

CSI

*Model-Based Development (MBD)
Coming soon to a theater near you*

Resolving MBD against DO-178B



Mike DeWalt
Chief Scientist, Aviation Systems
Certification Services, Inc.
+1.360.376.8110 voice
Mike.DeWalt@certification.com

Certification Services, Inc.

2005 Software/CEH Conference

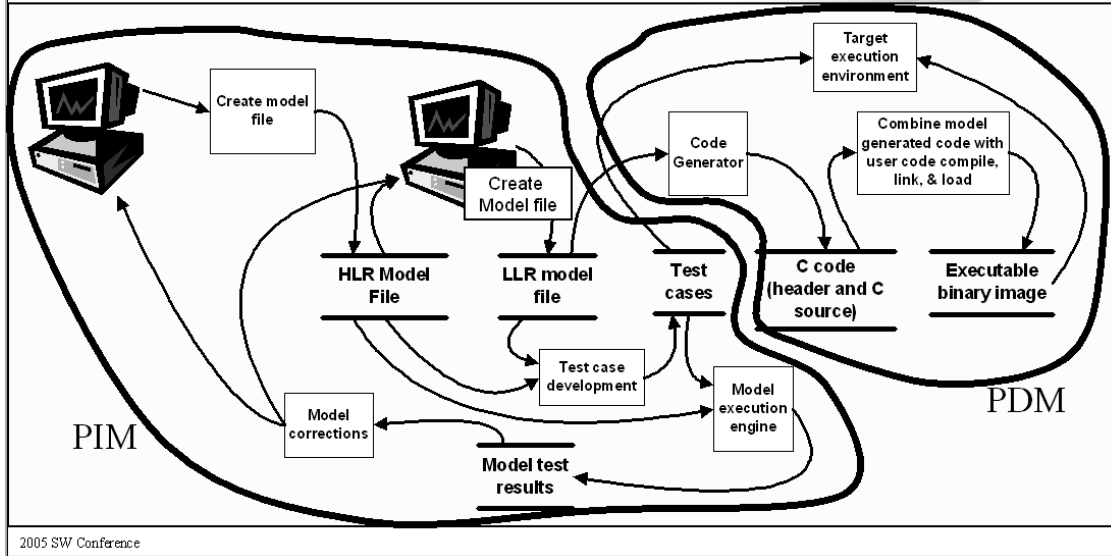
Goals



- Graphical vs. textual debate
- High- and low-level requirements, tool use
- Tool qualification
- Tool credit and DO-178B
- The great execution debate
- Examples

A tool for all seasons

- MBD with and without tool qualification



This slide represents a typical avionics use of model-based development. Model-based development might or might not require tools. It might or might not have a graphical user interface. It might or might not need qualification activities.

Users specify desired system behavior graphically at a standard workstation terminal. Results are encoded as a model that the user can study on the screen. Such modeling can address high-level requirements, low-level requirements, and, in some cases, portions of an architecture description. (Most models in this form are incomplete and must be supplemented with additional specifications through other means.) This can be considered the Platform Independent Model (PIM). That is, behavior is abstracted and independent of any real execution mechanism. Code can be created from the model, either by a tool or by hand. The code is then a platform-unique, executable representation of the model, usually referred to as the Platform Dependent Model (PDM).


Once the model is captured, a couple of paths can be taken.

One path is to use some combination of formal methods and heuristics to examine the correctness of the model. A list of desirable properties can be specified, and the model can be examined to determine if the properties are present. For example, an autoland system should never disconnect for any single sensor failure below the alert height. That process of comparing the desired properties of a system with its implementation, typically called model checking, is discussed in another presentation during this conference.

The more common path is creation of test cases covering the high-level and low-level requirements for both logical and continuous functions. Those cases can execute the model directly in a simulated or interpreted mode. Such an approach can detect some design errors missed by traditional manual review of the documentation -- unsurprising, given that such reviews usually precede production of source code. A nominally equivalent approach may also be pursued more conventionally, developing source code, generating object code, and performing the tests in the target machine. This generally implies additional effort and schedule burdens. The relationship between PIM testing and PDM testing will be discussed later.

