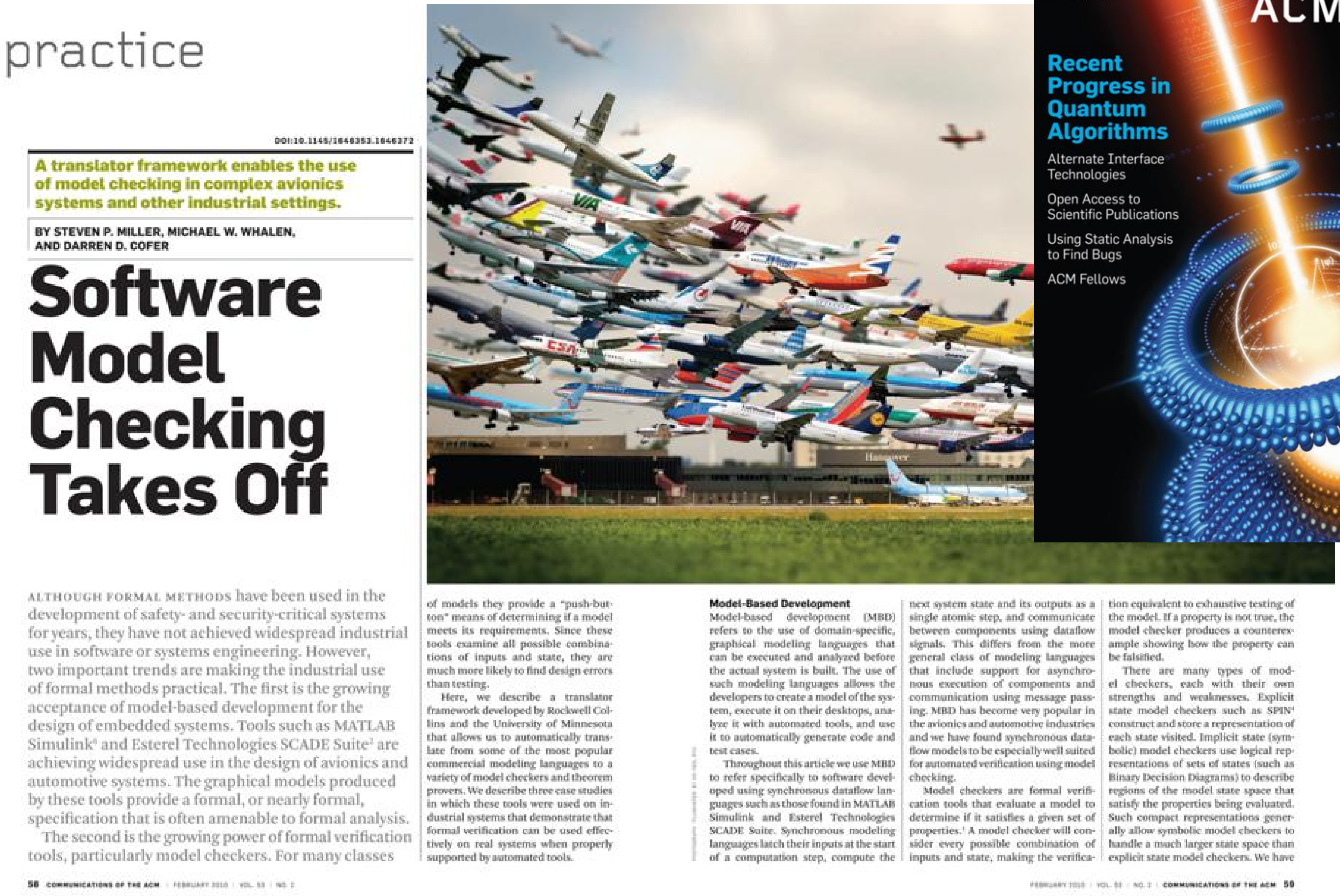
DO-178C

High Confidence Software & Systems Conference
8 May 2012

Dr. Darren Cofer
ddcofer@rockwellcollins.com



practice

DOI:10.1145/1646353.1646372

A translator framework enables the use of model checking in complex avionics systems and other industrial settings.

