North American Headquarters:

104 Fifth Avenue, 15th Floor
New York, NY 10011
USA

+1-212-620-7300 (voice)
+1-212-807-0162 (FAX)

European Headquarters:

46 rue d'Amsterdam
75009 Paris
France

+33-1-4970-6716 (voice)
+33-1-4970-0552 (FAX)

www.adacore.com

DO-178C: A New Standard for Software Safety Certification

SSTC 2010
Salt Lake City, Utah

Track 1
Monday, 26 April 2010
3:30 – 4:15 pm

Ben Brosgol • brosgol@adacore.com
Cyrille Comar • comar@adacore.com

AdaCore

Outline

DO-178B

- Summary
- Levels
- Life-Cycle Model
- Objectives
- Role of Testing
- Related documents

DO-178C

- Organization of revision effort
- Terms of Reference / rationale for approach
- Changes to Core Document
- Technology Supplements*
  - Tool Qualification
  - Model-Based Design and Verification
  - Object-Oriented Technology
  - Formal Methods

* Based on information available in February 2010

Safety-Critical Software: Background

What is "safety critical" software?

- Failure can cause loss of human life or have other catastrophic consequences

How does safety criticality affect software development?

- Regulatory agencies require compliance with certification requirements
- Safety-related standards may apply to finished product, development process, or both

Prescriptive

- Specify requirements on the *process* by which software is developed and fielded
  - Sound process adds confidence in soundness of result
- Example: DO-178B

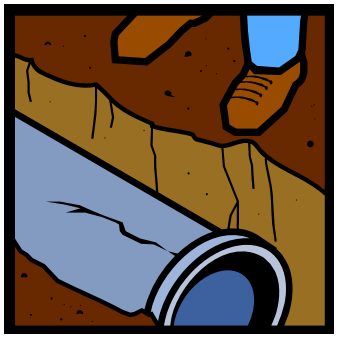
Goal-based

- Developer provides *safety cases*
  - *Claims* concerning system's safety-relevant attributes
  - *Arguments* justifying those claims
  - *Evidence* backing up the arguments
- Example: UK Defense Standard 00-56
  - "A Safety Case is a structured argument, supported by a body of evidence, that provides a compelling, comprehensible and valid case that a system is safe for a given application in a given environment"

Perspectives on DO-178B's Process-Based Approach

Quote from Gérard Ladier (Airbus), FISA-2003 conference

- "It is not feasible to assess the number or kinds of software errors, if any, that may remain after the completion of system design, development, and test"
- "Since dependability cannot be guaranteed from an assessment of the software product, it is necessary to have assurance on its development process"
- *"You can't deliver clean water in a dirty pipe"*



Quote from John Rushby, HCSS Aviation Safety Workshop, Oct 2006

- "Because we cannot demonstrate how well we've done, we'll show how hard we've tried"

DO-178B Basics

Software Considerations in Airborne Systems and Equipment Certification*, December 1992, published by RTCA

- EUROCAE** / ED-12B in Europe

Comprises a set of 66 "objectives" ("guidelines") for production of software for airborne systems

- Reliability: System does what it is supposed to do ⇨ no failures
  - Can trace each requirement to its implementing code and verification
  - No missing functionality
- Safety: System does not do what it is not supposed to do ⇨ no hazards
  - Can trace each piece of code back to a requirement
  - No additional functionality, no "dead code"
- Requires appropriate configuration management, quality assurance

"Level" of software establishes which objectives apply

---

* RTCA (www.rtca.org) is a U.S. Federal Advisory Committee whose recommendations guide FAA policy

** European Organisation for Civil Aviation Equipment (www.eurocae.org)

Software Levels Based on System Safety Assessment

Level A

- Anomalous behavior ⇨ catastrophic failure condition
  - "prevent continued safe flight and landing"

Level B

- Anomalous behavior ⇨ hazardous / severe-major failure condition
  - "serious or potentially fatal injuries to a small number of … occupants"

Level C

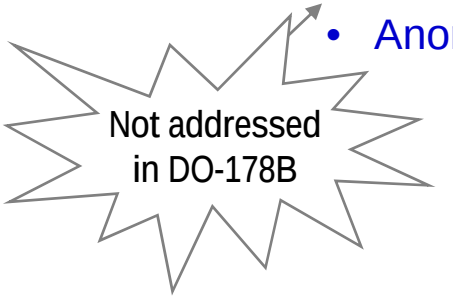
- Anomalous behavior ⇨ major failure condition
  - "discomfort to occupants, possibly including injuries"

Level D

- Anomalous behavior ⇨ minor failure condition
  - "some inconvenience to occupants"

Level E

- Anomalous behavior ⇨ no effect on aircraft operational capability or pilot workload



Structure / Goals / Usage

DO-178B guidelines organized into three major categories, each with a specified set of output artifacts

- Software Planning Process
- Software Development Processes
- "Integral" Processes

Appears oriented around new development efforts

- But may be applied to previously developed software, COTS, etc.

Strong emphasis on traceability

Implies traditional / static program build model

- Compile, link, execute

Used by FAA to approve software for commercial aircraft

- Developer organization supplies certification material
- Designated Engineering Representative ("DER") evaluates for compliance with DO-178B

"In a nutshell, what does this DO-178B specification really do?"*

- "It specifies that every line of code be directly traceable to a requirement and a test routine, and that no extraneous code outside of this process be included in the build"*

---

* Esterel Technologies, DO-178B FAQs, www.esterel-technologies.com/do-178b/

Other Aspects of DO-178B

Not specific to aviation

- Could be applied, in principle, to other domains (medical devices, etc.)

Includes glossary of terms

- “dead code”, “deactivated code”, “verification”, ...