CSI

Graphical vs. textual debate

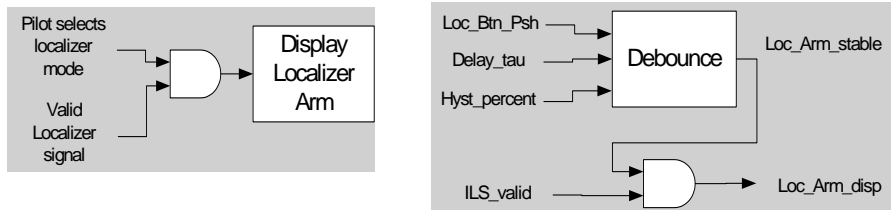


CSI *DO-178B definitions of high- and low-level requirements*

- High-level requirements - Software requirements *developed from* analysis of *system* requirements, *safety*-related *requirements*, and system *architecture*.
- Low-level requirements - Software requirements *derived from high-level* requirements, *derived* requirements, and design constraints *from which source code can be directly implemented without further information*.
- Assumes continuous refinement process with constraints on lower abstraction levels – examples a/c → Sys → SW...

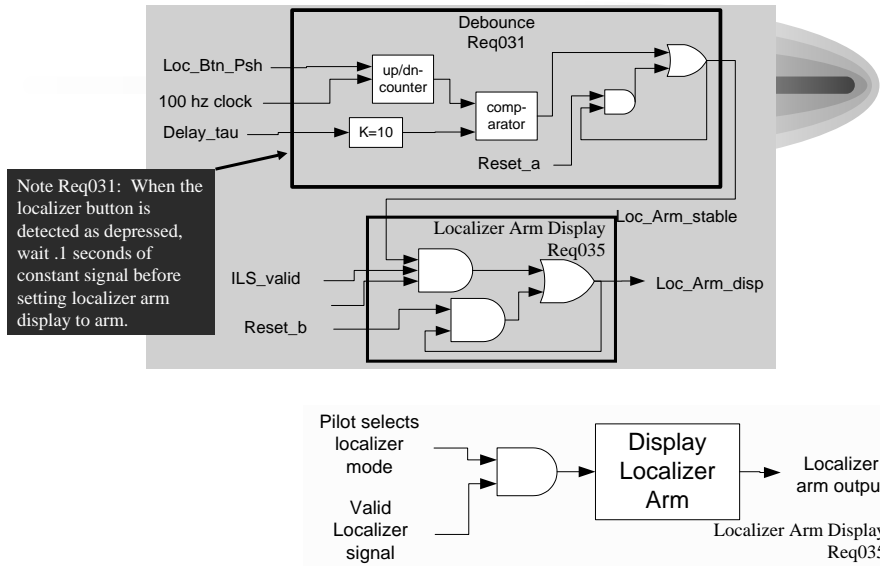
Equivalent high-level requirements representations

- A. The airplane shall display the localizer-arm indication when the pilot has selected the localizer mode and there is a valid localizer signal.
- B. The autopilot shall display the localizer-arm mode when the autopilot detects localizer mode button has been depressed and the localizer flag is valid.
- C. The autopilot shall display LocArm mode when (LocModeSel = True) .AND. (LocModeSel=stable).AND. (LocRecvr = True) .
- D.



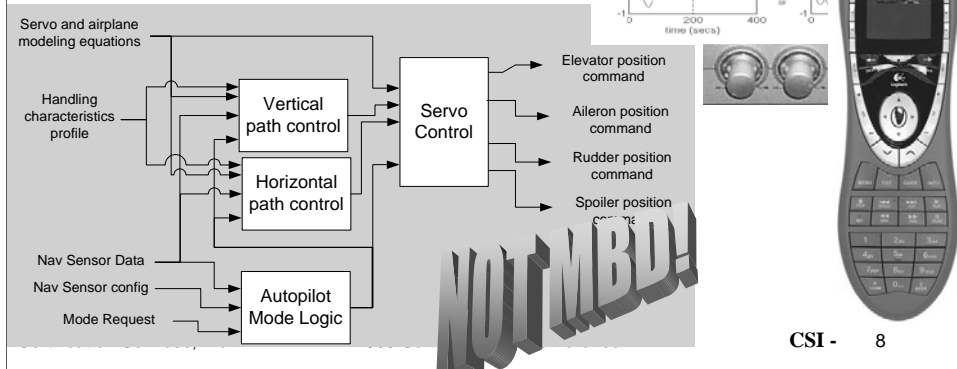
The same level of detail is available in all representations. They are merely different expressions of the same requirement.