BY STEVEN P. MILLER, MICHAEL W. WHALEN, AND DARREN D. COFER

Software Model Checking Takes Off

ALTHOUGH FORMAL METHODS have been used in the development of safety- and security-critical systems for years, they have not achieved widespread industrial use in software or systems engineering. However, two important trends are making the industrial use of formal methods practical. The first is the growing acceptance of model-based development for the design of embedded systems. Tools such as MATLAB Simulink[6] and Esterel Technologies SCADE Suite[2] are achieving widespread use in the design of avionics and automotive systems. The graphical models produced by these tools provide a formal, or nearly formal, specification that is often amenable to formal analysis.

The second is the growing power of formal verification tools, particularly model checkers. For many classes of models they provide a "push-button" means of determining if a model meets its requirements. Since these tools examine all possible combinations of inputs and state, they are much more likely to find design errors than testing.

Here, we describe a translator framework developed by Rockwell Collins and the University of Minnesota that allows us to automatically translate from some of the most popular commercial modeling languages to a variety of model checkers and theorem provers. We describe three case studies in which these tools were used on industrial systems that demonstrate that formal verification can be used effectively on real systems when properly supported by automated tools.

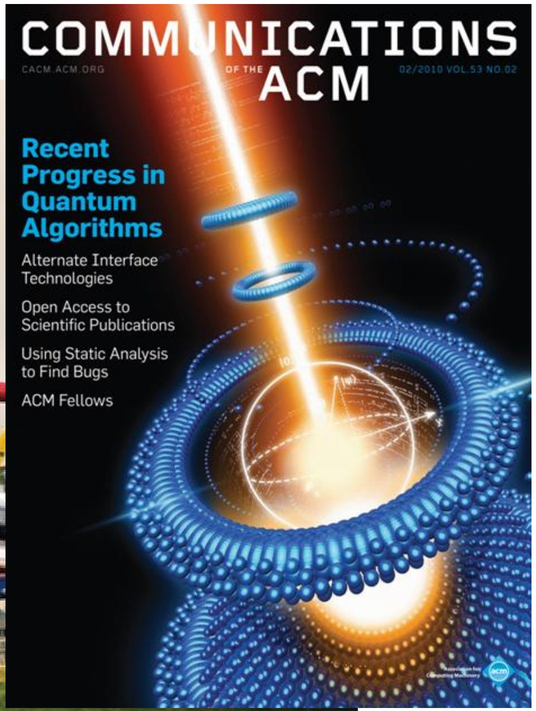
Model-Based Development

Model-based development (MBD) refers to the use of domain-specific, graphical modeling languages that can be executed and analyzed before the actual system is built. The use of such modeling languages allows the developers to create a model of the system, execute it on their desktops, analyze it with automated tools, and use it to automatically generate code and test cases.

Throughout this article we use MBD to refer specifically to software developed using synchronous dataflow languages such as those found in MATLAB Simulink and Esterel Technologies SCADE Suite. Synchronous modeling languages latch their inputs at the start of a computation step, compute the next system state and its outputs as a single atomic step, and communicate between components using dataflow signals. This differs from the more general class of modeling languages that include support for asynchronous execution of components and communication using message passing. MBD has become very popular in the avionics and automotive industries and we have found synchronous dataflow models to be especially well suited for automated verification using model checking.

Model checkers are formal verification tools that evaluate a model to determine if it satisfies a given set of properties.[1] A model checker will consider every possible combination of inputs and state, making the verification equivalent to exhaustive testing of the model. If a property is not true, the model checker produces a counterexample showing how the property can be falsified.

There are many types of model checkers, each with their own strengths and weaknesses. Explicit state model checkers such as SPIN[4] construct and store a representation of each state visited. Implicit state (symbolic) model checkers use logical representations of sets of states (such as Binary Decision Diagrams) to describe regions of the model state space that satisfy the properties being evaluated. Such compact representations generally allow symbolic model checkers to handle a much larger state space than explicit state model checkers. We have


COMMUNICATIONS OF THE ACM
CACM.ACM.ORG
02/2010 VOL.53 NO.02
Recent Progress in Quantum Algorithms
Alternate Interface Technologies
Open Access to Scientific Publications
Using Static Analysis to Find Bugs
ACM Fellows

