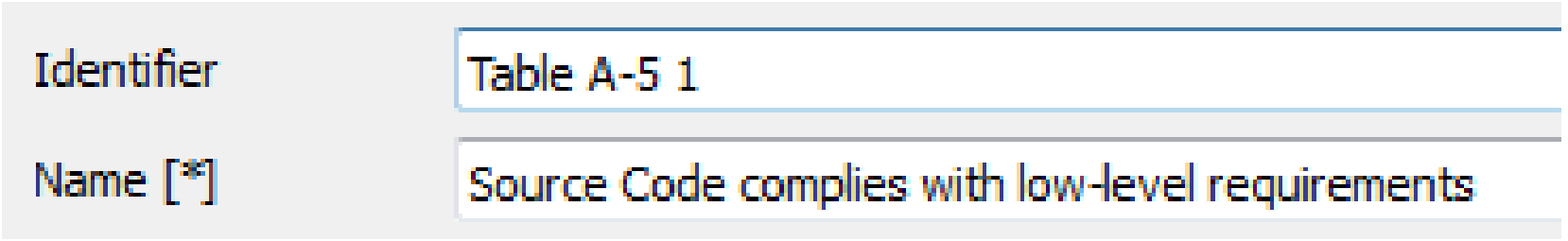


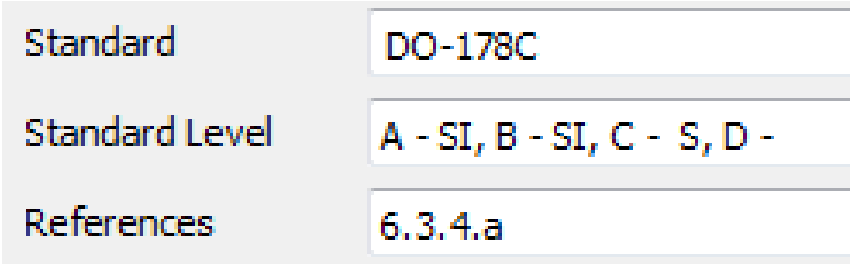
Requirements Traceability Across Software Levels



DO-178C Objectives

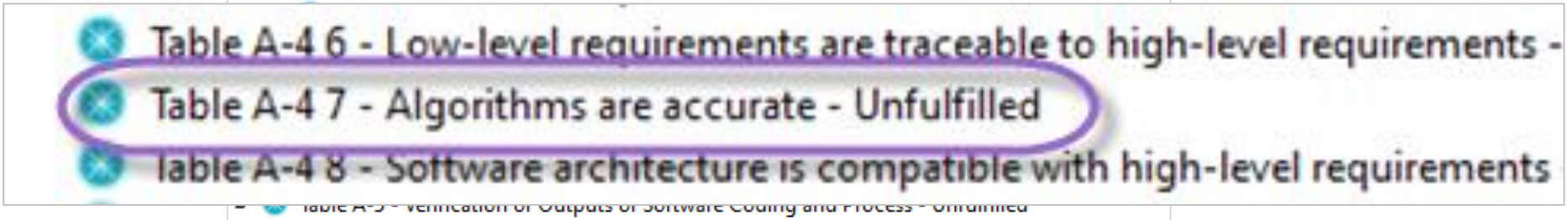
Design Assurance Level	Objectives	Objectives that must be verified with independence
A - Catastrophic	71	30
B - Hazardous	69	18
C - Major	62	5
D - Minor	26	2
E - No Effect	-	-





DO-178C Objectives

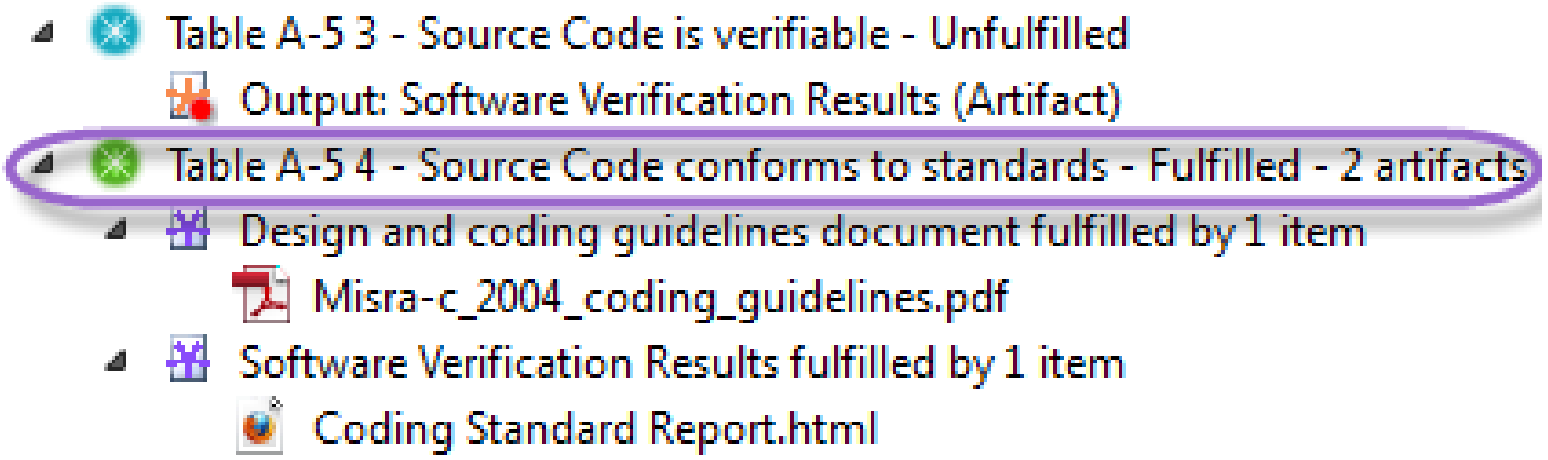
- Table A-1 - Software Planning Process - Unfulfilled
- Table A-2 - Software Development Process - Unfulfilled - 1 asset
- Table A-3 - Verification of Outputs of Software Requirements Process - Unfulfilled
- Table A-4 - Verification of Outputs of Software Design Process - Unfulfilled
  - Table A-4 1 - Low Level requirements comply with high level requirements - Unfulfilled
  - Table A-4 2 - Low-level requirements are accurate and consistent - Unfulfilled
  - Table A-4 3 - Low-level requirements are compatible with target computer - Unfulfilled
  - Table A-4 4 - Low-Level requirements are verifiable - Unfulfilled
  - Table A-4 5 - Low-level requirements conform to standards - Unfulfilled
  - Table A-4 6 - Low-level requirements are traceable to high-level requirements - Unfulfilled
  - Table A-4 7 - Algorithms are accurate - Unfulfilled



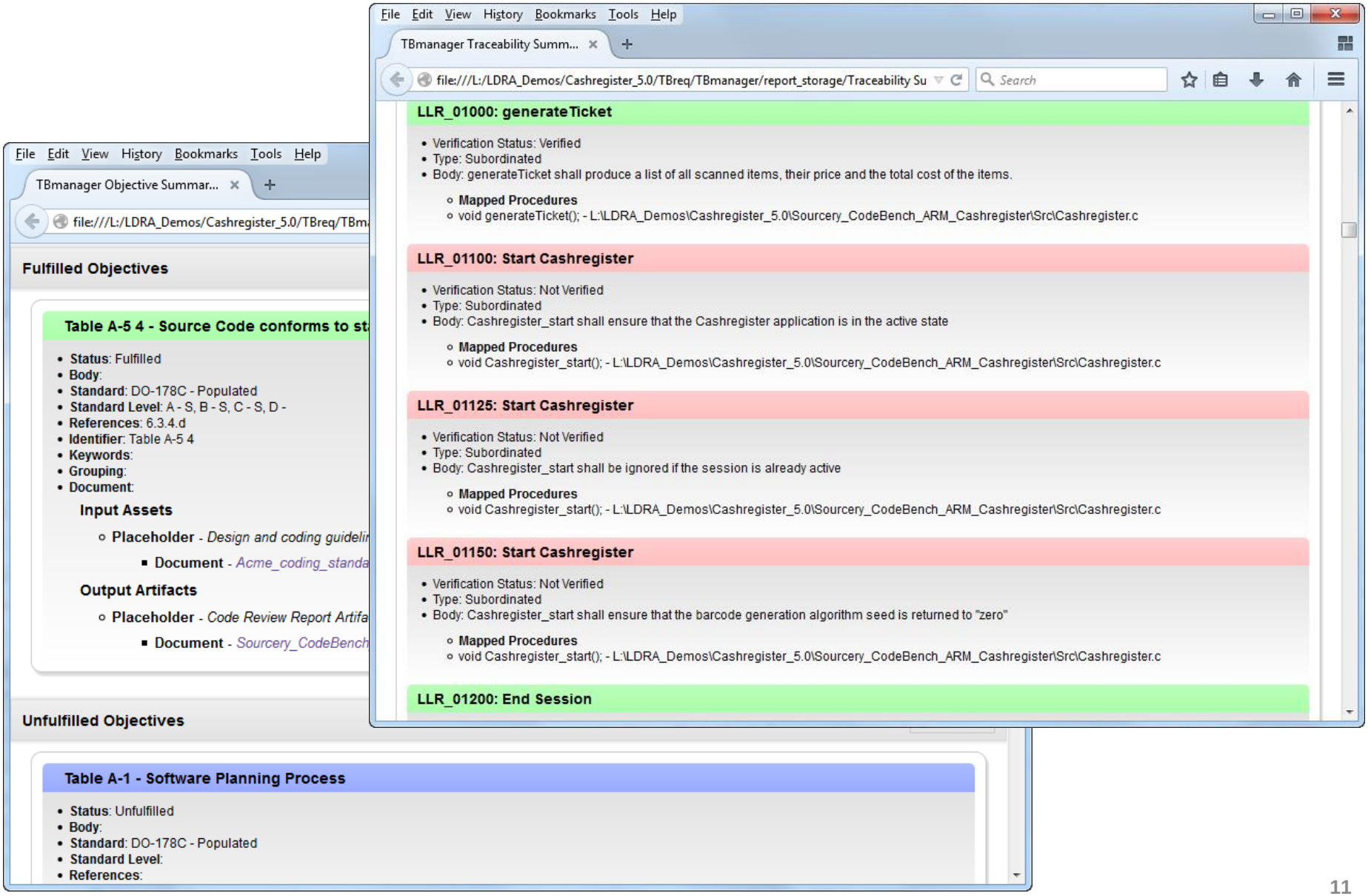
  - Table A-5 1 - Source Code complies with low-level requirements - Unfulfilled
  - Table A-5 2 - Source Code complies with software architecture - Unfulfilled
  - Table A-5 3 - Source Code is verifiable - Unfulfilled
  - Table A-5 4 - Source Code conforms to standards - Unfulfilled
  - Table A-5 5 - Source Code is traceable to low-level requirements - Unfulfilled
  - Table A-5 6 - Source Code is accurate and consistent - Unfulfilled
  - Table A-5 7 - Output of software integration process is complete and correct - Unfulfilled
  - Table A-5 8 - Parameter Data Item File is correct and complete - Unfulfilled
  - Table A-5 9 - Verification of Parameter Data Item File is achieved - Unfulfilled
- Table A-6 - Testing of Outputs of Integration Process - Unfulfilled
- Table A-7 - Verification of Verification Process Results - Unfulfilled
- Table A-8 - Software Configuration Management Process - Unfulfilled
- Table A-9 - Software Quality Assurance Process - Unfulfilled
- Table A-10 - Certification Liaison Process - Unfulfilled

Objectives Tracking

- Input assets and output artifacts can come from across the entire development lifecycle and must be mapped to objectives
- Evidence should be accumulated as it is produced, so inconsistencies or gaps may be easily identified
- Completed objectives tracking provides the audit evidence of standards adherence