Does not dictate

- Particular development process, design approach or notation
- Particular approach to hazard assessment (fault tree analysis, etc)
- Specific programming language(s) or software tools
- Requirements for personnel training
- Format for artifacts

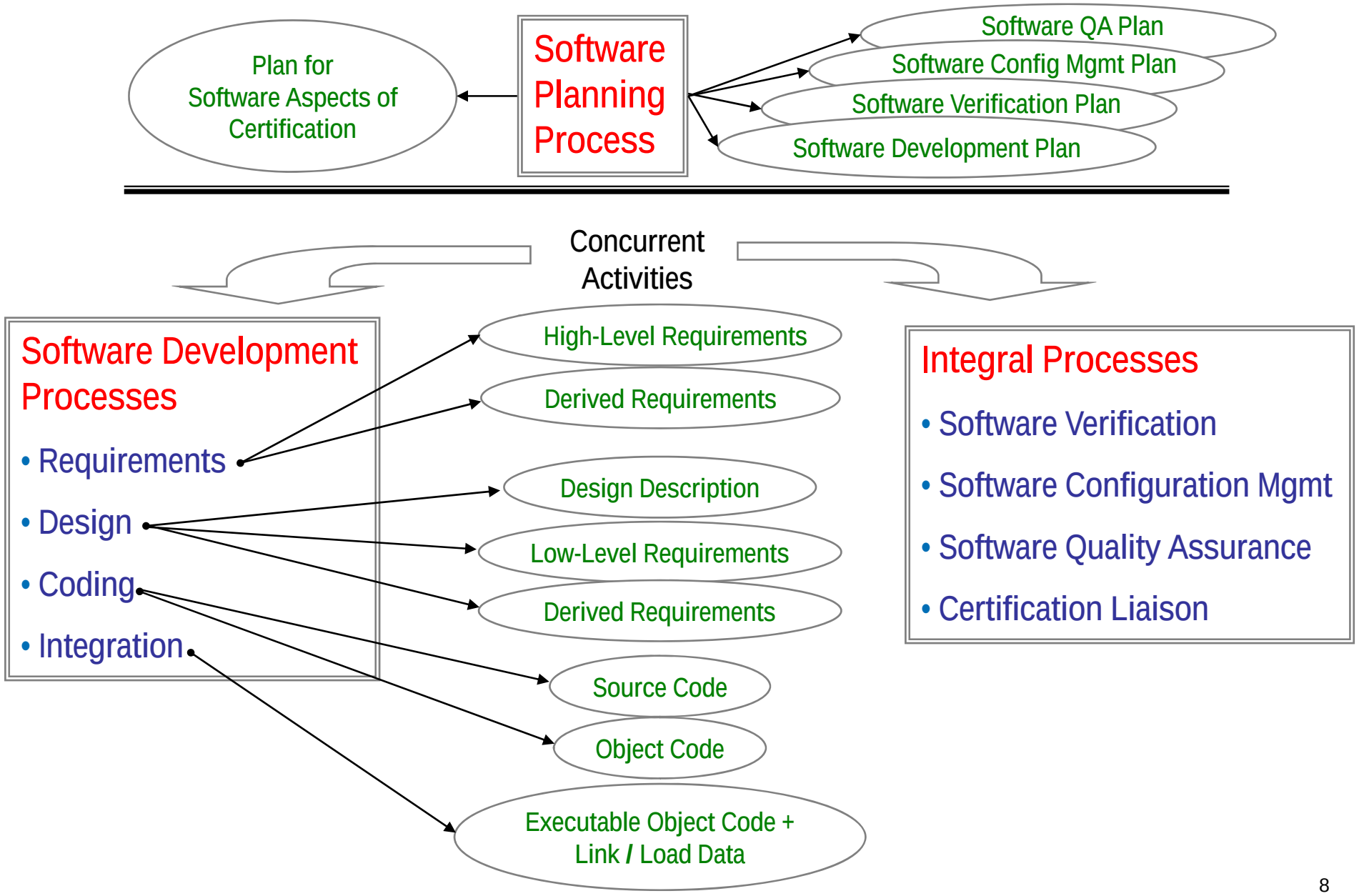
Tool qualification

- Ensures that tool provides confidence at least equivalent to that of the process(es) eliminated, reduced or automated
- Can qualify as a *verification tool* (bug may fail to detect errors but won’t introduce any) or as a *development tool* (bug may introduce errors)

What about security?

- No specific security-related objectives in DO-178B
- Work in progress under RTCA (SC-216) and EUROCAE (WG-72)

DO-178B Software Life Cycle Model



Summary of DO-178B "Objectives"

Process	Safety Level A	B	C	D
Software Planning Process	7	7	7	2
Software Development Process	7	7	7	7
Verification of Outputs of Software Requirements Process	3(ind) + 4	3(ind) + 4	6	3
Verification of Outputs of Software Design Process	6(ind) + 7	3(ind) + 10	9	1
Verification of Outputs of Software Coding & Integration Processes	3(ind) + 4	1(ind) + 6	6	0
Testing of Outputs of Integration Processes	2(ind) + 3	1(ind) + 4	5	3
Verification of Verification Process Results	8(ind)	3(ind) + 4	6	1
Software Configuration Management Process	6	6	6	6
Software Quality Assurance Process	3(ind)	3(ind)	2(ind)	2(ind)
Certification Liaison Process	3	3	3	3
Totals	66	65	57	28

Table shows number of objectives per process category
"ind" ⇨ need to show that objective is satisfied "with independence"

Sample DO-178B Objectives [Table A-5]

Verification of Outputs of Software Coding and Integration Processes

Objective	*Level*	*Output*
Source Code complies with low-level requirements	ABC	Software Verification Results
Source Code complies with software architecture	ABC	
Source Code is verifiable	AB	
Source Code conforms to standards	ABC	
Source Code is traceable to low-level requirements	ABC	
Source Code is accurate and consistent	ABC	
Output of software integration process is complete and correct	ABC	

Underlining of level ⇨ “objective should be satisfied with independence”

Some Issues related to Table A-5 Objectives [§6.3.4]

Reviews and Analyses of the Source Code

Conformance to standards

- Complexity restrictions
- Code constraints consistent with system safety objectives

Accuracy and consistency

- Stack usage
- Fixed-point arithmetic overflow and resolution
- Resource contention
- Worst-case execution timing
- Exception handling
- Use of uninitialized variables or constants
- Unused variables or constants
- Data corruption due to task or interrupt conflicts

Sample DO-178B Objectives [Table A-7]