Communications of the ACM, February 2010 (Vol. 53, No. 2)

Outline

- DO-178B
- SC-205
- DO-178C

Rockwell Collins

Definition

- Certification: Legal recognition by the regulatory authority that a product, service, organization or person complies with the requirements
  - Type certification: design complies with standards to demonstrate adequate safety
  - Product conforms to certified type design
  - Certificate issued to document conformance
- Example
  - We used verification tool X to accomplish these objectives.
  - These are the reasons why we think the tool is acceptable.
  - We ran 1000 tests using the tool, and this is why we think these 1000 tests are sufficient.
  - And (almost incidentally) here are the test results.

Convincing the relevant Certification Authority that all required steps have been taken to ensure the safety/reliability/integrity of the system

DO-178B

- “Software Considerations in Airborne Systems”
  - Certification authorities agree that an applicant can use guidance as a means of compliance (but not the only means) with regulations governing aircraft certification
- Software is not actually certified, but certification of an aircraft does include the “software aspects” of certification.

History

- DO-178 (1982)
  - Conceptual, “best practices”
- DO-178A (1985)
  - Development & verification processes described
  - 3 software levels defined
- DO-178B (1992)
  - Objectives/activities/evidence
  - Technology neutral
  - 5 levels

Software Level
A
B
C
D
E

Overview of DO-178B

- Defines the Process for Software Development
  - Objective based – specifies what is to be achieved, not how
- Identifies Five Software Levels (DAL in other contexts)

  A: Catastrophic (everyone dies)

  B: Hazardous/Severe (serious injuries)

  C: Major (significant reduction in safety margins)

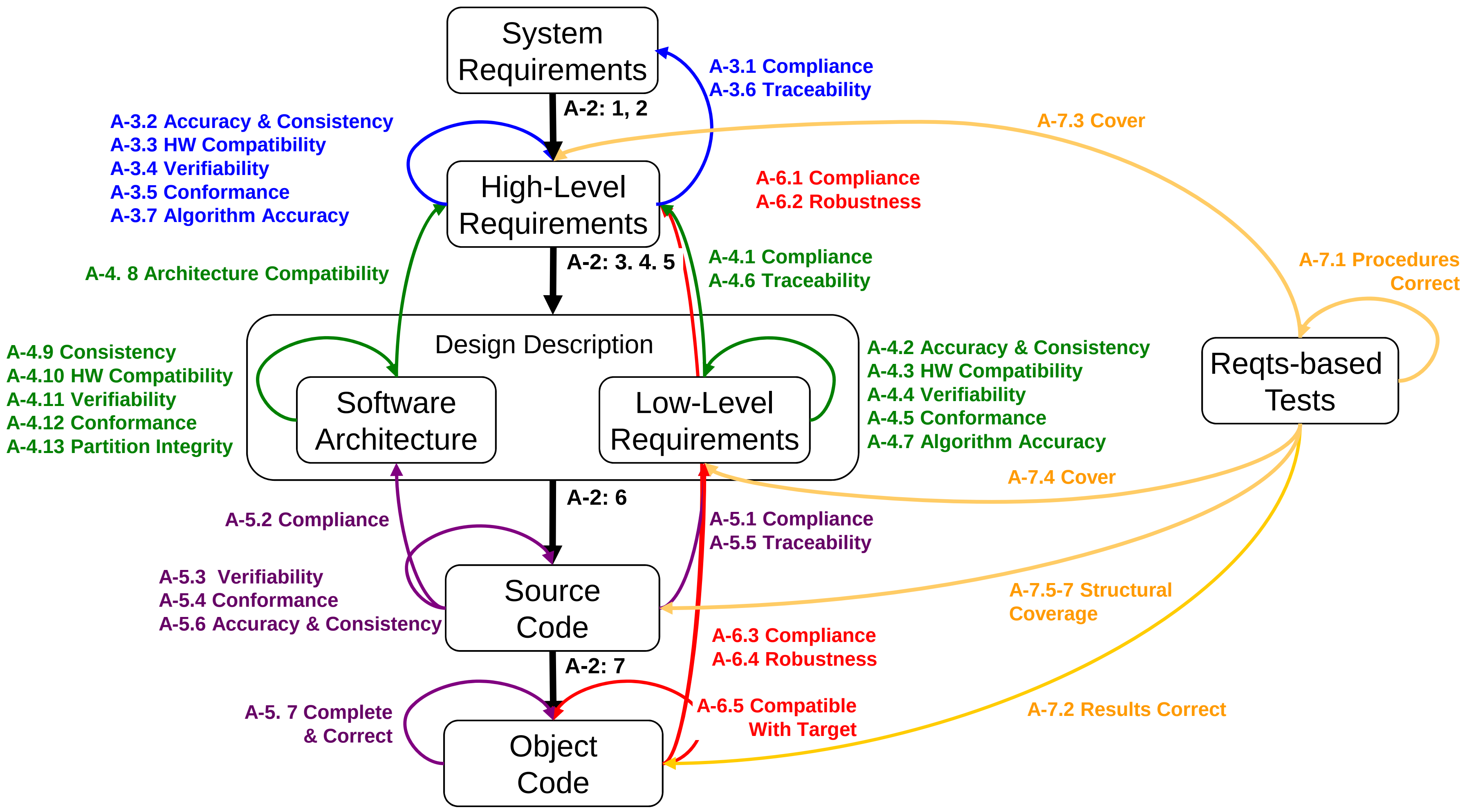
  D: Minor (annoyance to crew)