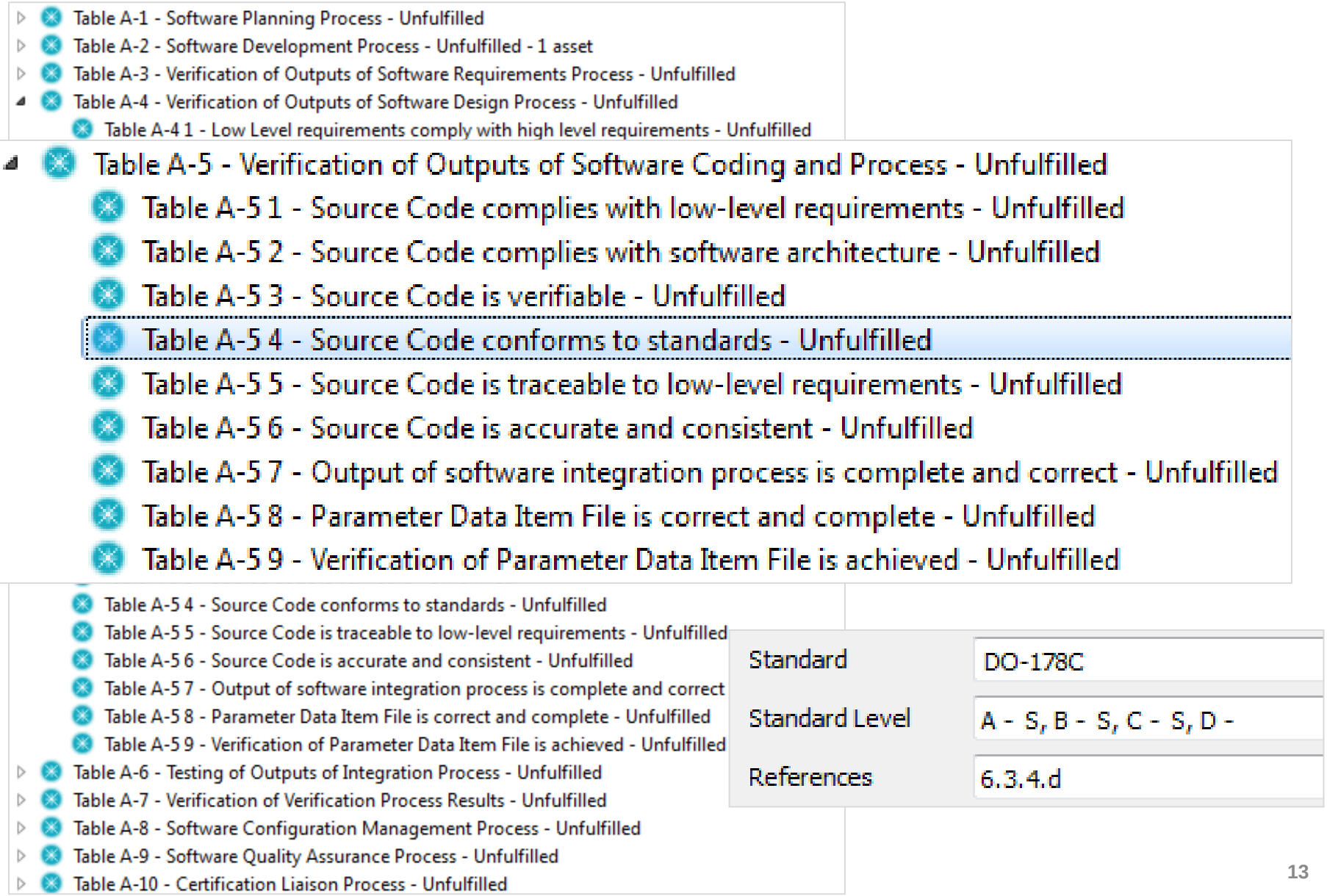


Traceability Reports



Aspect #2:
Coding Standards

DO-178C Objectives



Coding Standards

- Roughly 80% of software defects when using the C or C++ language, are attributable to the incorrect usage of 20% of the language constructs
- If the usage of the language can be restricted to avoid this subset that is known to be problematic, then the quality of the ensuing software is going to greatly increase

If you don't want defects in your code, then don't put them there!

Why use a Coding Standard?

- In many languages such as C, there are various constructs such as “goto” that when used incorrectly often lead to defects
- Other constructs such as operator precedence are not always fully understood by programmers
- It is easy to write code that is difficult to read
- Can help to prevent developers from trying to develop “*clever*” code
- The use of a Coding Standard can also help ensure that the code is written in a particular style; for example to ensure that the “tab” character is not used

Coding Standards

- DO-178C requires that developed code be "Uniformly designed and implemented"

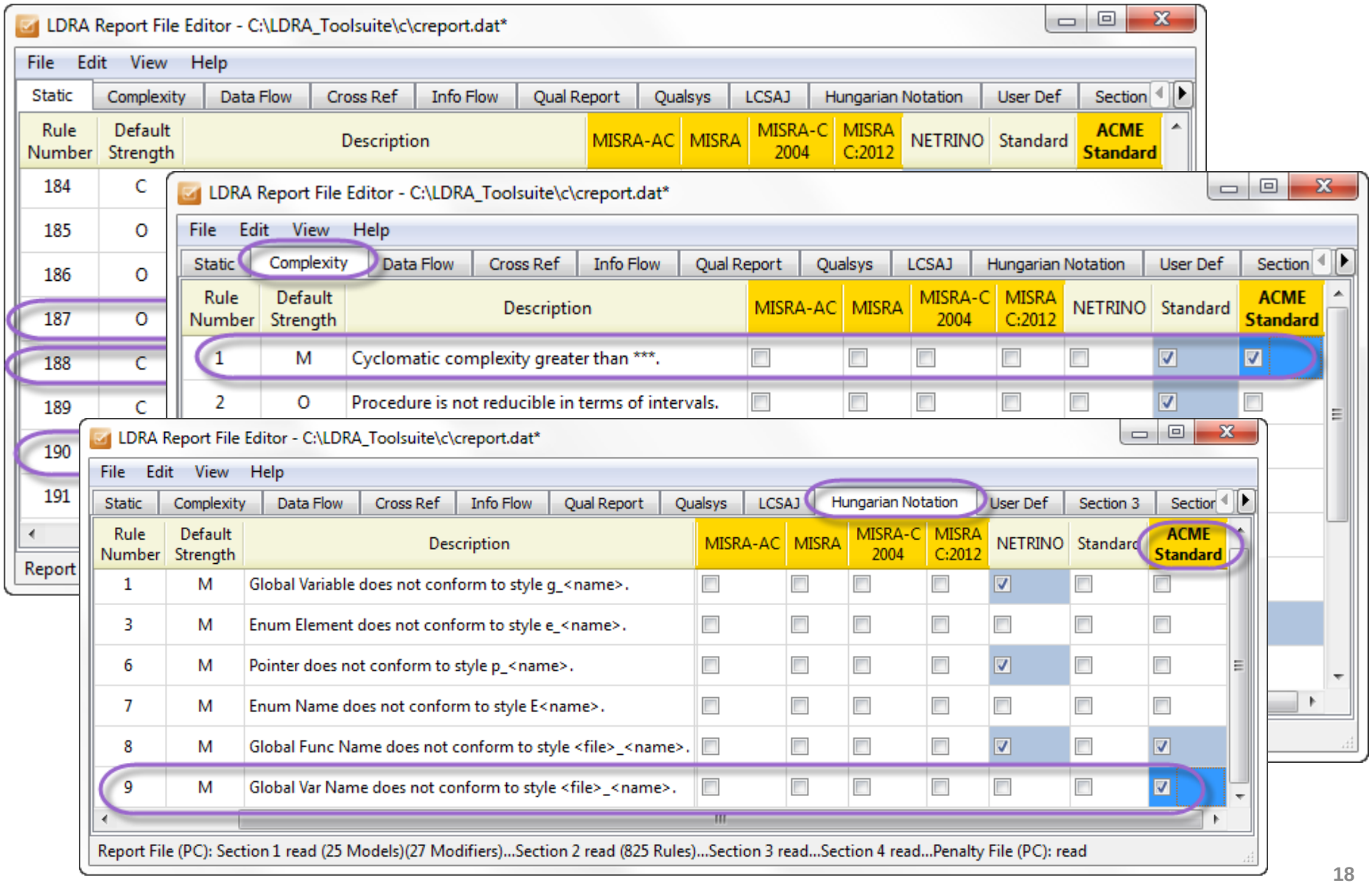
- DO-178C does not mandate any specific coding standard and as such any Coding Standard such as *MISRA C:2012*, *MISRA C++:2008*, *JSF++ AV*, … can be used

- Alternatively companies often create their own coding standard generally based on standards such as the ones stated above

Configure Standard

- Common sense rules
  - Don’t mix signed and unsigned types
- Reduced language subset
  - Ensure “goto” or “malloc” are not used
- Style guidelines
  - Ensure the “tab” character is not used
- Naming conventions
  - Ensure all public functions start with <filename>_
- Quality & complexity metrics
  - Ensure all functions have a low complexity

Configure Standard: Example

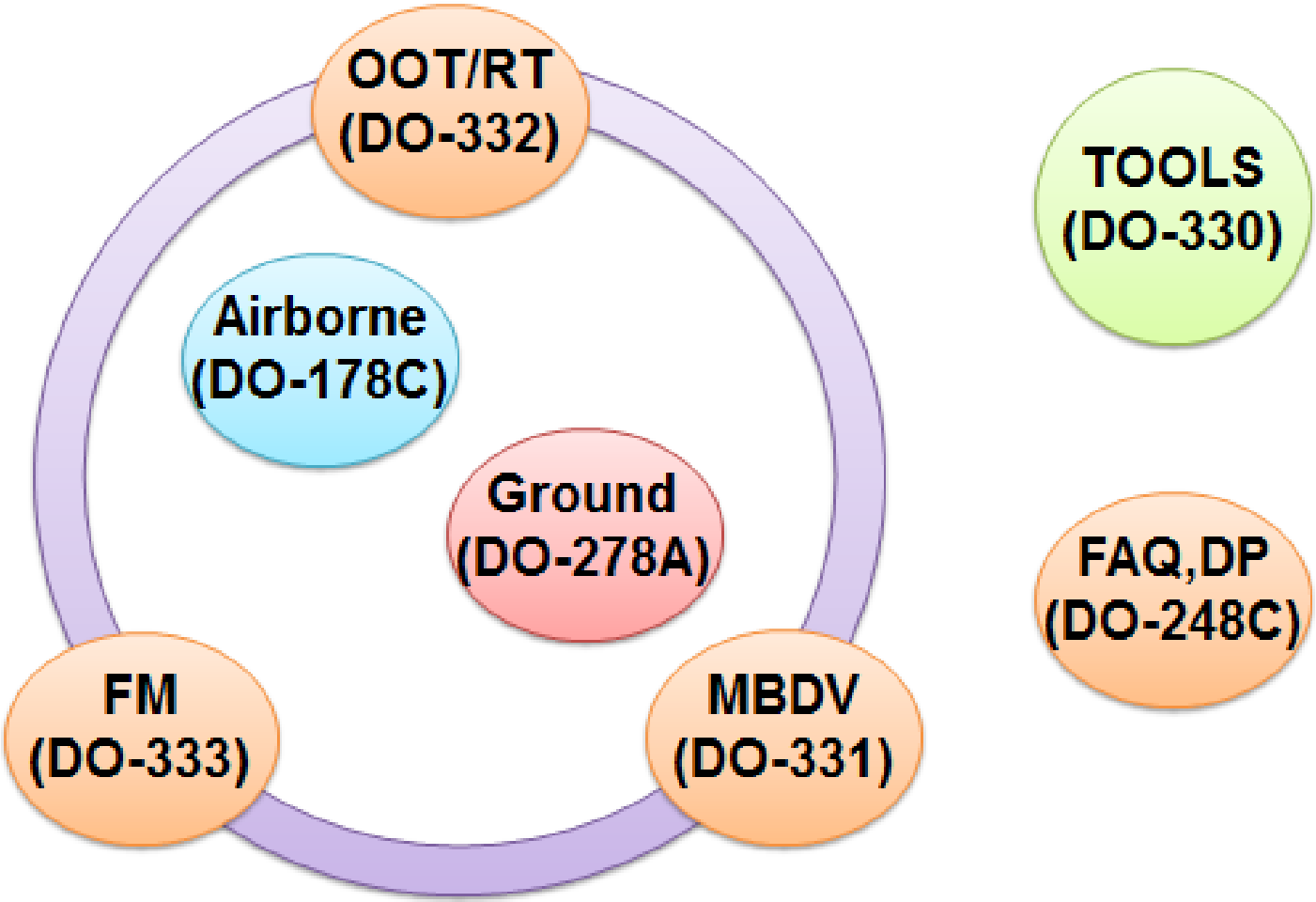


Checking Compliance

- Check the code manually
  - Needs to be done on the “undecidable” rules
  - But don’t really want to do it on all the code!
- Use a heavyweight tool
  - Slow (*Deep analysis*)
  - Detects the easy and hard to find defects
  - Tends to be “Pessimistic” – False Positives

Aspect #3:
Model Based Development and Verification

DO-178C & Supplementary Documents



DO-331: "Model-Based Development and Verification Supplement to DO-178C and DO-278"

DO-331

- MB.6.7: Model Coverage Analysis for Design Models
  - "Model coverage analysis is different than structural coverage analysis and therefore model coverage analysis does not eliminate the need to achieve the objectives of structural analysis per DO-178C section 6.4.4.2"
- MB.6.8.2: Model Simulation for Verification of the Executable Object Code
  - "The software testing objective, Executable Object Code is compatible with the target computer, cannot be satisfied by means of simulation (see DO-178C 6.4e). In particular tests dedicated to hardware/software integration verification always need to be performed in the target computer environment as defined in DO-178C section 6.4.3.a"

Accruing Coverage Credit

LDRA

Simulation

Test Cases

Model Coverage

+ 100 %

C Level Coverage

< 100 %

Host Computer

Test Cases

Additional Test Cases

C Level Coverage

+ 100 %

Target H/W

Test Cases

Additional Test Cases

C Level Coverage

+ 100 %

ASM Level (Level A)

+ 100 %

Independent Back to Back Testing

LDRA - LDRA Testbed Dynamic Overview Report - Code Composer Studio

File Edit Navigate Search Project LDRA Tools Run Window Help

Quick Access | CCS Edit | CCS Debug | LDRA | Resource

C/C++ P... | Project E... | Navigator

- TMS570_Launchpad_BSP
- TMS570_Launchpad_Dice
- TMS570_Launchpad_QuickBrownFox
- **TMS570_Launchpad_Simulink_CruiseControl [Active**
  - Binaries
  - Includes
  - Debug
  - LDRA
  - targetConfigs
  - main.c
  - model_reference_types.h
  - rtwtypes.h
  - sldvdemo_cruise_control2_harness_private.h
  - sldvdemo_cruise_control2_harness_types.h
  - sldvdemo_cruise_control2_harness.c
  - sldvdemo_cruise_control2_harness.h
  - sldvdemo_cruise_control2_private.h
  - sldvdemo_cruise_control2_types.h
  - sldvdemo_cruise_control2.c
  - sldvdemo_cruise_control2.h
  - sys_link.cmd
  - sysppvar.dat
  - target.ccxml
  - TMS570_Launchpad_BSP.lib

LDRA Testbed Dynamic Overview Report

file:///C:/LDRA_Workarea/TMS570_Launchpad_Simulink_CruiseControl_tbwrkfls/TMS570_Launchpad_Simulink_CruiseControl.ms3.ht

Combined Coverage Results

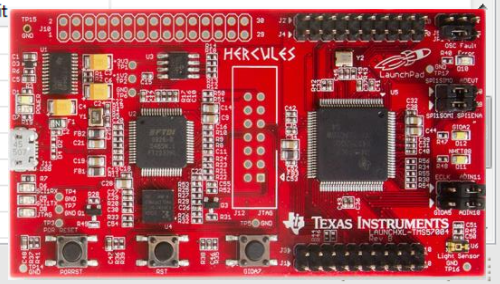
Procedure	Statement	Branch	MC/DC
TMS570_Launchpad_Simulink_CruiseControl (Set)	78	67	75
sldvdemo_cruise_control2_harness_step	+62	+47	[No BCs]
sldvdemo_cruise_control2_harness_initialize	+100	[No Branches]	[No BCs]
sldvdemo_cruise_control2_harness_terminate	+100	[No Branches]	[No BCs]
sldvdemo_cruise_control2	+91	+85	+75
sldvdemo_cruise_cont_initialize	+100	[No Branches]	[No BCs]
main	+100	+100	[No BCs]

[Top of Report | Contents]

LDRA Tool Suite Output | LDRA Code Violations | LDRA Code Coverage | Problems | Console | Progress | Expand

Double click a Function to Open in Editor.