Verification of Verification Process Results

Objective	*Level*	*Output*
Test procedures are correct	ABC	Software Verification Cases and Procedures
Test results are correct and discrepancies explained	ABC	Software Verification Results
Test coverage of high-level requirements is achieved	ABCD	
Test coverage of low-level requirements is achieved	ABC	
Test coverage of software structure (modified condition/decision coverage) is achieved	A	
Test coverage of software structure (decision coverage) is achieved	AB	
Test coverage of software structure (statement coverage) is achieved	ABC	
Test coverage of software structure (data coupling and control coupling) is achieved	ABC	

Underlining of level ⇨ “objective should be satisfied with independence”

Role of Testing in Software Verification

Test cases are to be derived from software requirements

- Requirements-based hardware/software integration testing
- Requirements-based software integration testing
- Requirements-based low-level testing

Test cases must fully cover the code

- Unexercised code may be due to any of several reasons
  - Missing requirement ⇨ Add new requirement
  - Missing test ⇨ Add new test case
  - Dead code ⇨ Remove it
  - Deactivated code ⇨ Show that it will not be executed
- Coverage on source versus object code
  - May be demonstrated on Source Code for Levels B and below
  - May be demonstrated on Source Code for Level A unless compiler generates object code not directly traceable to Source Code
    - Then need additional verification on object code to establish correctness of such generated code

Structural coverage is not “white box” testing

- Need to show that all exercised code is traceable to requirements

AdaCore

Required Coverage Depends on Safety Level

	Level A	Level B	Level C	Level D
Statement Coverage * Every statement has been invoked at least once	✓	✓	✓	
Decision Coverage * Described below	✓	✓		
Modified Condition / Decision Coverage * Described below	✓			

Decision Coverage

"Condition"

- A Boolean expression containing no Boolean operators; e.g., `X>0`

"Decision"

- A Boolean expression composed of conditions and zero or more Boolean operators; e.g., `X>0 and Y=2`
- If a condition appears more than once in a decision, each occurrence is a distinct condition
  - `X>0 and X>0` has *two* conditions

Decision coverage

- Every point of entry and exit in the program has been invoked at least once
- Every decision in the program has taken all possible outcomes at least once

```
if X>0 and Y=2 then
   Z := X+Y;
end if;
```

One test sufficient for statement coverage

X = 1, Y = 2 ⇨ `True` ⇨ if statement, assignment

Two tests sufficient for decision coverage

X = 1, Y = 2 ⇨ `True` ⇨ if statement, assignment

X = 0, Y = 2 ⇨ `False` ⇨ if statement

Modified Condition / Decision Coverage

MC / DC = Decision coverage + additional requirements

- Every condition in a decision in the program has taken all possible outcomes at least once
- Each condition in a decision has been shown to *independently affect* that decision's outcome
  - With all other conditions constant, changing the truth value of the condition changes the result of the decision

```
if X>0 and Y=2 then
   Z := X+Y;
end if;
```

X>0	Y=2	X>0 and Y=2
True	True	True
True	False	False
False	True	False
False	False	False

Need 3 tests* for MCDC

- (True, True), (True, False), (False, False)
- (True, True), (False, True), (False, False)

} Choose one of these test vectors

For further information see tutorial NASA / TM-2001-210876

- *A Practical Tutorial on Modified Condition / Decision Coverage*, by Kelly Hayhurst, Dan Veerhusen, John Chilenski, Leanna Rierson

* In general, at least *n+1* tests are needed for a decision with *n* conditions

Related Standards and Other Material

DO-248B

- Provides clarification of DO-178B guidance material (does not introduce new guidance)
  - 12 errata corrections
  - 76 Frequently-Asked Questions (FAQ)
  - 15 Discussion Papers

CAST Papers (Certification Authority Software Team)