  E: No Effect (OK to use Windows)
- Higher level -> more objectives
  - But no explicit rationale for leveling
- Requires auditable evidence of specific processes
  - Software Planning
  - Software Development
  - Software Verification
  - Software Configuration Management
  - Software Quality Assurance
  - Certification Liaison
- Objectives Summarized in Annex A
  - Different objectives and requirements for each SW level

DO-178B overview

- Primarily a quality document, not safety
- Demonstrate that software implements requirements and nothing else
  - No surprises

DO-178B Verification Objectives



Verification in DO-178B

- Verification = review + analysis + test
- Testing of airborne software has two complementary objectives.
  - One objective is to demonstrate that the software satisfies its requirements.
  - The second objective is to demonstrate with a high degree of confidence that errors which could lead to unacceptable failure conditions, as determined by the system safety assessment process, have been removed.
- Formal methods can be used to meet these goals
  - Sometimes better

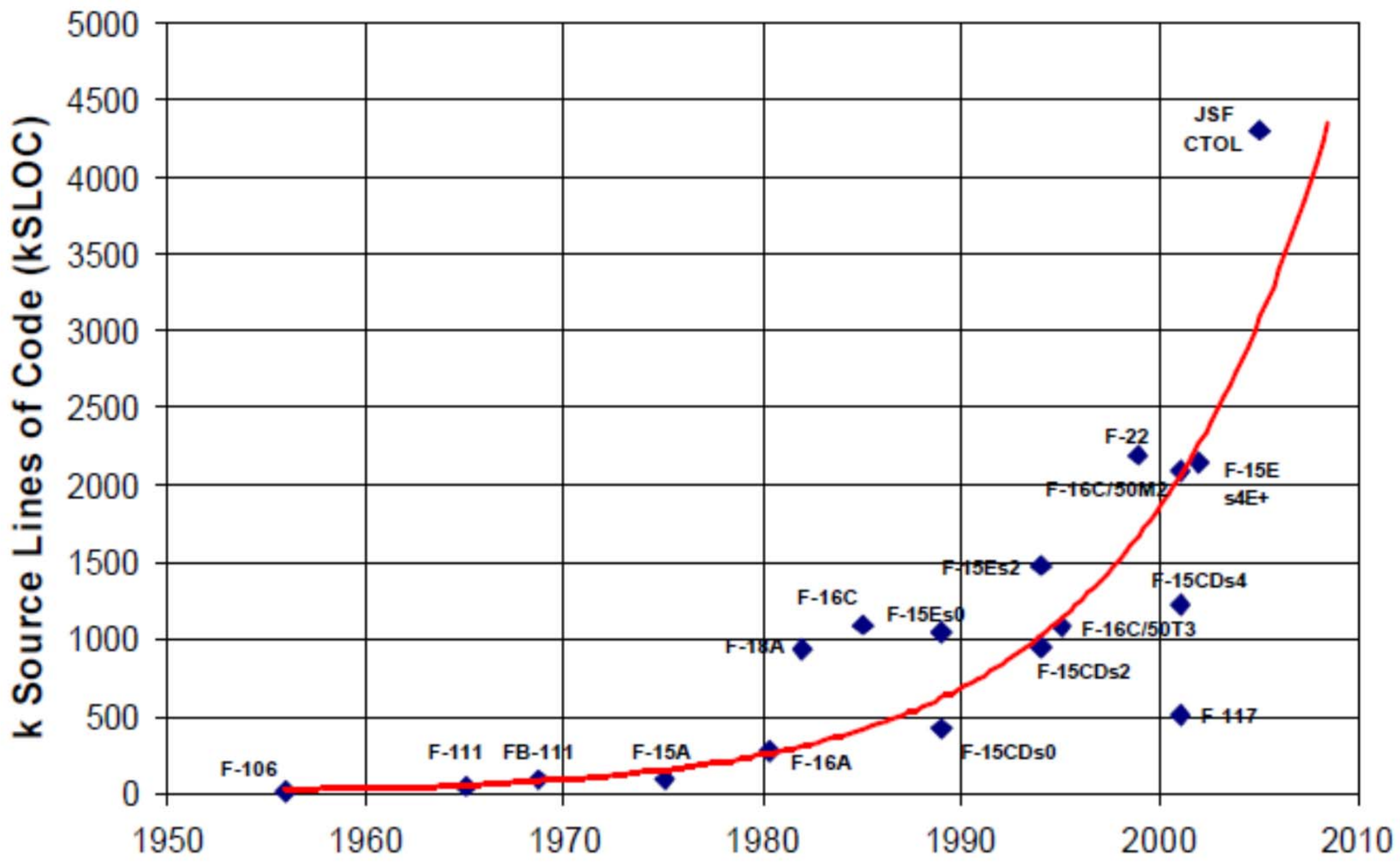
Verification principles

- Requirements-based testing
- Coverage metrics
- Traceability

Coverage metrics

- Defines four coverage metrics
- MC/DC
  - Purpose
  - Benefits
  - (preview for MBD & FM discussion...)