	Percentage	Success Limit
L:\LDRA_Demos\TICCS60_Workspace\TMS570_Launchpad_\		
Combined Coverage Run		
Statement Coverage	91	100
Branch/Decision Coverage	85	100
Modified Condition / Decision Coverage	75	100
sldvdemo_cruise_control2		
sldvdemo_cruise_cont_initialize		
L:\LDRA_Demos\TICCS60_Workspace\TMS570_Launchpad_\		

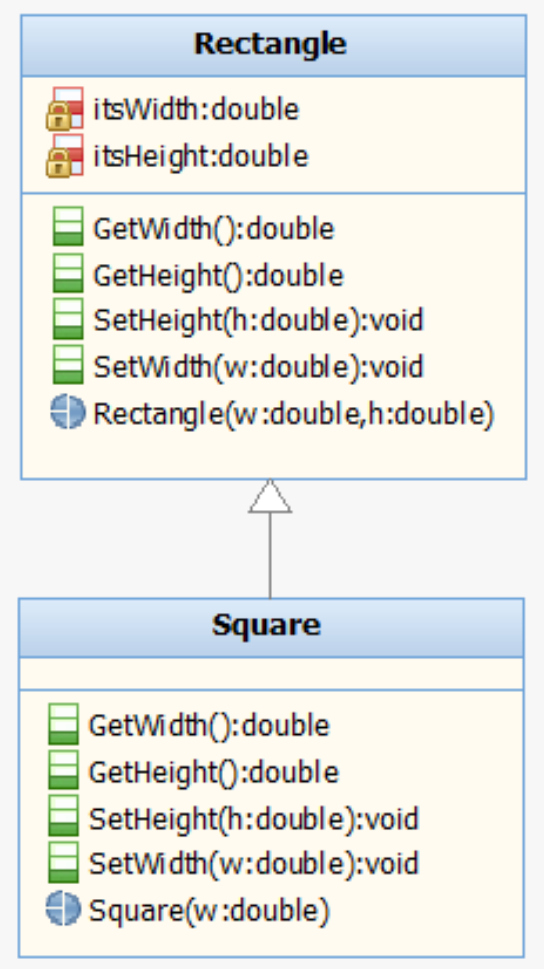


Done

Aspect #4:
Subclass Verification for Local Type Consistency

Design Verification Testing Scenarios

- Assume a subclass *Square* is being added to an existing parent C++ class *Rectangle*
- Override parent class virtual methods with new implementations in the subclasses
- In compliance with the DO-332, ensure that the low level requirements of the parent class are met by the subclasses, thereby verifying local type consistency



DO-332: "Object-Oriented Technology and Related Techniques Supplement to DO-178C and DO-278A"

Parent & Subclass Implementation

LDRA

- virtual void Rectangle::SetWidth (double w) {
  itsWidth=w;
  }

  Post condition: itsHeight does not change

- void Square::SetWidth (double w) {
  Rectangle::SetWidth(w);
  Rectangle::SetHeight(w);
  }

  Post condition: itsHeight set to w

Traceability

- Low level requirements map to the parent class
- Subclasses must be proven to be type consistent by meeting the same verification criteria for redefined functions

Requirement Name: Set Width

Requirement Number: LL_3_u001

Body:

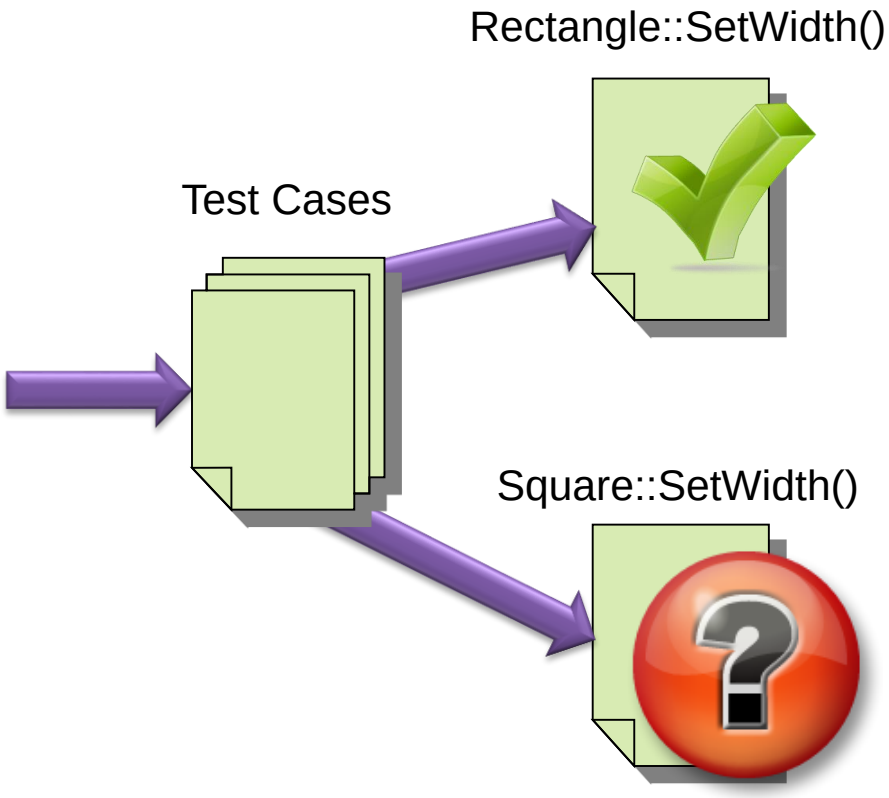
The software shall correctly set the width of the shape.

Post Condition:

```
assert((itsWidth == w) && (itsHeight == old.itsHeight));
```

Invariant:

```
assert(itsWidth > 0);
```



Parent Class (Rectangle) Results

- Use negative testing to verify that *SetWidth()* does not modify the member variable *itsHeight*

Test Case	Regression P / F	Procedure
1	PASS	Rectangle::Rectangle
2 Negative Testing		Rectangle::SetWidth

Variable	Type	Initial Value	Final Value	Expected	Status
itsHeight	Double	1.000000e+000	1.000000e+00	No Change	PASS
itsWidth	Double	2.000000e+000	4.000000e+00	Change	PASS

Subclass (Square) Results

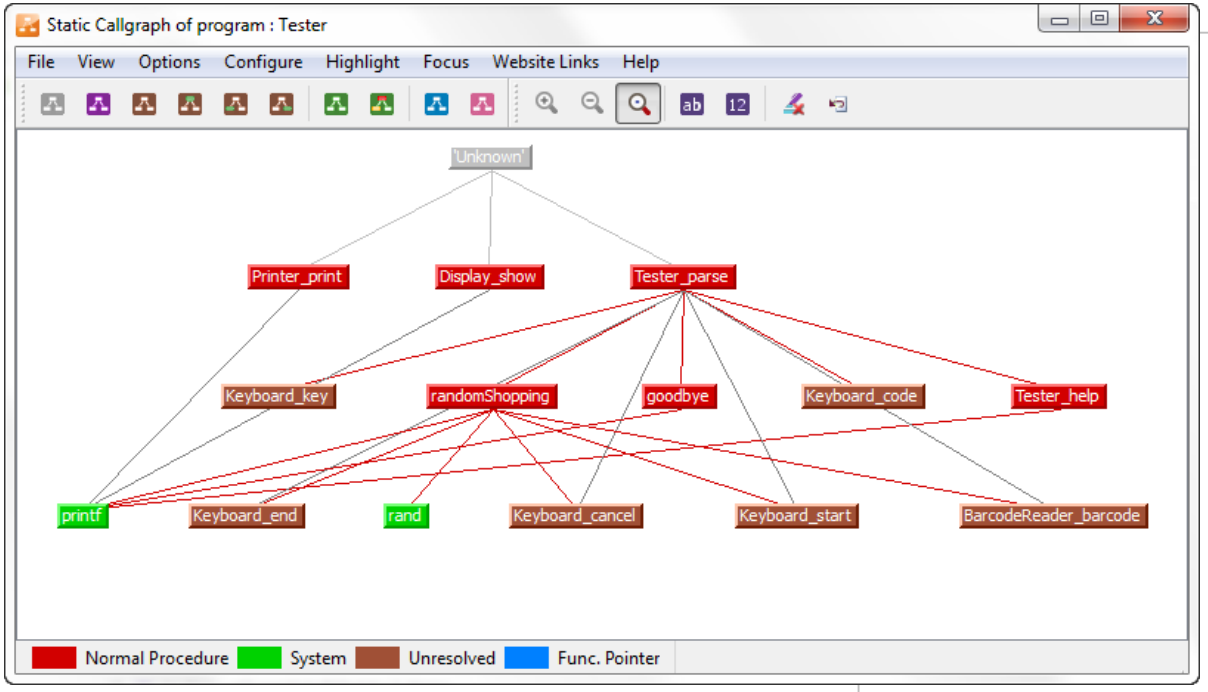
- The same tests, run against the subclass, demonstrate type inconsistency

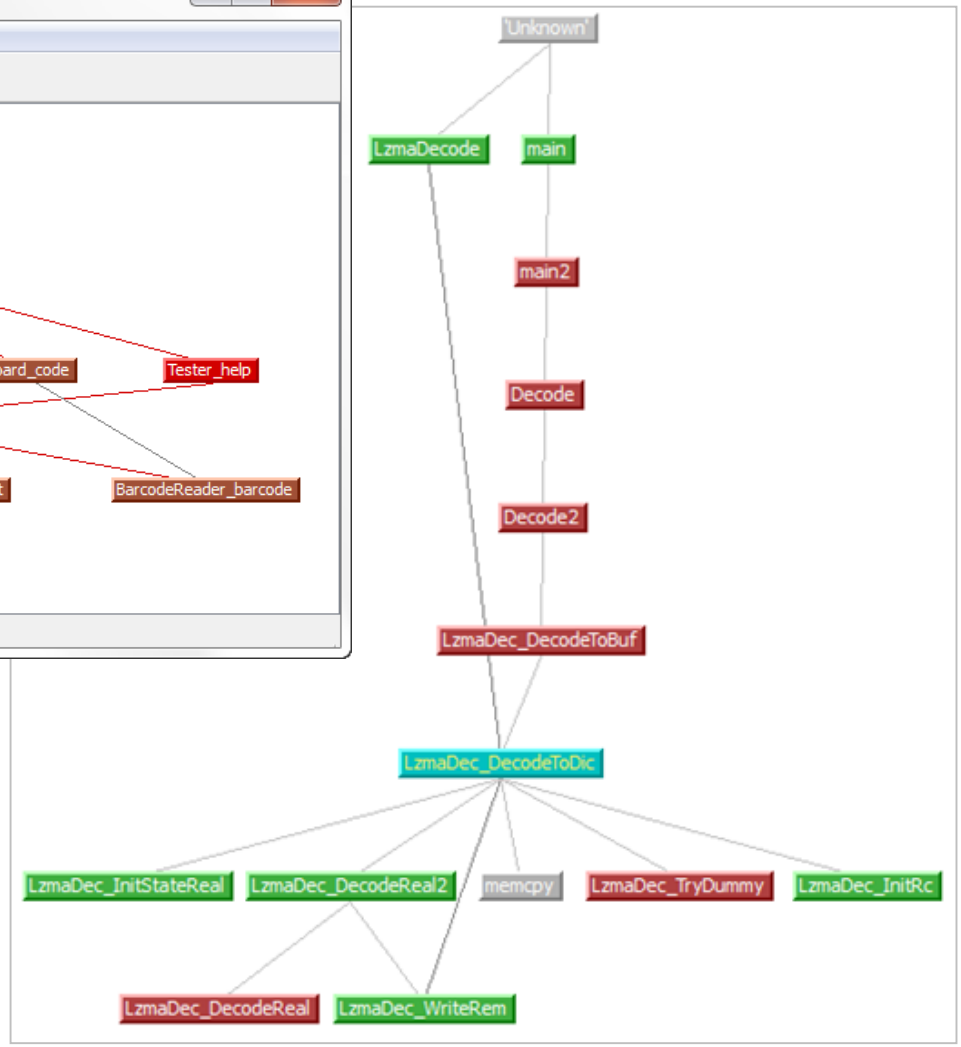
Test Case	Regression P / F	Procedure
1		C:\Square.tcf
2		C:\Square SetWidth.tcf

Variable	Type	Initial Value	Final Value	Expected	Status
itsHeight	Double	1.000000e+000	**4.000000e+00**	No Change	**FAIL**
itsWidth	Double	2.000000e+000	4.000000e+00	Change	PASS