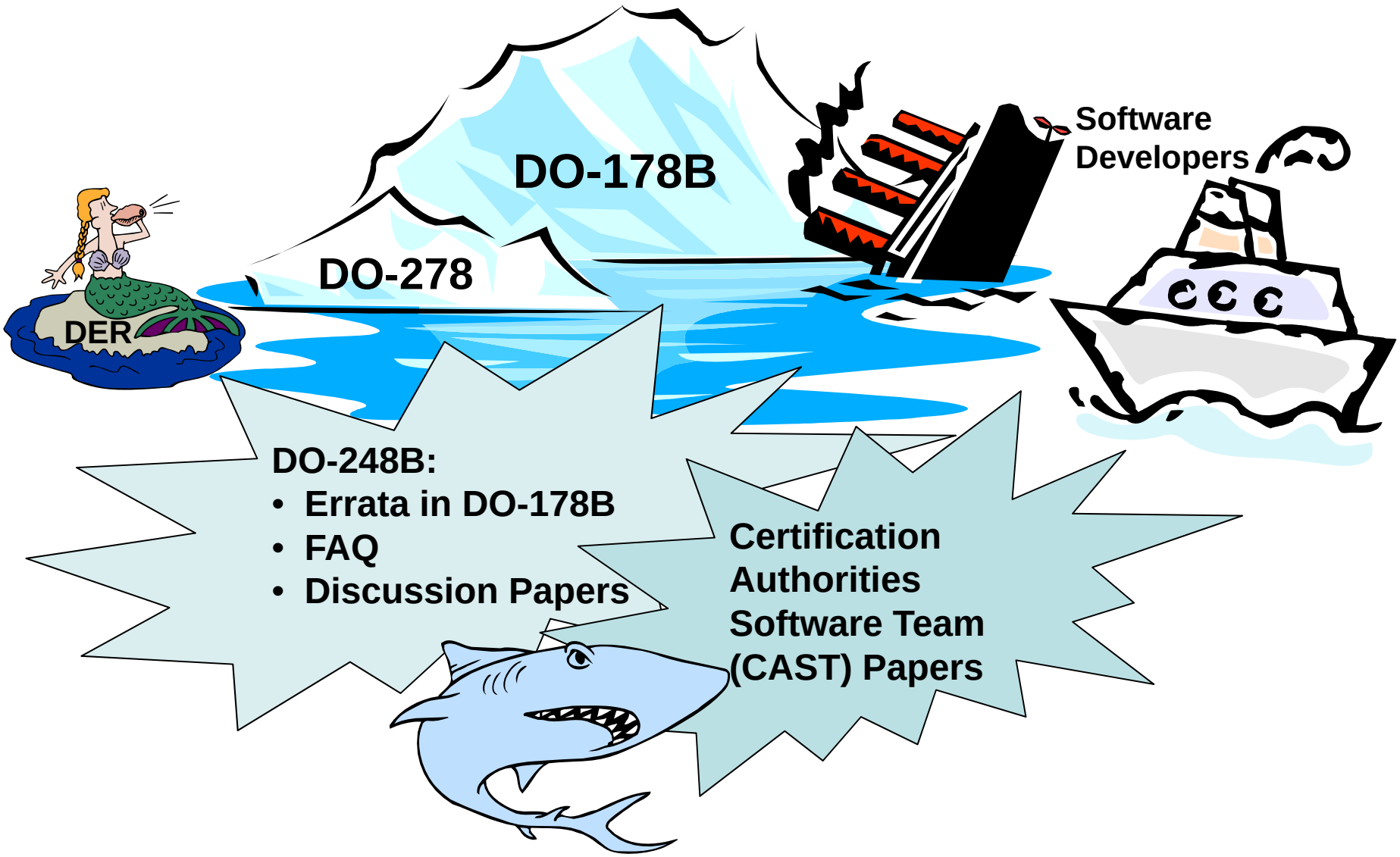
- Further clarification of DO-178B

DO-278

- Guidelines for Communication, Navigation, Surveillance, and Air Traffic Management (CNS/ATM) Systems Software Integrity Assurance
- Applies to software in CNS/ATM systems used in ground- or space-based applications that affects aircraft safety
- Largely based on DO-178B

DO-278 Assurance Level	DO-178B Software Level
AL1	A
AL2	B
AL3	C
AL4	No equivalent
AL5	D
AL6	E

Navigating the Software Safety Certification Waters



Why Revise DO-178B?

Address / accommodate "new" software technologies

- Object-Oriented Programming
- Model-based design / automatic code generators
- COTS software and tools, including real-time operating systems

Recognize that there is more to software verification than testing

- Formal methods
- Abstract interpretation

Consider combining with ground-based safety standard (DO-278)

Take into account the supplementary papers / commentaries on DO-178B

- Certification Authorities Software Team (CAST) papers
- Issues Papers (IPs)

Try to remove/reduce the need for DERs to make subjective judgments

Correct errors

- Example: requirements on development tool qualification don't distinguish the host environment (in which the tool runs) from the target environment (in which the system runs)

DO-178C "Terms of Reference"

Objectives

- Promote safe implementation of aeronautical software
- Provide clear and consistent ties with the systems and safety processes
- Address emerging software trends and technologies
- Implement an approach that can change with the technology

Activities (partial)

- Modify DO-178B
- Develop supplements to document technology-specific or method-specific guidance and guidelines
- Develop and document rationale for each DO-178B objective

Other considerations (partial)

- Maintain technology-independent nature of DO-178B objectives
- Modifications to DO-178B should:
  - Strive to minimize changes to existing text
  - Consider economic impact relative to system certification without compromising system safety
  - Address clear errors or inconsistencies / fill any clear gaps in DO-178B
  - Meet a documented need to a defined assurance benefit

AdaCore

Initial Controversy over DO-178C approach (1)

C. Michael Holloway (NASA), message posted to SC205/WG71 e-mail forum, 5 April 2005: *Are We Heading in the Right Direction?*

"The trend in the world (at least outside the U.S.) seems to be away from prescriptive standards and towards goal-based standards, which require the presentation of cogent arguments for safety (and other desired attributes), rather than the simple completion of certain prescribed processes.

This is a good trend. Although DO-178B/ED-12B has some attributes of a goal-based standard, for the most part it is a prescriptive standard of the sort that is increasingly difficult to justify on technical grounds.

The TOR's [Terms of Reference for DO-178C] insistence on minimizing changes to the current document is likely to ensure that DO-178C/ED-12C remains a mostly prescriptive standard. This is not, in my opinion, a good thing."

Initial Controversy over DO-178C approach (2)

Martyn Thomas (Praxis), message posted to SC205/WG71 e-mail forum, 10 August 2005: *The Right Direction?*

"I strongly support the proposal that safety standards should move away from prescribing specific processes to requiring evidence that the delivered system has the required properties. The relationship between development processes and the properties of the resulting systems is simply too weak to justify any confidence that a system will be safe just because it has been developed in the specific way prescribed by some standard.

...the core of safety certification must be formal reasoning about properties (ideally assisted by automated analysis tools). That, in turn, requires that the safety requirements are stated with mathematical rigour.

...certification should depend on a mathematically sound argument that the delivered system has the necessary safety properties..."

Rationale for DO-178C's Approach

Alternatives

- Major surgery on DO-178B
  - Remove prescriptive content, making document goal-oriented
  - Add supplements for requirements for specific approaches
- Minor surgery
  - Address new technologies in supplements when prescribed approaches may be replaced or augmented by others (e.g., formal verification)

Decision was to go with "minor surgery"

- Core document fixes known issues and adds clarifications (e.g. from DO-248B)
- Technology-specific supplements "add, delete or otherwise modify objectives, activities, explanatory text, and software life cycle data in DO-178C/ED-12C"

Reasoning

- DO-178B works
  - No commercial aviation deaths caused by software that had been certified to Level A
- Experience with goal-based approach has been mixed
  - Change of mindset required for both developer and regulator
  - Safety cases tend to focus on individual hazards, but most accidents are due to a combination of factors

DO-178C Development Framework (1)

Joint RTCA Special Committee 205 and EUROCAE* Working Group 71

- "SC-205 WG-71", officially abbreviated as "SCWG"
- Co-chairs: Jim Krodel (US), Gérard Ladier (Europe)
- Secretaries: Leslie Alford (US), Ross Hannan (Europe)
- Government agency representatives: Barbara Lingberg (FAA), Jean-Luc Delamaide (EASA)

Web site (registration required)