Problem: abstraction gap



Indisputable high-level requirements development tools

- VAPS
- mdAutoLand



In these examples, desired behavior is captured from system-level specifications, and binary/source code is generated. In the mdAutoLand example, the tool may consist of N preconfigured autopilots that need only gain values for completeness. The tool examines all desired modes, response times, and servo characteristics, and produces a configuration that matches the input parameters with the proper gains. Tool qualification thus implies evaluation of autopilot correctness and of code generation.

Assume that the autopilot has five baseline configurations (wing leveler, heading/altitude controller, and so on). All high-level requirements for the tool become derived high-level requirements. These derived requirements must, by definition, be safety-validated at the system level. Development of code for each of the five autopilot configurations will follow a typical DO-178B development. Evaluation can show that the code is correct. Tool qualification will confirm that the proper gains are computed and instantiated within an autopilot.

A qualified tool “guarantees” that the correct autopilot template is selected, that the proper gains are set, and that the executable object code is, to an acceptable level of confidence, correct.

While such a tool does not exist for avionics, it is available in the world of audio-visual entertainment, albeit in non-qualified form. At least one company produces a remote control with capabilities similar to the “generic reprogrammer” described above. You, as the owner of the remote, specify the name and model of each audio/video component that you wish to control. When connected to the Internet, the remote is programmed with the (multiple) infrared protocols appropriate for your components.

MBD tool credit and DO-178B

- 178B – refinement model (rewriting errors)
- Abstraction gaps and qualification approach
 - (HLR to LLR) vs (.src to .obj)
 - HLR to .obj
- Out-of-the-box vs whole-table approach vs objective-by-objective approach
- MBD composed of a set of tools – boils down to tool qualification

DO-178B is based on successive refinement of abstract representations of behavior into concrete executable representations of that behavior. Refinement generally proceeds by rewriting one level of behavioral specification (for example, requirements) into more detailed levels of specification. Indeed, object code is merely a specification for machine execution. The objectives of Tables A3 through A7 directly address errors in the rewriting process by increasing confidence that such errors have been eliminated. We can thus reason about the behavioral correctness of our execution model in steps.

DO-178B Tables A3 through A5 are targeted for human review of high-level requirements, low-level requirements, architecture and source code that were developed using the refinement process.

Note that the object code does not have a similar review structure. Instead, the successful execution of a complete set of normal-, robustness-, and target-specific test cases based on requirements is considered adequate. One tool vendor offered its MBD as a straightforward extension of this philosophy - .exe to .obj. to .asy to .src to .model. Therefore, the modeling tool can be seen as just a language compiler at a higher level than normal.

From a DO-178B viewpoint, MBD is not a single tool. MBD is a combination of tools that must be analyzed. Analysis must illuminate the relationships of DO-178B objectives to tool properties and confirm that those relationships are acceptable.

Tool credit

- Goal of objectives: to reduce in-service errors by operating on error classes
- Approach using Annex A
 - What objectives are being made obsolete?
 - How are associated errors mitigated?
- Approach using alternate means (safety case)
 - How is correct behavior assured?
 - How are emergent errors mitigated?
 - What is the role of properties?

Criteria for Annex A objectives

- Objective satisfied by tool – partially/fully
- Objective requires manual conventional effort – partially/fully
- Assumptions needed for satisfaction
 - User (restricted constructs, use of tool, etc.)
 - Development environment (host, compilers, options, etc.)
 - Target environment (i386, PPC, I/O, etc.)
 - Execution environment (RTOS, scheduling, data formats, additional procedures, etc.)