Aspect #5:
Data and Control Coupling Coverage

Control Flow Analysis





Data Flow Analysis

- How is data used?

```
169  #pragma vector=ADC12_VECTOR
170  __interrupt void ADC12ISR (void)
171  {
172    switch(__even_in_range(ADC12IV,34))
173    {
174    case  0: break;
175    case  2: break;
176    case  4: break;
177    case  6:
178          adc12_result = ADC12MEM0;
179          adc12_data_ready = 1;
180          _BIC_SR_IRQ(LPM3_bits);
181          break;
182    case  8: break;
```

adc12_result (Pri)		u16	adc12.c	<Global scope>	G	E	60				
			adc12.h/adc12.c		G	E	62				
				ADC12ISR	G	D	178				
				adc12_single_conversion	G	R	157				

Data and Control Coupling Coverage

- DO-178C section 6.4.4.2 c states:

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

- Control coupling **coverage** is ensuring that every invocation of a function has been exercised
- Data coupling **coverage** is ensuring that we have exercised every access to the data

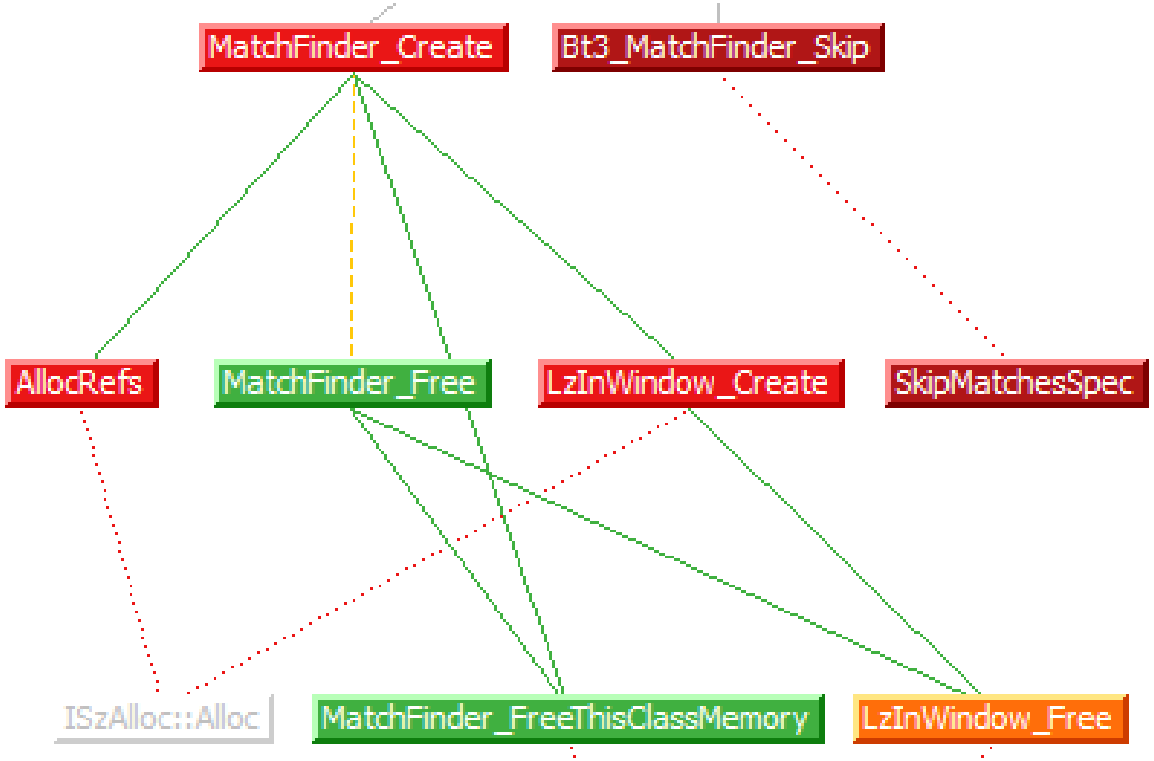
Data Coupling and Control Coupling

Technical Note

Author: Professor M. A. Hennell

Control Coupling Coverage

- Table A-7 5 - Test coverage of software structure (modified condition/decision coverage) is achieved - Unfulfilled
- Table A-7 6 - Test coverage of software structure (decision coverage) is achieved - Unfulfilled
- Table A-7 7 - Test coverage of software structure (statement coverage) is achieved - Unfulfilled
- Table A-7 8 - Test coverage of software structure (data coupling and control coupling) is achieved - Fulfilled - 2 assets
  - Software Verification Results fulfilled by 2 items
    - Callgraph - Pass/Fail Coverage
    - DataCouplingReport.html
- Table A-7 9 - Verification of additional code that can not be traced to Source Code, is achieved - Unfulfilled



Data Coupling Coverage

- Table A-7 5 - Test coverage of software structure (modified condition/decision coverage) is achieved - Unfulfilled
- Table A-7 6 - Test coverage of software structure (decision coverage) is achieved - Unfulfilled
- Table A-7 7 - Test coverage of software structure (statement coverage) is achieved - Unfulfilled
- Table A-7 8 - Test coverage of software structure (data coupling and control coupling) is achieved - Fulfilled - 2 assets
  - Software Verification Results fulfilled by 2 items
    - Callgraph - Pass/Fail Coverage
    - DataCouplingReport.html
- Table A-7 9 - Verification of additional code that can not be traced to Source Code, is achieved - Unfulfilled

	Call Depth / Parameter Name					Used on lines...					
Variable Name	**Alias**	**File**	**Procedure**	**Type Code**	**Attribute Code**						
OutBound.itsIException		Calculator.cpp	Calculator::calc_C::setItsIException	P	D						
	1 itsIException	Calculator.cpp	Calculator::calc_C::OutBound_C::setItsIException	P	D	118					
_p_		Calculator.cpp	Calculator::calc_C::calc_C	P	D	128					
		Tester.cpp	Tester::calc_C::calc_C	P	D	130					
arg		TesterBuilder.cpp	TesterBuilder::cancelTimeout	P	E	58					
				P	R	60 *****					
arg1		Calculator.cpp	Calculator::add	P	E	177					
				P	R	180					
arg1			Calculator::calc_C::InBound_C::add	P	E	26					
				P	R	29					
	1 arg1		Calculator::add	P	E	177					
				P	R	180					
arg1			Calculator::calc_C::InBound_C::divide	P	E	34					
				P	R	37					

data access not exercised!

Aspect #6:
Object Code Verification

DO-178B - The "Hidden Objective"

- In 6.4.4.2, b, there is a Level A Objective that is "*hidden*" in the Structural Coverage Analysis guidance

6.4.4.2 Structural Coverage Analysis

The objective of this analysis is to determine which code structure was not exercised by the requirements-based test procedures. The requirements-based test cases may not have completely exercised the code structure, so structural coverage analysis is performed and additional verification produced to provide structural coverage. Guidance includes:

a. The analysis should confirm the degree of structural coverage appropriate to the software level.

b. The structural coverage analysis may be performed on the Source Code, unless the software level is A and the compiler generates object code that is not directly traceable to Source Code statements. Then, additional verification should be performed on the object code to establish the correctness of such generated code sequences. A compiler-generated array-bound check in the object code is an example of object code that is not directly traceable to the Source Code.

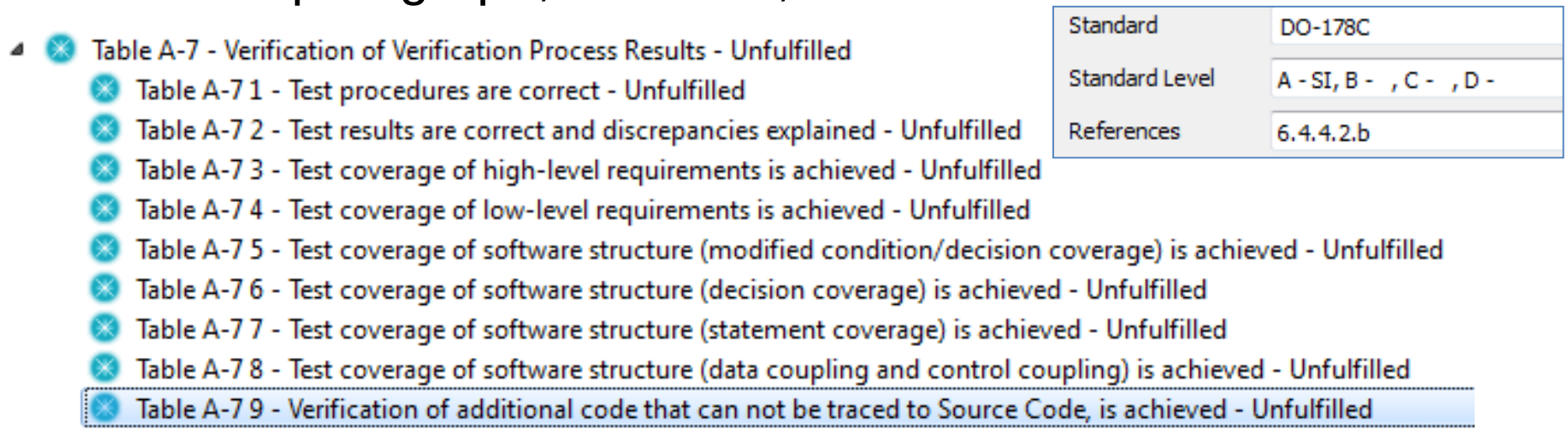
c. The analysis should confirm the data coupling and control coupling between the code components.

DO-178B 6.4.4.2.b

- For level A, it is necessary to show 100% coverage of not just the high-level language, but also verify any additional, non-traceable Executable Object Code (EOC)
- Generally we can't read Object Code, but we can read the generated assembler code and since there must be a one to one relationship between the Object Code and Assembler, we could either:
  - Manually inspect and verify the generated assembler code
- Or:
  - Automatically run the existing test cases and obtain the assembler level code coverage

In DO-178C, EOC Verification Is Official!

- In DO-178C, Executable Object Code verification is now added to Table A-7, the Objectives table for testing
  - "Verification of additional code, that cannot be traced to Source Code, is achieved."
  - The Activity associated with this new Objective is in the same paragraph, 6.4.4.2 b, as in DO-178B



Object Code Verification

- Object code verification hinges on how much the control flow structure of the compiler-generated object code differs from that of the application source code from which it was derived (therefore raising questions about traceability)
- Even if no options are set, then the control flow of the generated code will differ from the source code
- If options are set for the compiler to optimize the code or to add array bounds checking, then the control flow of the generated code will possibly greatly differ from that of the source code

Sample “C” Code

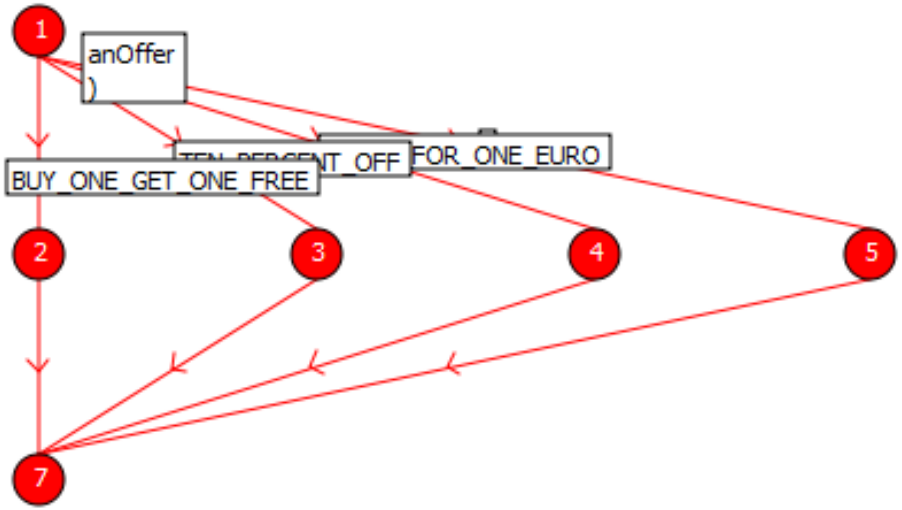
- Consider the following simple C code:

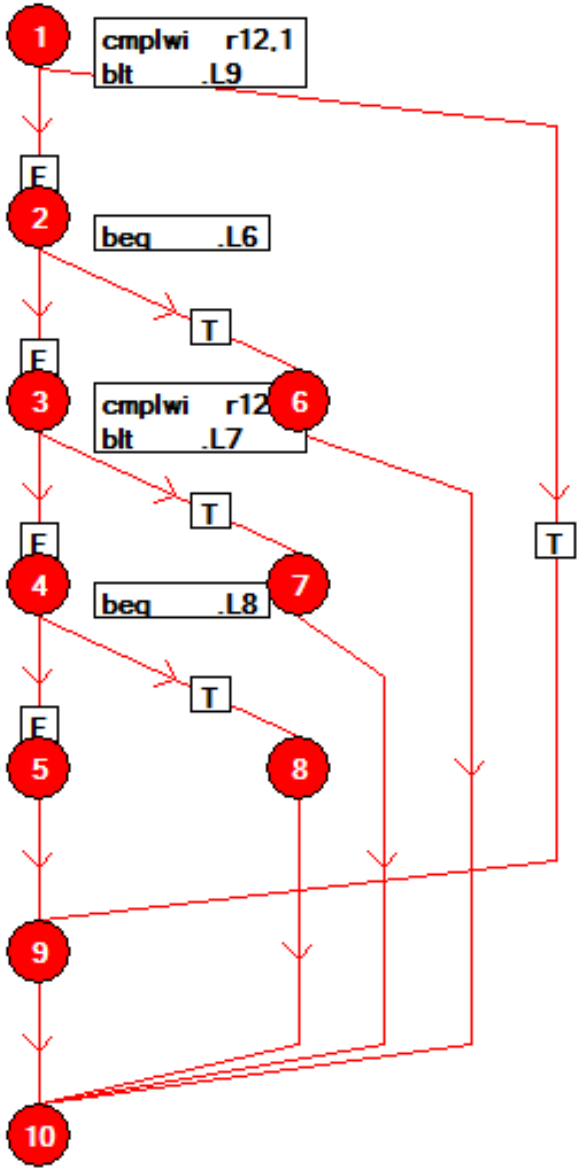
```c
#include "misrac_types.h"
#include "specialoffer.h"

/*
 * Get the price which depends on which special offer, if any, is used
 */
LDRA_uint32_t SpecialOffer_getPrice(const LDRA_uint32_t aQuantity,
    const LDRA_uint32_t aUnitPrice, const tSpecialOffer anOffer)
{
  LDRA_uint32_t price;
  switch (anOffer)
  {
    case BUY_ONE_GET_ONE_FREE:
      price = aUnitPrice * ((aQuantity + 1U) >> 1U);
      break;
    case TEN_PERCENT_OFF:
      price = (aUnitPrice * aQuantity * 9U) / 10U;
      break;
    case THREE_FOR_ONE_EURO:
      price = ((aQuantity / 3U) * 100U) + ((aQuantity % 3U) * aUnitPrice);
      break;
      /* no offer */
    default:
      price = aUnitPrice * aQuantity;
      break;
  }
  return price;
}
```