- ultra.pr.erau.edu/SCAS/

Subgroups

- **SG1: SCWG Document Integration** (Marty Gasiorowski / Ron Ashpole)
- **SG2: Issues and Rationale** (Fred Moyer / Ross Hannan)
- **SG3: Tool Qualification** (Leanna Rierson / Frédéric Pothon)
- **SG4: Model Based Design and Verification** (Mark Lillis / Pierre Lionne)
- **SG5: Object-Oriented Technology** (Greg Millican / Jan-Hendrik Boelens)
- **SG6: Formal Methods** (Kelly Hayhurst / Duncan Brown)
- **SG7: CNS/ATM and Safety** (Don Heck / David Hawken)
  - "CNS/ATM" = Communications, Navigation and Surveillance / Air Traffic Management

* European Organisation for Civil Aviation Equipment (www.eurocae.org)

AdaCore

DO-178C Development Framework (2)

Revision effort began in 2005, expected to complete in December 2010

- Two plenary meetings per year, alternating between US and Europe

The group is completely open, based on volunteer's participation

About 110 regular participants in 2009

- Many one-time visitors
- Some plenaries had more than 200 participants

About half from the US and half from Europe

- Europe: mostly Great Britain, France, Germany, Sweden
- Some participation from Brazil and recently China

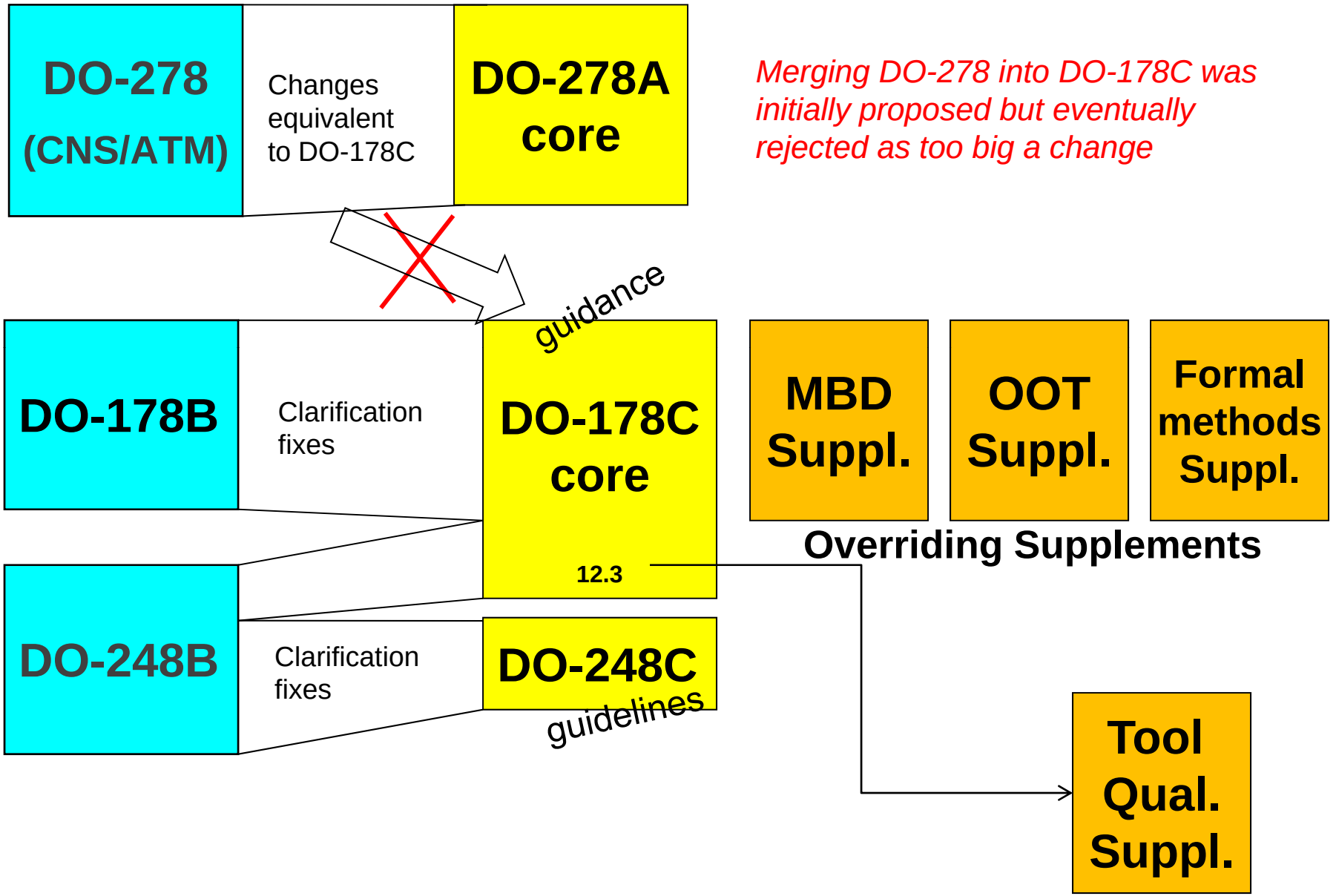
Three main kinds of organizations represented

- Airframe industry and contractors (Boeing, Airbus, Lockheed, Rockwell Collins, Honeywell, Thales, Eurocopter, …)
- Government agencies / certification authorities and DERs (FAA, EASA, NASA, Transport Canada, DGAC…)
- Tool vendors (LDRA, Esterel, Mathworks, AdaCore, Verocel, …)

Plenary vote for acceptance

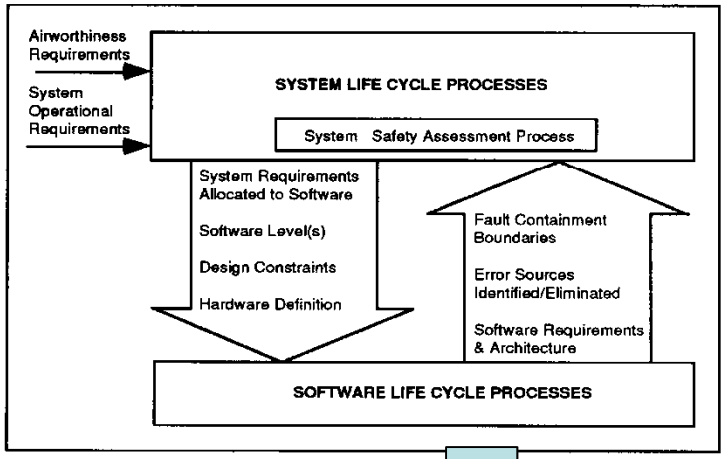
- "Almost-unanimity" is the rule

Expected Result(s)