Compliance data

- Credit analysis document (178B section 11.32)
- Organize by table or by objective
- Similar to AC 20-148 (RSC)
 - Credit (full/partial)
 - Assumptions
 - Additional activities and associated evidence

CSI

Example form

- Free template: cathy.vierthaler@certification.com

	Verification of outputs of software coding and integration processes	Credit assessment	Assumptions	Additional user activities	Credit justification
5-1	Source code complies with low-level requirements. 6.3.4a				
5-2	Source code complies with software architecture. 6.3.4b				
5-3	Source code is verifiable. 6.3.4c				
5-4	Source code conforms to standards. 6.3.4d				
5-5	Source code is traceable to low-level requirements. 6.3.4e				
5-6	Source code is accurate and consistent. 6.3.4f				
5-7	Output of software integration process is complete and correct. 6.3.5				

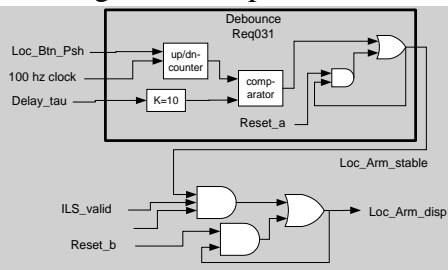
Certification Services, Inc.

2005 Software/CEH Conference

CSI - 13

Example: Annex A, Table A5 Source Code

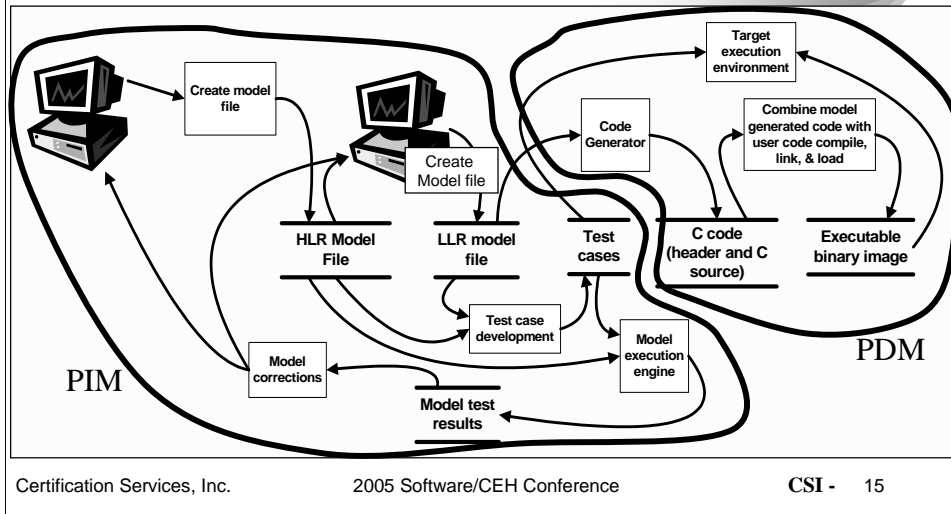
- Traceability/compliance LLR
 - Goodness
 - Standards compliance
 - Architecture compliance
 - Integration complete
- Criteria
 - Satisfied by tool (partial/full)
 - Satisfied conventionally (partial/full)
 - What can go wrong
 - Assumptions
 - Additional activities



This slide relates to the application source code produced by the tool.

Assume that our tool has a set of assembly macros for each block on the screen. Assume further that only the parameters, inputs, and outputs are instantiated relative to the application.

The great execution debate



Can we take credit for DO-178B objectives satisfied by the PIM, without repeating activities on the PDM? The question is related to issues such as structural coverage and normal- and robustness testing. In some avionics domains -- notably, ones whose failure conditions have no adverse consequences on safety -- studies have shown that most errors are found in the PIM and that few errors occur in the field due to PDM issues. While the data looks promising, the difficulty is providing the appropriate levels of confidence in the data provided by unqualified tool vendors. Although more complete data is provided by qualified tool vendors, regulators are reluctant to grant sweeping credit. If all activities are repeated in the PDM environment, there is rarely a certification issue. Repetition does, however, affect cost and schedule. The tools needed to extract data from the PDM are inevitably specific to the target execution platform.

Call for conviction

- Guilty of applying superior design approaches
- Use of circumstantial evidence vs. good forensics