Assembler Code Structure

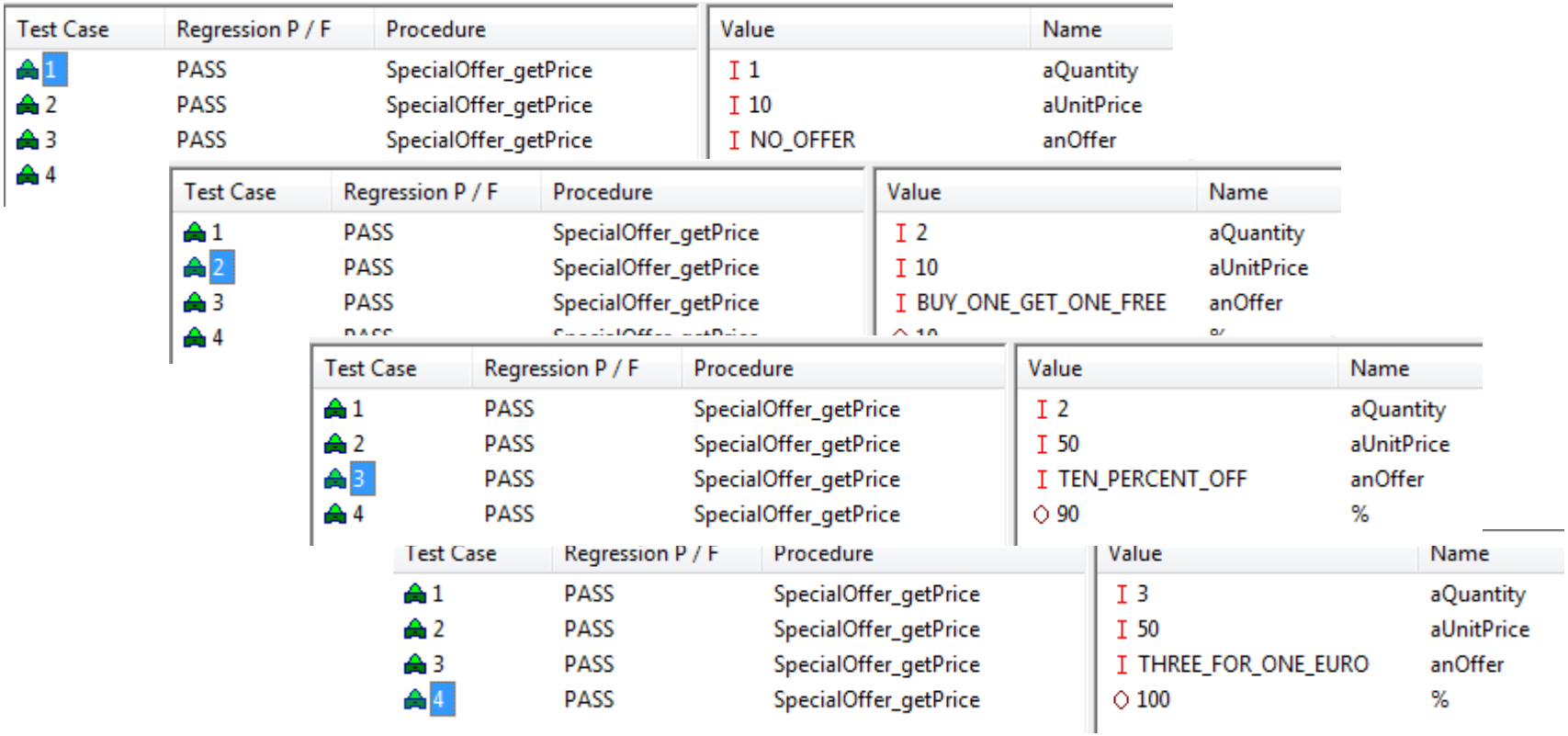
- As we can see, the structure of the generated assembler code is quite different to that of the C code





C Code Structural Coverage

- In order to get 100% statement and 100% decision coverage of the “C” code, we need to create four test cases:



Test Case	Regression P / F	Procedure
1	PASS	SpecialOffer_getPrice
2	PASS	SpecialOffer_getPrice
3	PASS	SpecialOffer_getPrice
4		

Value	Name
1	aQuantity
10	aUnitPrice
NO_OFFER	anOffer

Test Case	Regression P / F	Procedure
1	PASS	SpecialOffer_getPrice
2	PASS	SpecialOffer_getPrice
3	PASS	SpecialOffer_getPrice
4		

Value	Name
2	aQuantity
10	aUnitPrice
BUY_ONE_GET_ONE_FREE	anOffer

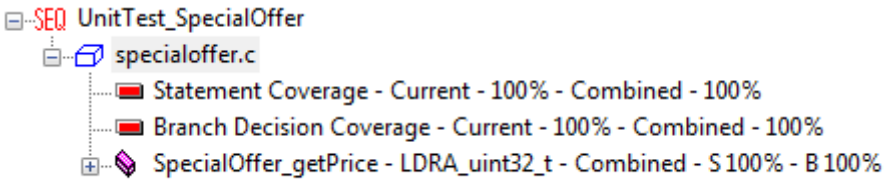
Test Case	Regression P / F	Procedure
1	PASS	SpecialOffer_getPrice
2	PASS	SpecialOffer_getPrice
3	PASS	SpecialOffer_getPrice
4	PASS	SpecialOffer_getPrice

Value	Name
2	aQuantity
50	aUnitPrice
TEN_PERCENT_OFF	anOffer
90	%

Test Case	Regression P / F	Procedure
1	PASS	SpecialOffer_getPrice
2	PASS	SpecialOffer_getPrice
3	PASS	SpecialOffer_getPrice
4	PASS	SpecialOffer_getPrice

Value	Name
3	aQuantity
50	aUnitPrice
THREE_FOR_ONE_EURO	anOffer
100	%

C Code Coverage



LINE NUMBER REF. (SOURCE)	STATEMENT	PREVIOUS RUNS	CURRENT RUN	COMBINED
150 (16)	LDRA_uint32_t	-	-	-
151	SpecialOffer_getPrice (	0	4	4
152	const LDRA_uint32_t aQuantity ,	-	-	-
153 (17)	const LDRA_uint32_t aUnitPrice ,	-	-	-
154	const tSpecialOffer anOffer)	-	-	-
155 (18)	{	-	-	-
156 (19)	LDRA_uint32_t	-	-	-
157	price ;	-	-	-
158 (20)	switch (	0	4	4
159	anOffer	0	4	4
160	)	0	4	4
161 (21)	{	0	4	4
162 (22)	case BUY_ONE_GET_ONE_FREE :	0	1	1
163 (23)	price = aUnitPrice * (	0	1	1
164	(aQuantity + 1U) >> 1U) ;	0	1	1
165 (24)	break ;	0	1	1
166 (25)	case TEN_PERCENT_OFF :	0	1	1
167 (26)	/* LDRA_INSPECTED 96 S	-	-	-
168	*/	-	-	-
169 (27)	price = (aUnitPrice * aQuantity * 9U) /	0	1	1
170	10U ;	0	1	1
171 (28)	break ;	0	1	1
172 (29)	case THREE_FOR_ONE_EURO :	0	1	1
173 (30)	/* LDRA_INSPECTED 96 S	-	-	-
174	*/	-	-	-
175 (31)	price = (	0	1	1
176	(aQuantity /	0	1	1
177	3U) * 100U) + (	0	1	1
178	(aQuantity % 3U) * aUnitPrice) ;			
179 (32)	break ;			
180 (33)	/* no offer */			
181 (34)	default :			
182 (35)	price = aUnitPrice * aQuantity ;			
183 (36)	break ;			
184 (37)	}			
185 (38)	return			
186	price ;			
187 (39)	}			

LDRA Testbed ® Dynamic Coverage Analysis Report

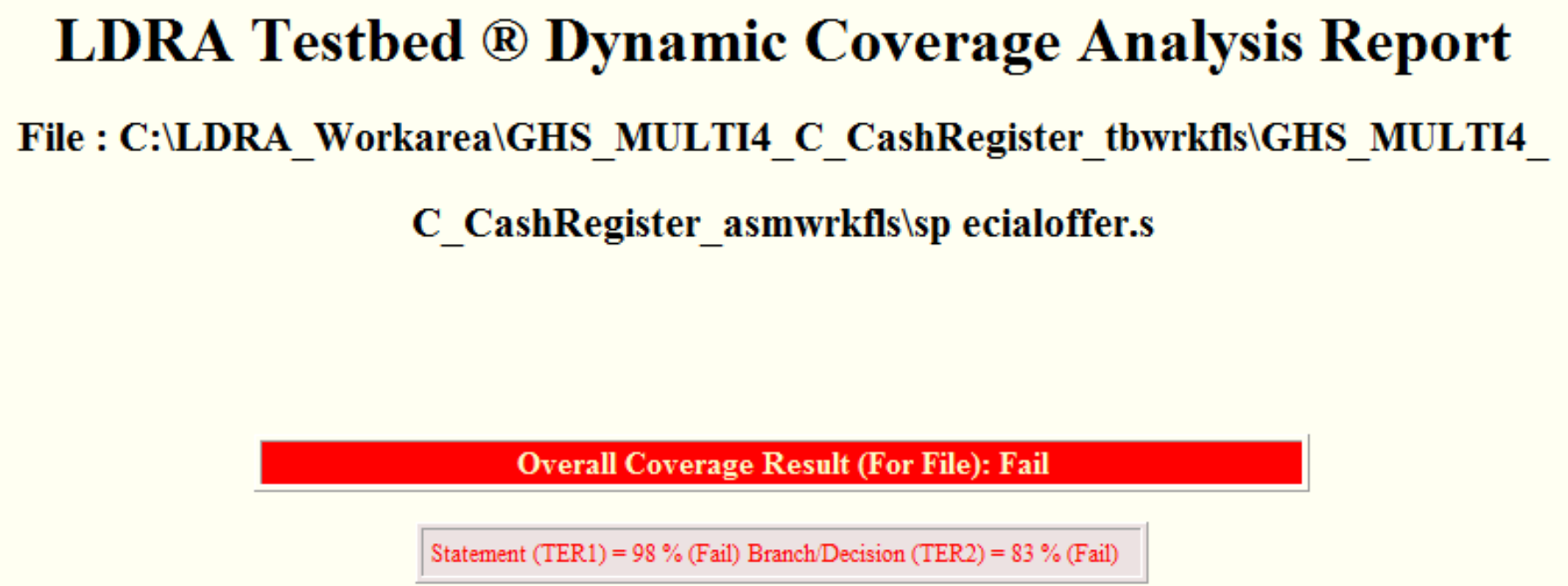
File : C:\LDRA_Demos\GHS_MULTI4_Workspace\GHS_MULTI4_C_CashRegister\src\specialoffer.c

Overall Result (For File): Coverage Metrics required to achieve DO-178B Level A Attained

Statement (TER1) = 100 % Branch/Decision (TER2) = 100 % MC/DC : Not Applicable

Assembler Structural Coverage

- At the same time, when we inspect the code coverage on the assembler code, we observe that we have not achieved 100% coverage