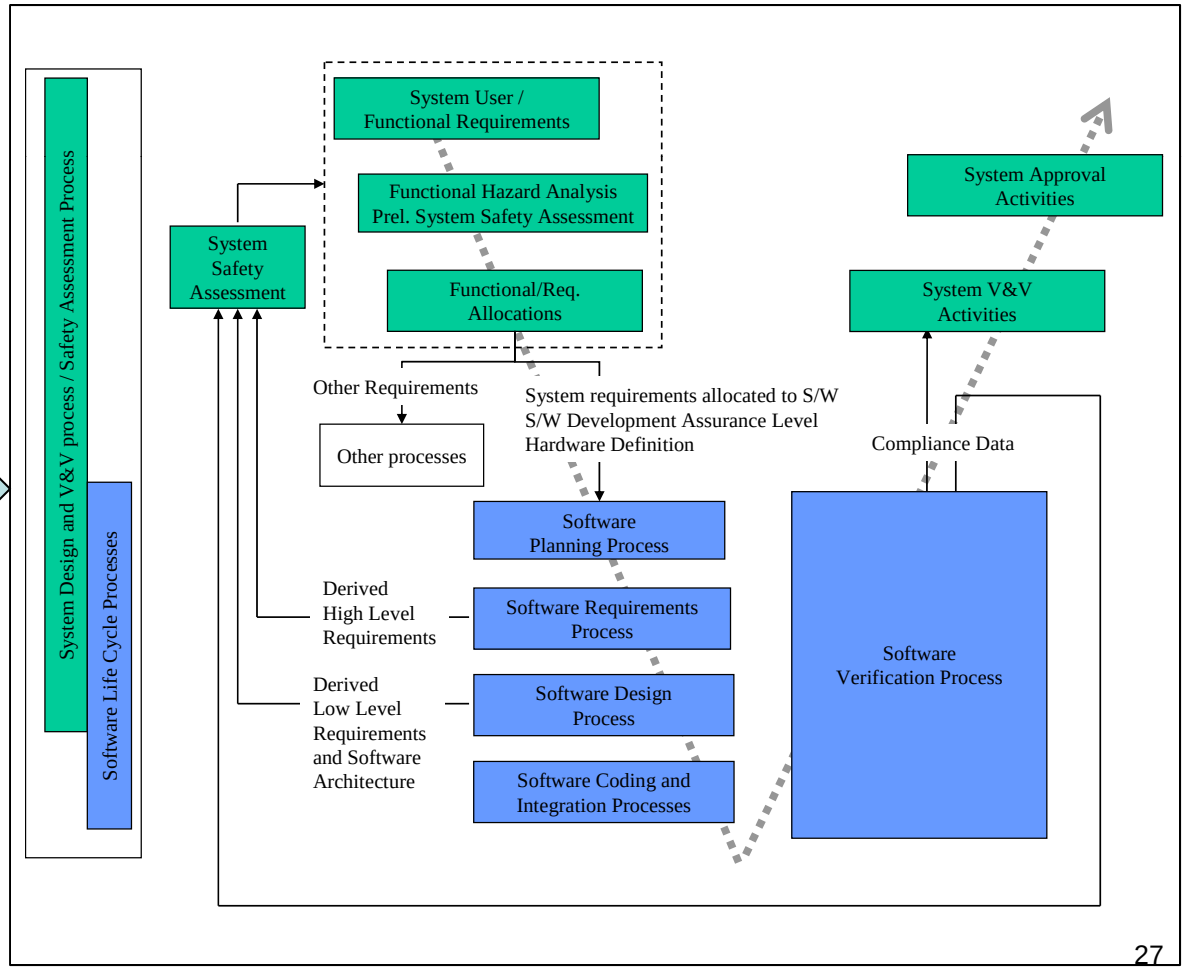
Merging DO-278 into DO-178C was initially proposed but eventually rejected as too big a change

Example of Change in Core Document: Figure 2-1



DO-178B

DO-178C
(IP 50, Rev. K)



Example of Change: Section 2.2

DO-178B

2.2 Failure Condition and Software Level

…

The failure condition of a system is established by determining the severity of failure conditions on the aircraft and its occupants. An error in the software may cause a fault that contributes to a failure condition. Thus, the level of software integrity necessary for safe operation is related to the system failure conditions.

DO-178C (IP 50, Rev. K)

2.2 System Safety Assessment Process and Software Development Assurance Level

…

The SWDAL of a software component is determined as part of the system safety assessment process by establishing how an error in a software component relates to the system failure condition(s) and the severity of that failure condition(s).

…

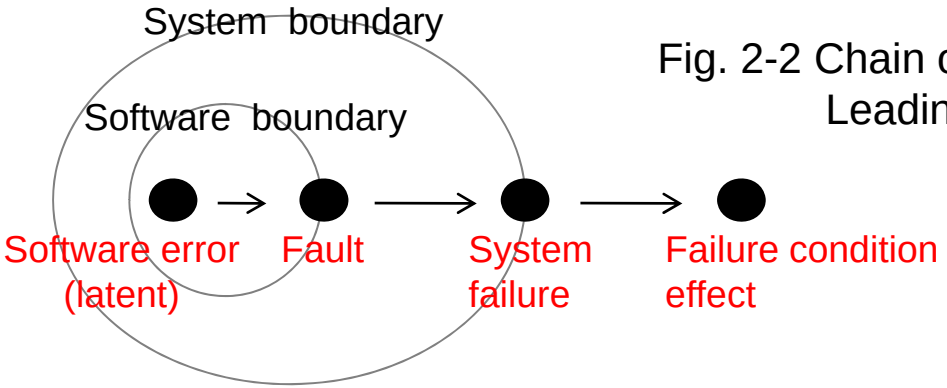


Fig. 2-2 Chain of Events for Software Error Leading to Failure Condition

Tool Qualification Supplement (1)

Qualification needed when processes are eliminated, reduced or automated without manual verification