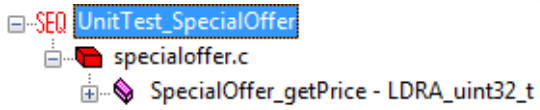


Assembler Code Coverage

- There is one **decision** that is not covered

```
=============================================================================================================================
LINE NUMBER                                                                                  PREVIOUS  CURRENT
REF. (SOURCE)     STATEMENT                                                                  RUNS      RUN       COMBINED
=============================================================================================================================
34 (31)         SpecialOffer_getPrice:                                                     4         4         8
35 (32)                       mr        r8,r3                                              4         4         8
36 (33)                       mr        r9,r4                                              4         4         8
37 (34)                       mr        r12,r5                                             4         4         8
38 (36)         #     .bf                                                                  -         -         -
39                                                                                         -         -         -
40              .LDW01:                                                                    4         4         8
41 (38)         #17:      const LDRA_uint32_t aUnitPrice, const tSpecialOffer anOffer)     -         -         -
42 (39)         #18: {                                                                     -         -         -
43 (40)         #19:    LDRA_uint32_t price;                                               -         -         -
44 (41)         #20:    switch (anOffer)                                                   -         -         -
45                            cmplwi    r12,1                                            4         4         8
46 (42)                       blt       .L9                                              4         4         8
47 (43)                       beq       .L6                                              3         3         6
48 (44)                       cmplwi    r12,3                                            2         2         4
49 (45)                       blt       .L7                                              2         2         4
50 (46)                       beq       .L8                                              1         1         2
51 (47)                       b         .L9                                              0  ***    0  ***    0  ***
52 (48)                                                                                  -         -         -
53              .L6:                                                                       1         1         2
54 (50)         #21:    {                                                                  -         -         -
55 (51)         #22:      case BUY_ONE_GET_ONE_FREE:                                       -         -         -
56 (52)         #23:        price = aUnitPrice * ((aQuantity + 1U) >> 1U);                 -         -         -
57 (53)         #line23                                                                    -         -         -
58 (54)         #.lin.C.3A.5CLDRA_Workarea.5CGHS_MULTI4_C_CashRegister_tbwrkfls.5CGHS      -         -         -
59                                                                                         -         -         -
60              .LDWlin1:                                                                  1         1         2
61 (55)                       addi      r12,r8,1                                         1         1         2
62 (56)                       srwi      r12,r12,1                                        1         1         2
63 (57)                       mullw     r12,r9,r12                                       1         1         2
64 (58)                       b         .L4                                              1         1         2
65 (59)                                                                                  -         -         -
66              .L7:                                                                       1         1         2
67 (61)         #24:        break;                                                         -         -         -
68 (62)         #25:      case TEN_PERCENT_OFF:                                            -         -         -
69 (63)         #26:         /*LDRA_INSPECTED 96 S */                                     -         -         -
70 (64)         #27:        price = (aUnitPrice * aQuantity * 9U) / 10U;                   -         -         -
71                            mullw     r12,r9,r8                                        1         1         2
72 (65)                       mr        r11,r12                                          1         1         2
```

100% Assembler Coverage



- With the addition of one extra test, we can get 100% coverage of the assembler code:

Test Case	Regression P / F	Procedure
1	PASS	SpecialOffer_getPrice
2	PASS	SpecialOffer_getPrice
3	PASS	SpecialOffer_getPrice
4	PASS	SpecialOffer_getPrice
5	PASS	SpecialOffer_getPrice

Value	Name
3	aQuantity
50	aUnitPrice
(tSpecialOffer)(THREE_FOR_ONE_EURO+1)	anOffer
150	%

LDRA Testbed ® Dynamic Coverage Analysis Report

File : C:\LDRA_Workarea\GHS_MULTI4_C_CashRegister_tbwrkfls\GHS_MULTI4_C_CashRegister_asmwrkfls\sp ecialoffer.s

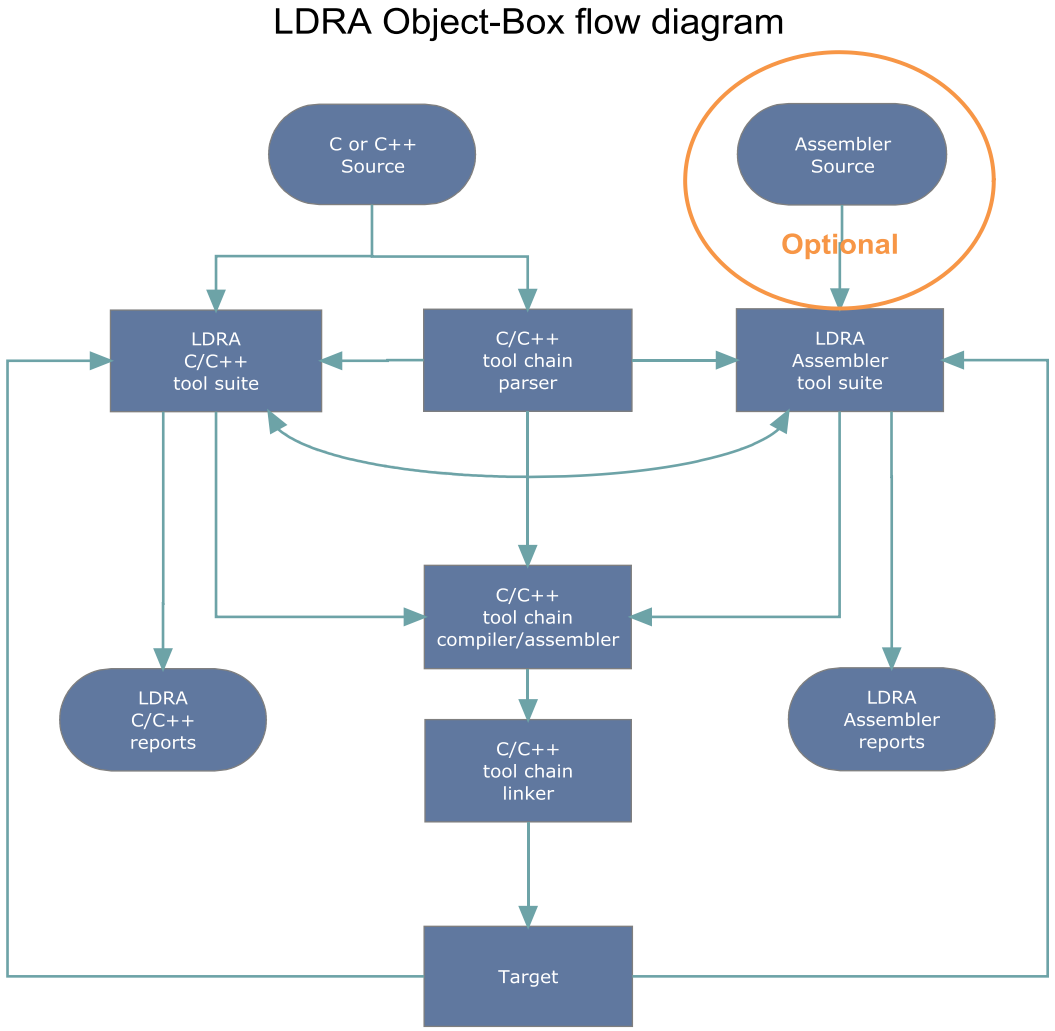
Overall Coverage Result (For File): Pass

Statement (TER1) = 100 % (Pass) Branch/Decision (TER2) = 100 % (Pass)

LDRA Object-Box

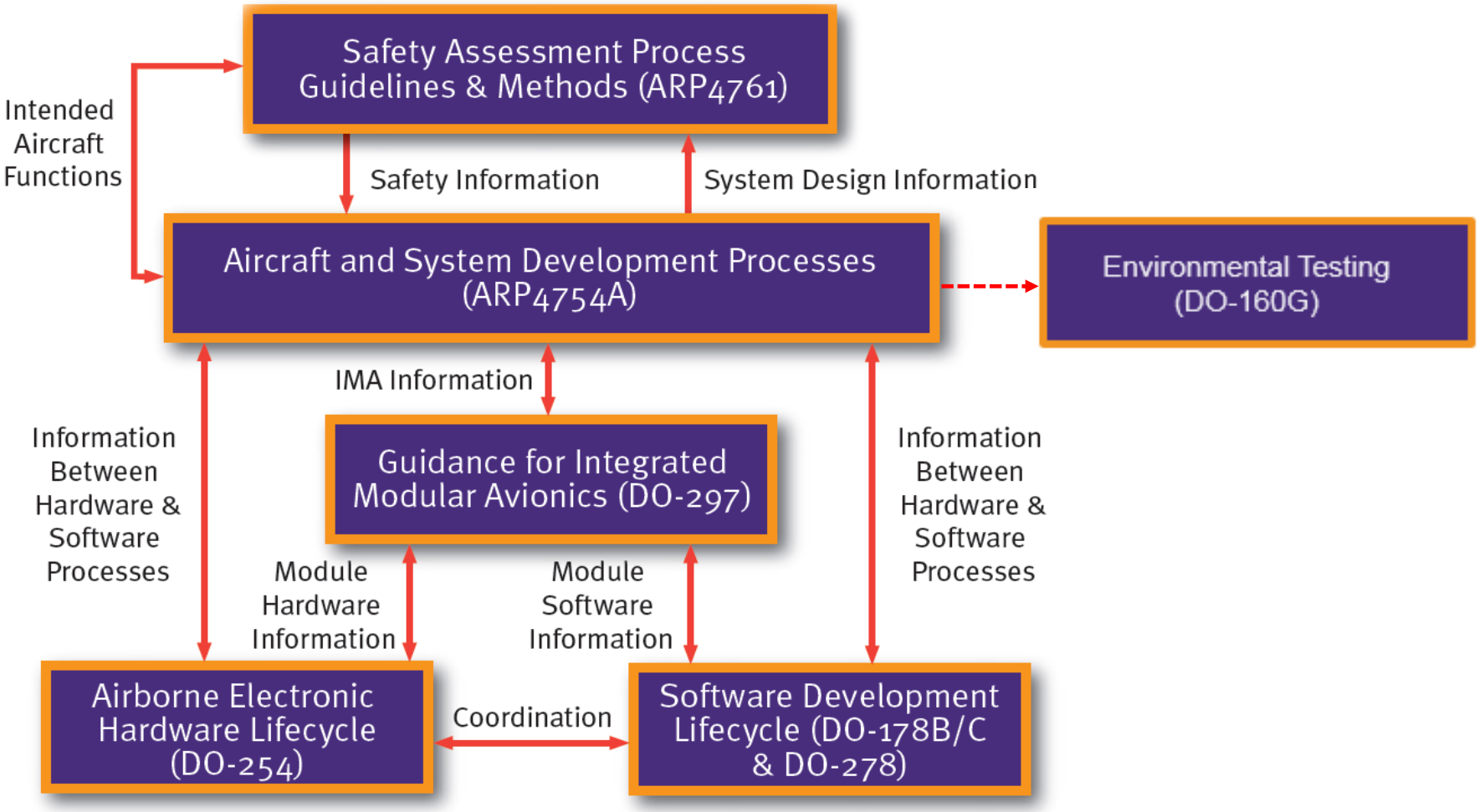
- Links the LDRA C/C++ tool suite with the appropriate LDRA assembler tool suite

- Tests can be run in:
  - Black Box
  - White Box
  - Red Box



Aspect #7:
Compliance Knowledgebase

Required Standards Coverage



Convert Standards Into Compliant Process

- But complicated and ambiguous guidance across multiple standards makes it hard!
  - In addition to DO-178C/DO-278A and DO-254, there are:
    - DO-178C/DO-278A supplements:
      - Model Based, OOT and Formal Methods
    - DO-254: Scope of standard has recently been expanded by EASA certification memo
    - DO-330 Tool Qualification
    - DO-248C– Rationale (FAQ's)
    - System & safety standards: ARP-4754A and ARP-4761
- Each standard requires detailed interpretation

Comprehensive FAA/EASA Knowledge Base

- Process Compliance Documents (PCDs)
  - 90% complete
  - Reduces project planning costs by 50%
  - Used by FAA to train Aircraft Certification Engineers
- Checklists
  - Assurance that ***all*** project transition criteria have been met
  - Checklists ask everything you need to know about applying DO-178 and DO-254 guidance
- LCMS PCDs and checklists have been used by hundreds of successful DO-178 and DO-254 FAA and EASA applicants over the past 15 years

Process Compliance Documents (PCDs)

- PCDs for all project deliverable documents
  - ARP 4754A (Aircraft systems)
  - DO-297 (Integrated modular avionics)
  - DO-178C (Airborne software)
  - DO-278A (Ground control software)
  - DO-254 (Complex electronic hardware)
- All planning, development & verification documents
- Comprehensive, up to 90% complete
- Approved by the FAA, EASA, TCCA
- Approved by all major OEM's

Aspect #8: Compliance Management System

Compliance Management System

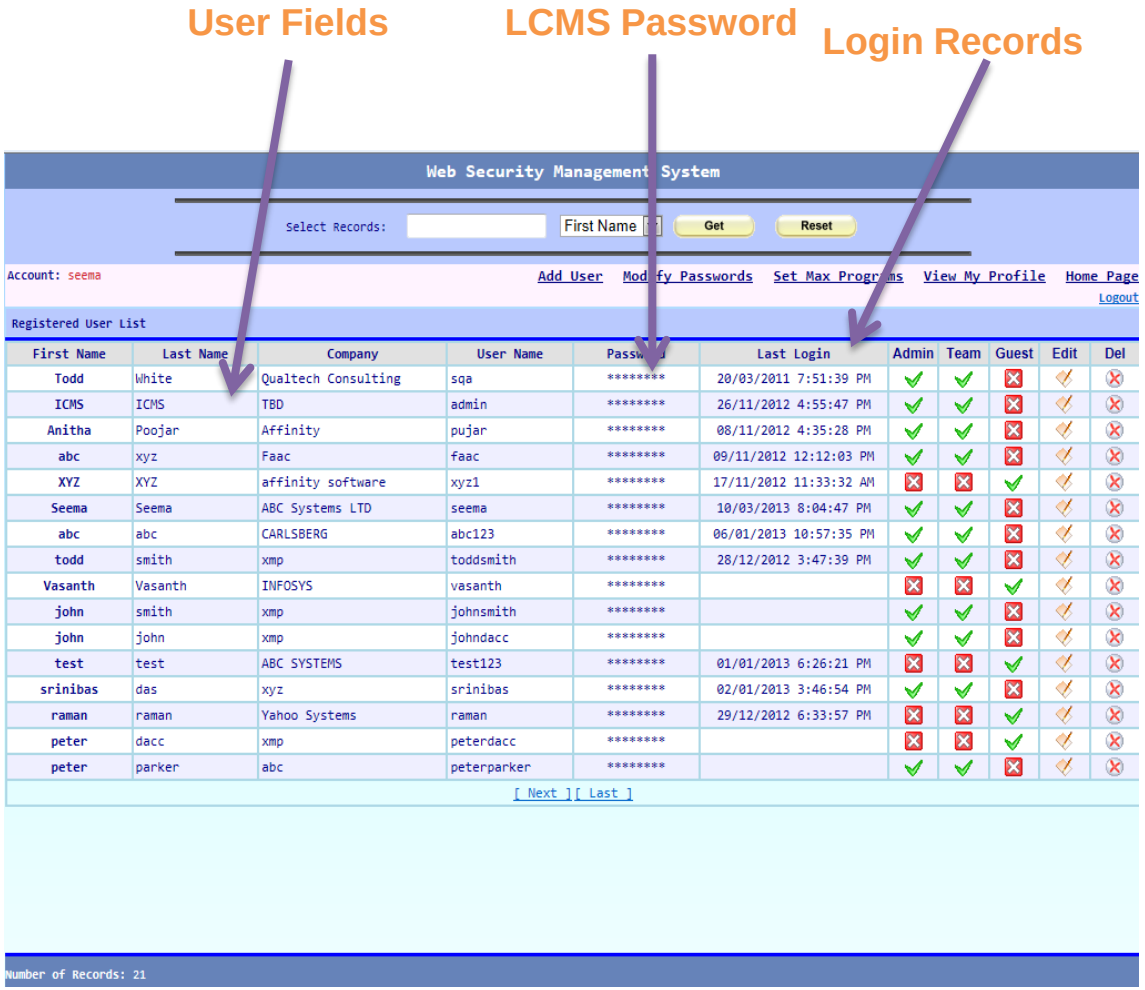
- DO-178C, DO-278A and DO-254 Lifecycle processes, not just the product, must be compliant
  - Though standards do not prescribe a particular process, your lifecycle processes must be:
    - Consistent
    - Repeatable
    - Transparent
  - You must have an infrastructure in place to support these objectives
- LDRA Compliance Management System (LCMS)

LDRA Compliance Management System

- Everything you need to organize and prepare for the Stages of Involvement (SOI) audits and to “show compliance”
- Used by Designated Engineering Representatives (DERs) and subject matter experts worldwide
- LCMS tools:
  - Security management system
  - Project management
  - Document review management
  - Reviews & analysis management
  - Problem reporting system
  - Archive management system

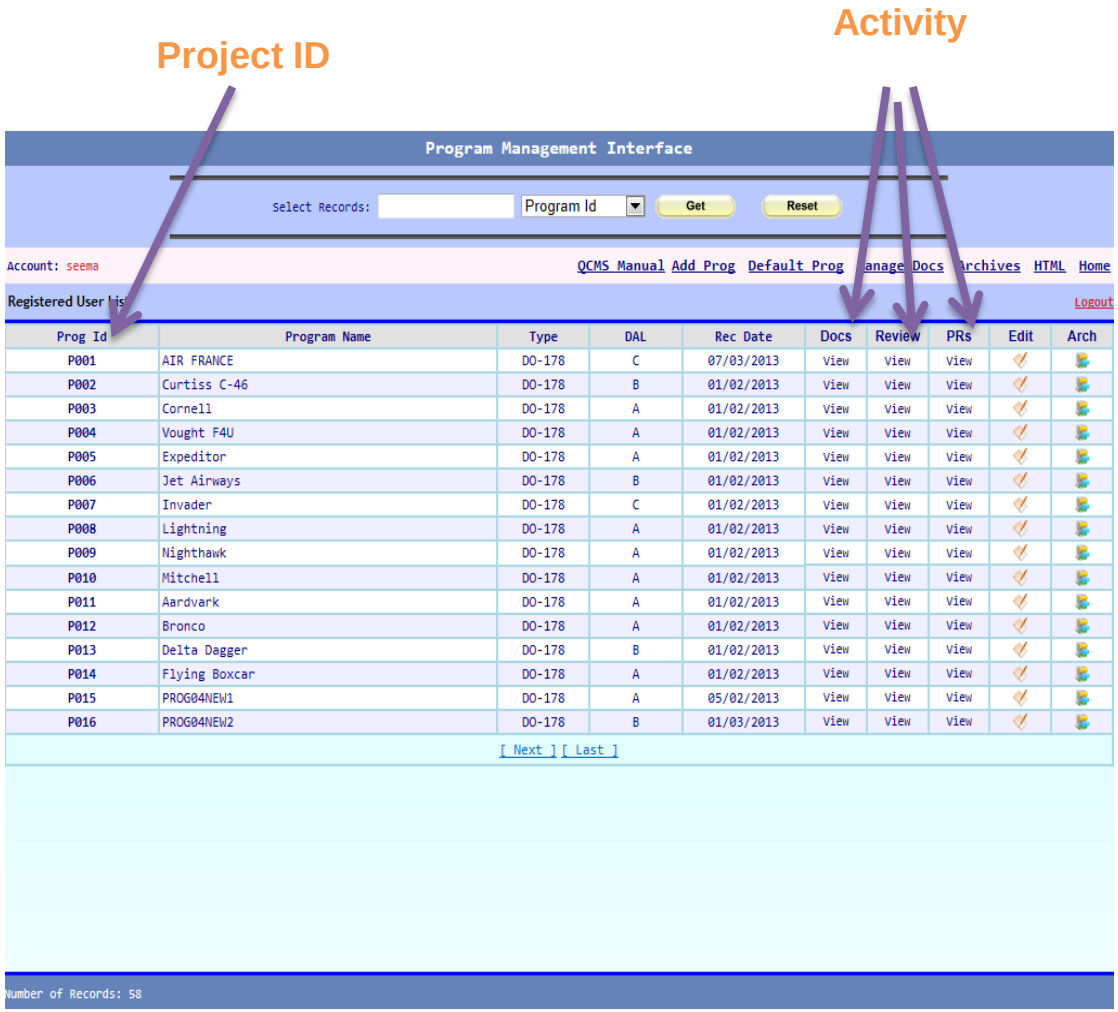
Security Management

- Named user tied to 32-bit secure login
- Supports project, admin and guest users
- Login and usage records automatically maintained



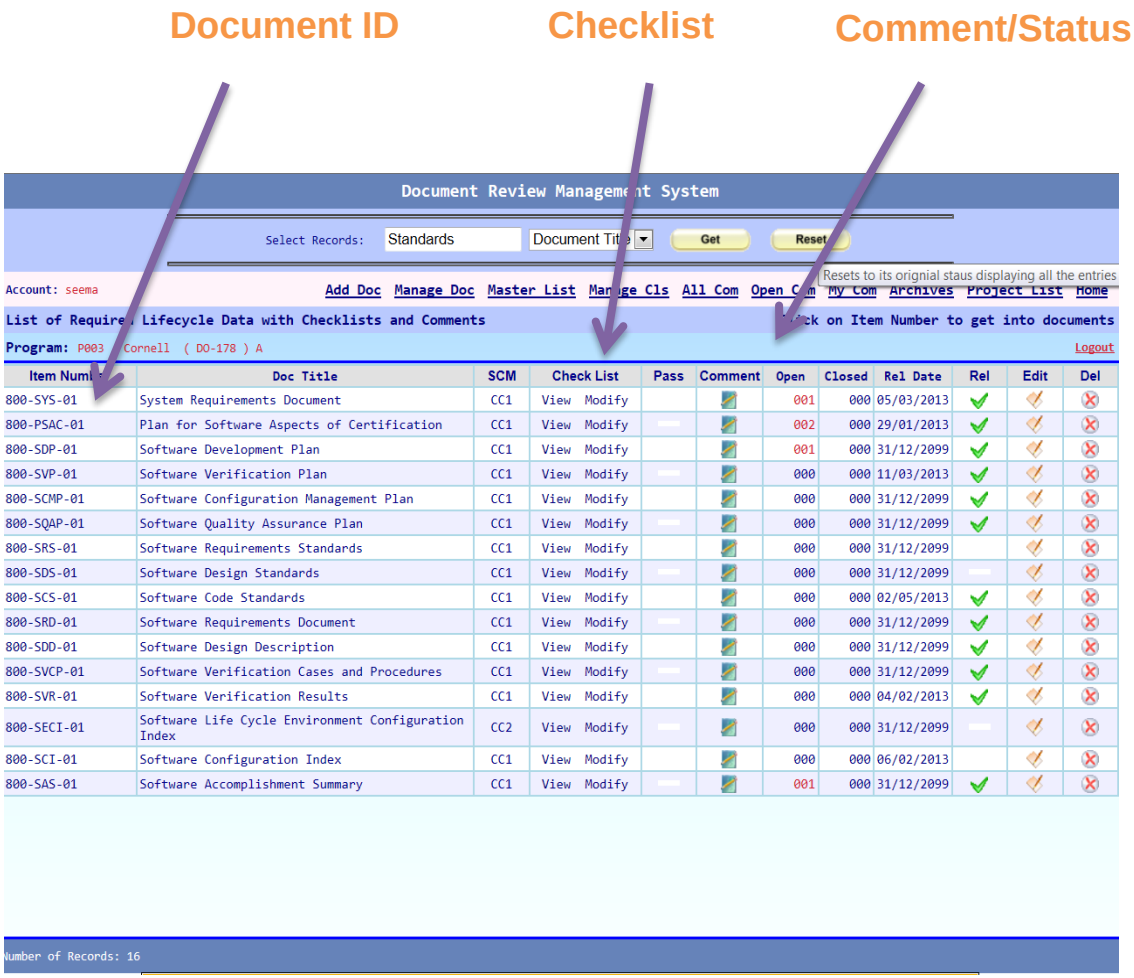
Project Management

- Select project
- Select activity
  - Document review
  - Action item review
  - Problem reporting
- Confirm project
  - Aero standard
  - DAL
  - Start date



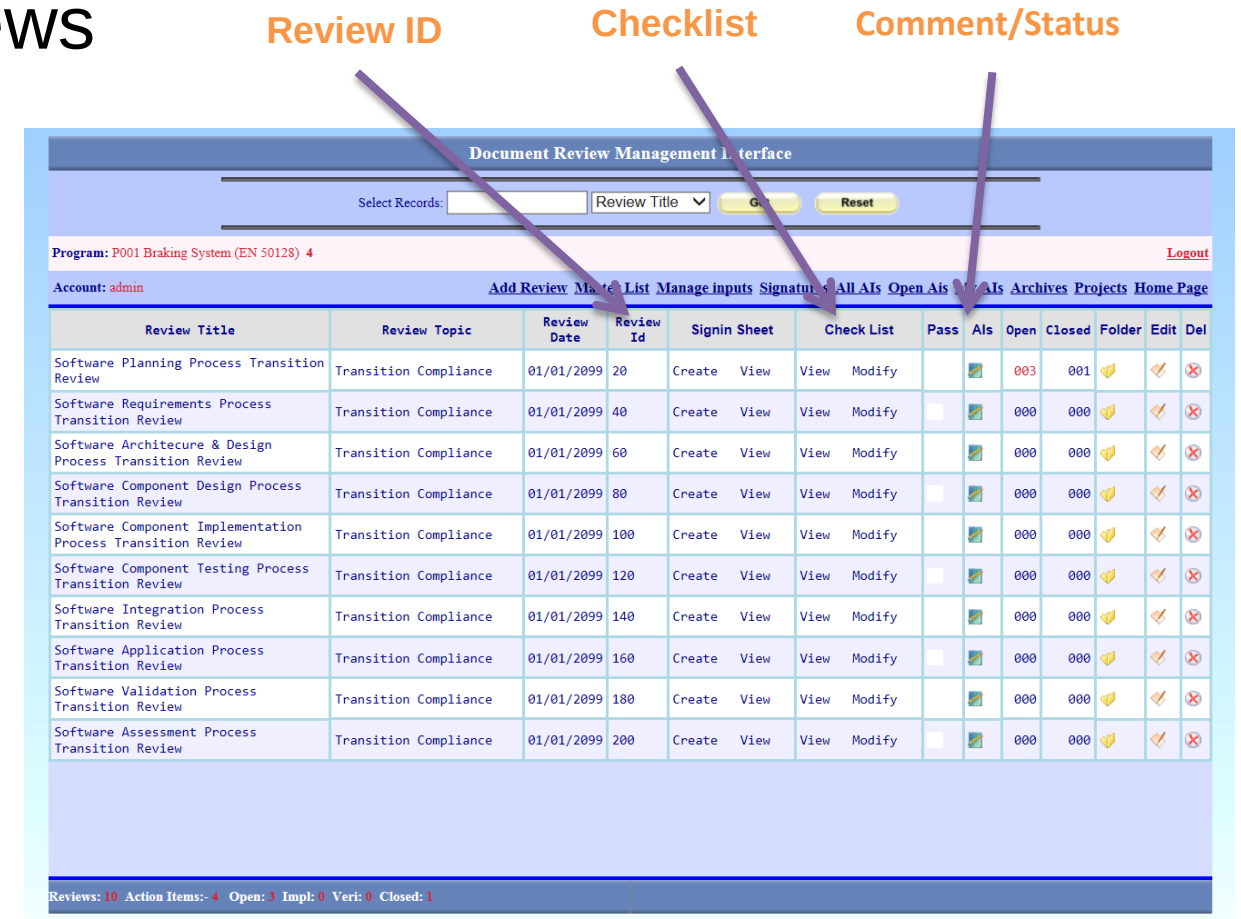
Document Review Management

- Review document
  - View / modify checklist
  - Comment on document
  - Track comments
  - Document approval status
  - Reviews performed by individuals
  - SCM category