DO-178B	DO-178C	
2 cases	**3 criteria and 5 levels (TQL)**	
Development Tool	**Criterion 1**	*Could insert an error*
	Criterion 2	*Could fail to detect an error and is used to reduce other development or verification activities*
Verification Tool	**Criterion 3**	*Could fail to detect an error*

Tool Qualification Supplement (2)

Example: Criterion 2 versus Criterion 3

Proof Tool → *Source code verification* → Criterion 3

\+

Reduce robustness testing → Criterion 2

Static Analysis Tool → *Source code review* → Criterion 3

\+

Remove defensive code → Criterion 2

Tool Qualification Supplement (3)

Mostly for Formal Methods & Model-Based Design

Software Level	Criteria		
	1	2	3
A	TQL-1	TQL-4	TQL-5
B	TQL-2	TQL-4	TQL-5
C	TQL-3	TQL-5	TQL-5
D	TQL-4	TQL-5	TQL-5

TQL 1: DO-178 level A

TQL 2: DO-178 level B

TQL3: DO-178 level C

TQL4 : Complete requirements
Describe architecture
More verifications

TQL5 : TOR verification

Tool Qualification Supplement has same structure as DO-178C

- Tool Planning Process, Tool Development Process, etc
- TQL (Tool Quality Level) analogous to SWDAL

Model-Based Development and Verification

A model is an abstract representation of system characteristics

- *Specification model* expresses high-level requirements (eg functions, performance, safety)
- *Design model* expresses low-level requirements / architecture
  - Data structures, data flow, control flow
- *Verification model* represents life cycle data of verification process

Benefits

- Unambiguous notation
- Supports use of automated code generation, automated test generation

Model-Based Development/Verification Supplement

- Acknowledges need for “Higher-Level Requirements” from which development model is derived
- Model usage affects most of the DO-178C-specified life cycle processes
- Traceability, etc, required for source program are required for design model
  - Demonstrate property for model, show that the property is preserved in the source code
- Model simulator may be useful tool

Object-Oriented Technology ("OOT")

What is OOT?

- Software development methodology supported by language features
  - Primary focus is on data elements and their relationships
  - Secondary focus is on the processing that is performed
- Applicable during entire software "life cycle"

Language concepts (OOP, or "Object-Oriented Programming")

- *Object* = state ("attributes") + operations ("methods")
- *Class* = module + object creation template
- *Encapsulation* = separation of interface (spec for methods) from implementation (state, algorithms)

Object-Oriented Design ("OOD")
also known as
Object-Based Programming

- *Inheritance* = specialization ("is-a") relationship between classes
  - Extend a class, adding new state and adding/overriding operations
- *Polymorphism* = ability of a variable to reference objects from different classes at different times
- *Dynamic binding ("dispatching")* = interpretation of operation applied to polymorphic variable based on current class of referenced object

Object-Oriented Technology ("OOT"), cont'd.

Additional OOP elements

- Single versus multiple inheritance
  - "Interface" for a simple form of multiple inheritance
- Use of constructors / destructors

Related language features

- Method overloading
- Type conversion
- Inline expansion

Other modern language features that complicate safety certification

- Generic templates
- Exceptions
- Concurrency

OO Technology & Safety Certification

Why consider OOT for safety-critical software?

- Data-centric approach eases maintenance of many large systems
- Model-driven architecture / UML tools may generate OO code to be certified
- Many programmers know OO languages such as C++, Java, or Ada 95
- Languages (such as Ada) used for safety-critical systems have OO features
- May want to take OO legacy code and apply DO-178 *à posteriori*
- Issues explored at NASA/FAA Object-Oriented Technology in Aviation (OOTiA) workshops

What's the catch?

- Paradigm clash
  - OOT's distribution of functionality across classes, versus safety analysis's focus on tracing between requirements and implemented functions
- Technical issues
  - The features that are the essence of OOP complicate safety certification and raise security issues (e.g. ensuring integrity of "vtable")
  - Dynamic memory allocation, VM implementations
- Cultural issues
  - Many DERs / TOE evaluation personnel are not language experts and are (rightfully) concerned about how to deal with unfamiliar technology

OOT Supplement

New OOT-specific objectives and activities for Software Verification Process

- Based on Liskov Substitution Principle:
  - "Let *q(x)* be a property provable about objects *x* of type *T*. Then *q(y)* should be true for objects of type *S* where *S* is a subtype of *T*"

OO.6.5 Local Type Consistency Verification
The use of inheritance with method overriding and dynamic dispatch requires additional verification activities that can be done either by testing or by formal analysis.

OO.6.5.1 Local Type Consistency Verification Objective
Verify that all type substitutions are safe by testing or formal analysis.

OO.6.5.2 Local Type Consistency Verification Activity
For each subtype where substitution is used, perform one of the following:
- formally verify substitutability,
- ensure that each class passes all the tests of all its parent types which the class can replace, or
- for each call point, test every method that can be invoked at that call point (pessimistic testing).

- New guidance for dynamic memory management verification
  - Need to verify absence of problems such as dangling references, fragmentation, storage exhaustion, unbounded allocation or deallocation time

New objectives and activities related to virtualization (OO.E.1.7)

- Issue of code versus data (objectives generally apply to code, not data)

Virtualization defines an intermediate execution platform…. One must ensure that all data that is used as executable code for the execution platform be verified as executable code.

Formal Methods Supplement: Concepts

Formal Method = Formal Model + Formal Analysis

- May be applied at various stages in software development
- Formal methods are used as a verification technique

Formal Model

- System abstraction with unambiguous, mathematically defined syntax and semantics
- Examples
  - Graphical models (state machines)
  - Text-based (Z, set theory, programming language subsets)
- May be used to capture some system properties (e.g. Worst-Case Execution Time)

Formal Analysis

- Provides guarantees/proofs of software properties, compliance with requirements
- An analysis method can only be regarded as formal if it is sound
  - It never asserts that a property holds if it is possible for the property to not hold
  - Converse is a usability issue
- Kinds of formal analysis
  - Deductive (theorem proving)
  - Model checking
  - Abstract interpretation

Implemented in (qualified) tools

Formal Methods Supplement*

Mainly adapts Software Verification Process section of DO-178C

- Goal is to prevent and eliminate requirements, design and code errors throughout the software development processes