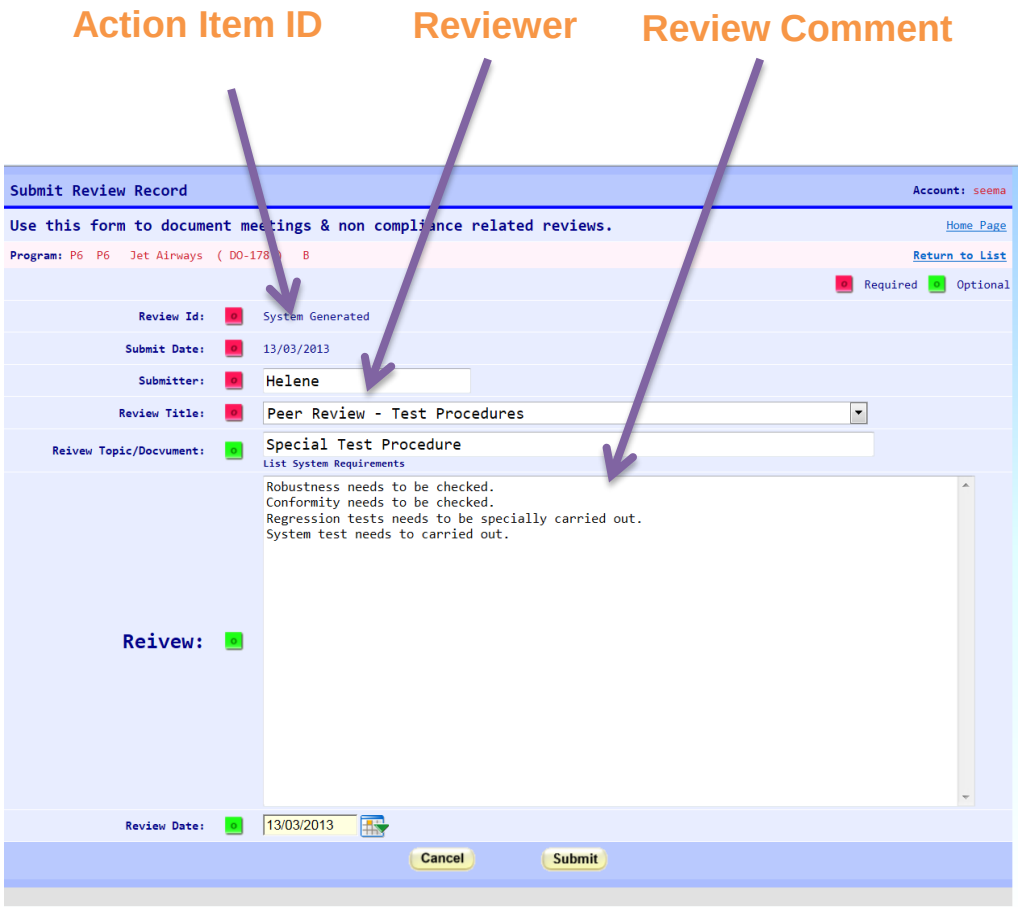
Reviews and Analysis Management

- Transition reviews
  - Planning
  - Development
  - Verification
- Peer reviews
  - Requirements
  - Design
  - Code
  - Verification



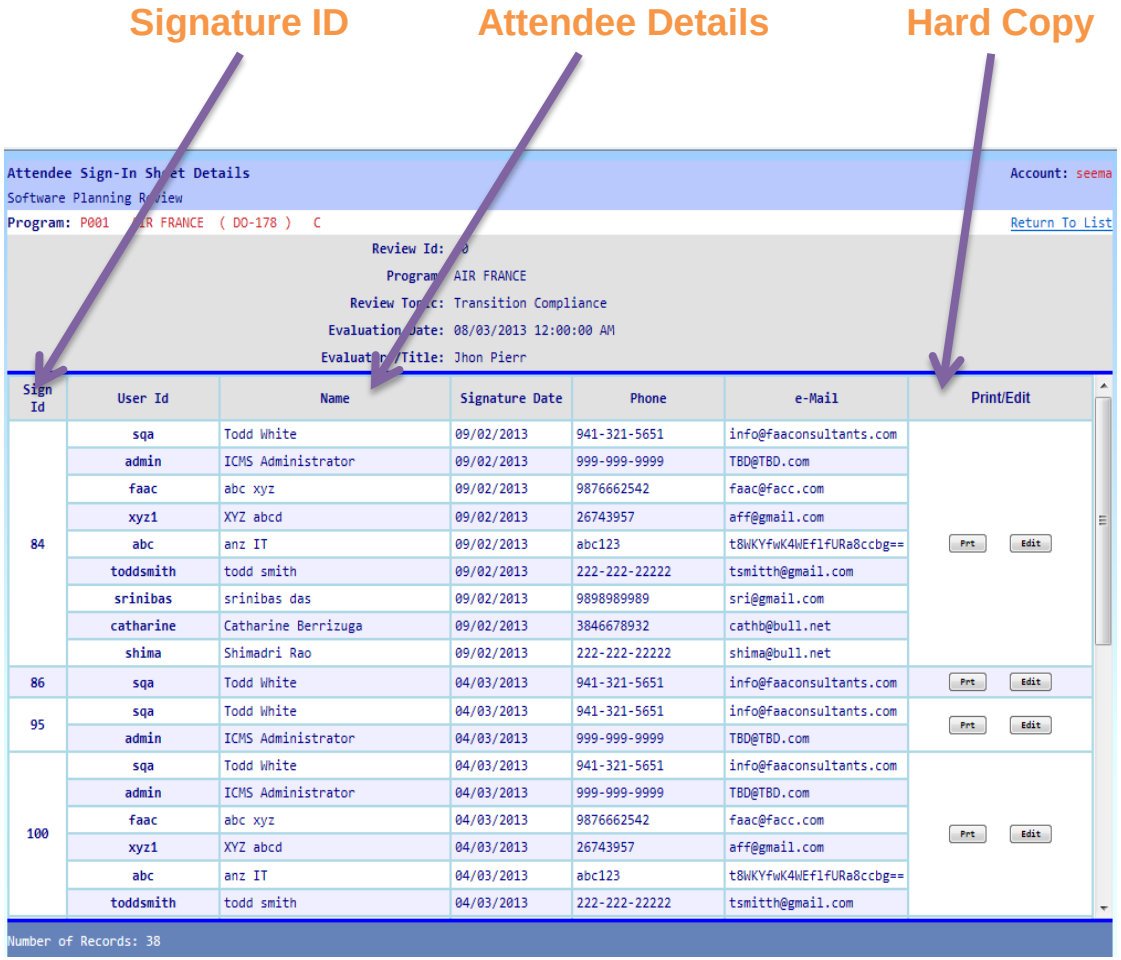
Action Item Review

- Document meetings, review issues and non-compliances
- System generated action item ID
- Review comment text entry
- Essential compliance records
- Automated attendee and sign-off lists



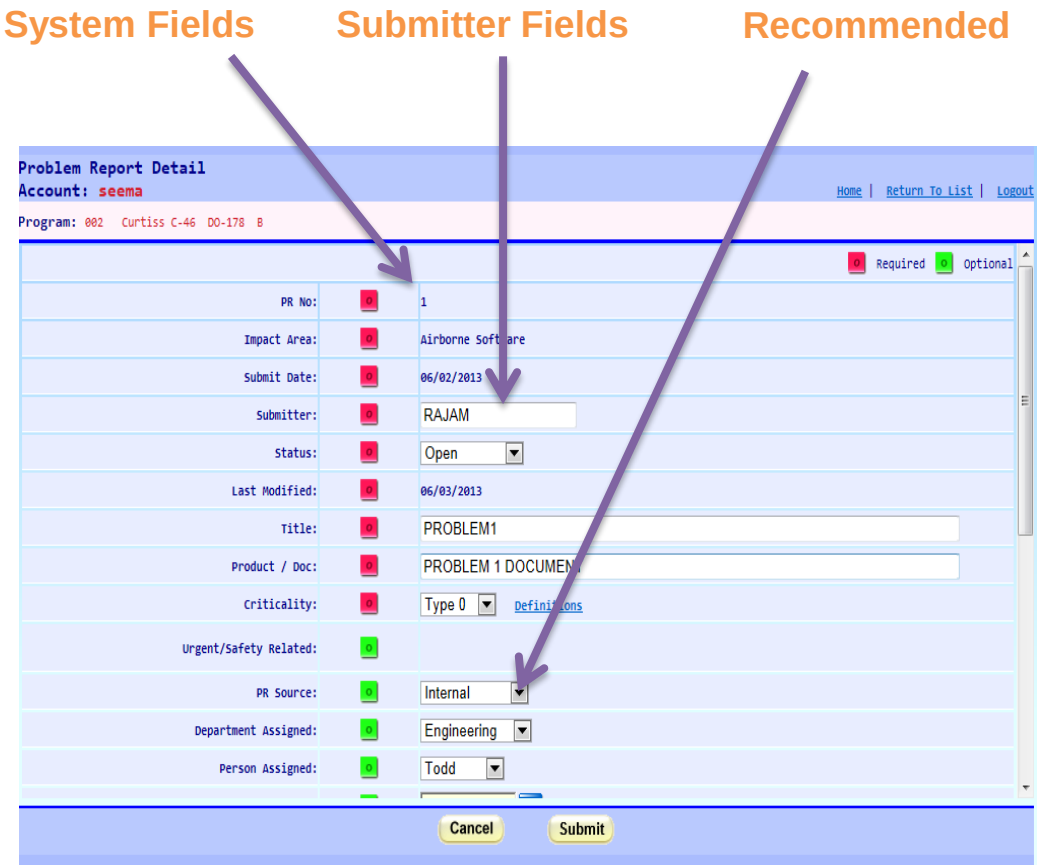
Review Attendee List

- Documents all attendee and contact info
- Identifies independent reviewers
- Links to signature sheet (sign ID)
- Prints hard copy signature sheet (Required by regulators)



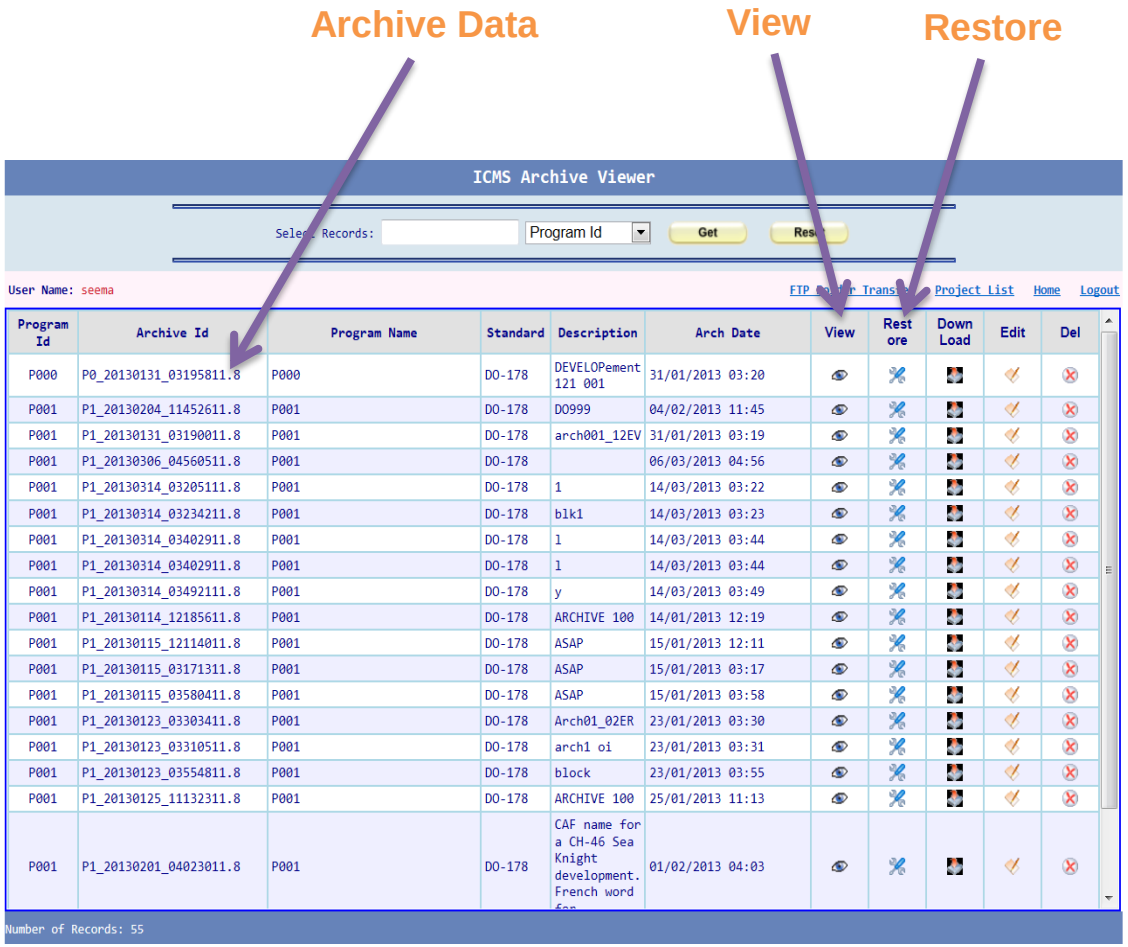
Problem Reporting Management

- Provides SC1/CC1/HC1 problem reporting
- Review comment text entry
- Required and Optional fields
- Automated email notifications
- Automated search



Archive Management

- Facilitates archiving of directories and files
- Also supports archive restore
- Provides “view” of archived files



Summary

- In order to meet the new DO-178C objectives and activities, we have looked at the following 8 aspects:

1. Traceability Management
2. Coding Standards
3. Model Based Development and Verification
4. Subclass Verification for Local Type Consistency
5. Data & Control Coupling Coverage
6. Object Code Verification
7. Compliance Knowledgebase
8. Compliance Management System

Questions & Answers

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

For further information:

www.ldra.com

info@ldra.com

@ldra_technology

LDRA Software Technology

LDRA Limited