Formal methods are *complementary* to testing

- Testing shows that functional requirements are satisfied and detects errors
- Formal methods can increase confidence that no anomalous behavior will occur
  - May find faults that are not detected by testing

Formal methods cannot establish verification evidence for the target hardware

- Therefore testing on the target is still required
- But: formal analysis of source code can be used to reach [compliance with Low-Level Requirements] provided that complementary analyses show the *property* preservation between source code and object code

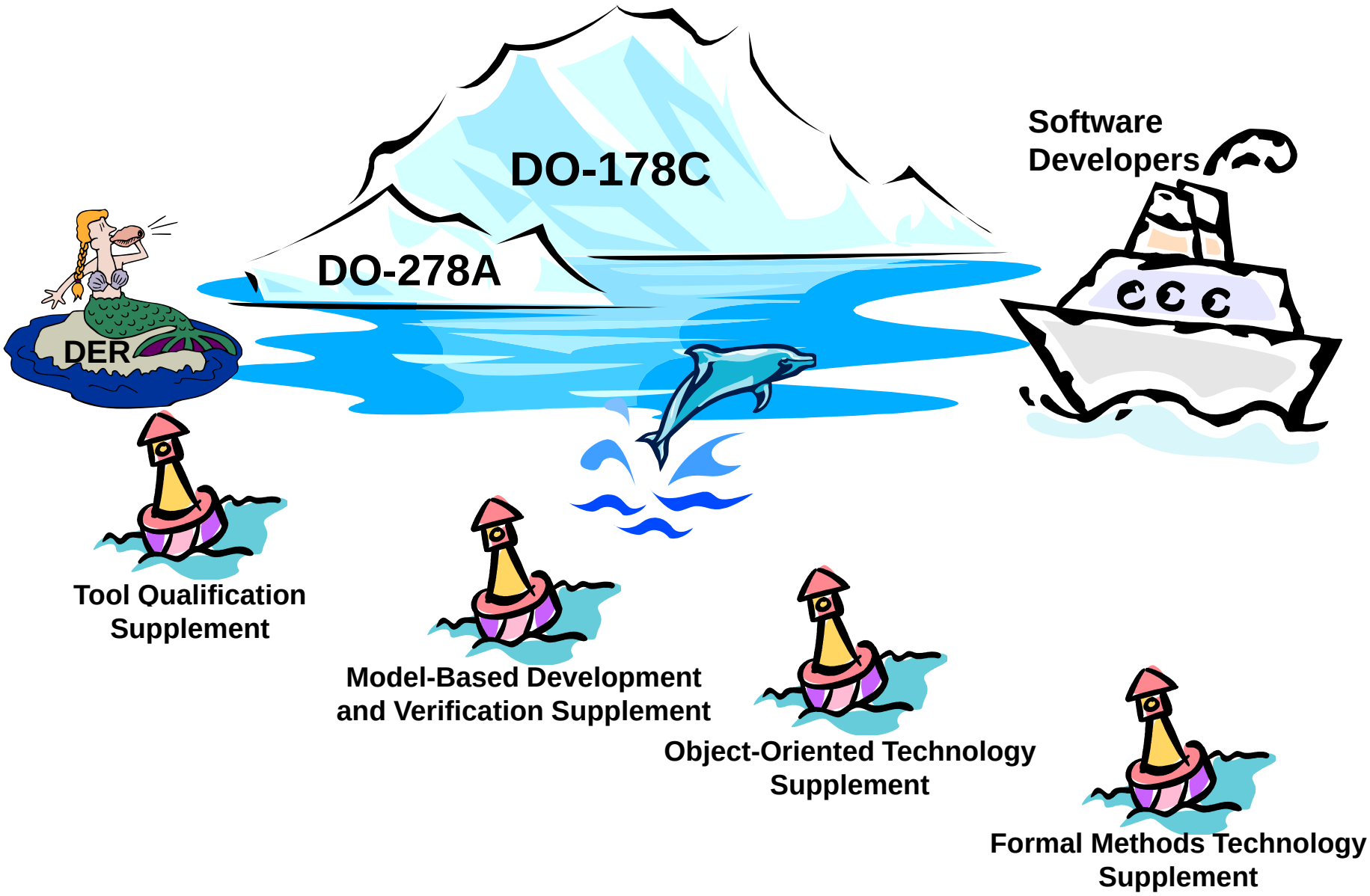
Uses of Formal Methods

- Formal specification of life-cycle artifacts
- Formal derivation of life-cycle artifacts
- Formal analysis of life-cycle artifacts

---

* This slide largely derived from *Working towards DO-178C/ED-12C, DO-248C/ED-94C and DO-278A/ED-109A*, invited presentation by James Chelini (Verocel) at ACM SIGAda 2009

Navigating the Software Safety Certification Waters



References / Web Resources

DO-178B

- RTCA SC-167 / EUROCAE WG-12. RTCA/DO-178B – *Software Considerations in Airborne Systems and Equipment Certification*, December 1992
- Certification Authority Software Team (CAST): www.faa.gov/aircraft/air_cert/design_approvals/air_software/cast/cast_papers/

DO-178C

- SC-205/WG-71 website: forum.pr.erau.edu/SCAS

Open-DO

- Community initiative to promote open-source software and lean/agile methods in developing and certifying high-integrity systems
- www.open-do.org

University of York (UK) Safety-Critical Mailing List Archives

- www.cs.york.ac.uk/hise/safety-critical-archive

Object-Oriented Technology and Safety Certification

- *Handbook for Object-Oriented Technology in Aviation (OOTiA)*, October 2004. www.faa.gov/aircraft/air_cert/design_approvals/air_software/oot

Acronyms and Abbreviations

ATM Air Traffic Management

CAST Certification Authority Software Team

CNS Communications, Navigation, and Surveillance

COTS Commercial Off-The-Shelf

DO-178B, DO-178C *[Not acronyms, these are names/numbers of documents]*

DER Designated Engineering Representative

EUROCAE European Organisation for Civil Aviation Equipment

FAA Federal Aviation Administration

MC/DC Modified Condition/Decision Coverage

OOP Object-Oriented Programming

OOT Object-Oriented Technology

OOTiA Object-Oriented Technology in Aviation

RTCA *[Not an acronym, this is the name of an organization]*