Software in military aircraft

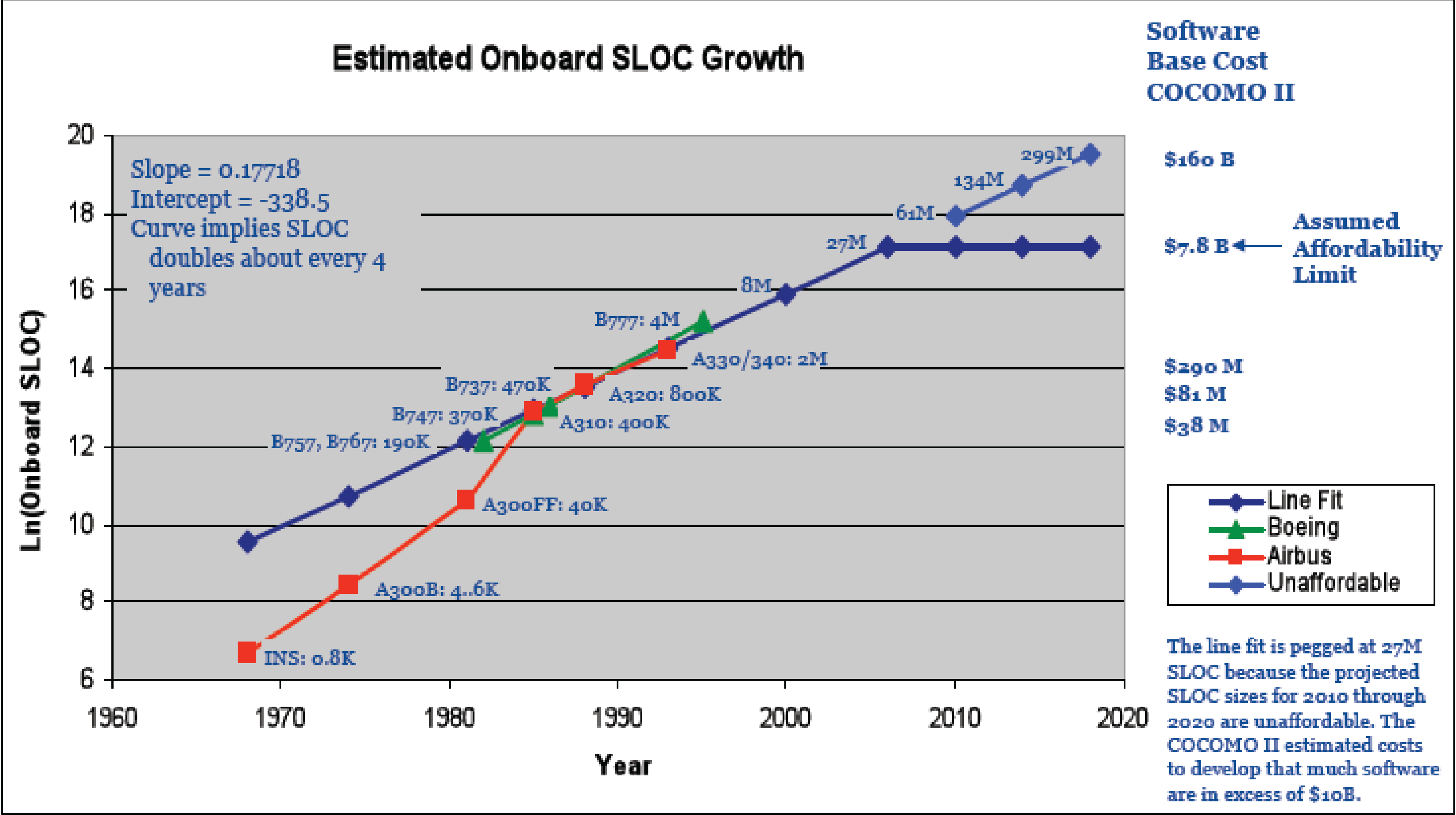


Source: D. Gary Van Oss (USAF), “Avionics Acquisition, Production, and Sustainment: Lessons Learned – The Hard Way,” NDIA Systems Engineering Conference, Oct 2002.

Rockwell Collins

Software in commercial aircraft

AVSI AEROSPACE VEHICLE SYSTEMS INSTITUTE



The line fit is pegged at 27M SLOC because the projected SLOC sizes for 2010 through 2020 are unaffordable. The COCOMO II estimated costs to develop that much software are in excess of $10B.

Airbus data source: J.P. Potocki De Montalk, Computer Software in Civil Aircraft, Sixth Annual Conference on Computer Assurance (COMPASS '91), Gaithersberg, MD, June 24-27, 1991.
Boeing data source: John J. Chilenski. 2009. Private email.

Why use formal methods with avionics SW?

- Reduce cost
  - Early detection/elimination of defects
- Increase confidence
  - Complete examination of models and requirements
- Satisfy certification objectives
  - DO-178C

Certification – the near future

- RTCA & EUROCAE form committee(s) to update DO-178B and develop DO-178C
  - Start: 2005
  - Finish: ~~2008~~ ~~2010~~ 2011
- Joint effort between US and Europe
  - RTCA : SC-205 : DO-178B : FAA ::
    EUROCAE : WG-71 : ED-12B : EASA
- Terms of Reference governing update
  - minimize changes to core document, yet…
  - update to accommodate 15+ years of SW experience
- Strategy: Address new technologies in “supplements”
  - OO, FM, MBD
  - Also tool qualification
- Other issues
  - Air/ground synergy (DO-278)
  - Rationale, consolidation, issues, errata (DO-248)

Some conditions

- Maintain the current objective-based approach for software assurance.
- Maintain the technology independent nature of the DO-178B objectives.
- Modifications to DO-178B/ED-12B should:
    1. Strive to minimize changes to the existing text (i.e., objectives, activities, software levels, and document structure)
    2. Consider the economic impact relative to system certification without compromising system safety
    3. Address clear errors or inconsistencies in DO-178B/ED-12B
    4. Fill any clear gaps in DO-178B/ED-12B
    5. Meet a documented need to a defined assurance benefit.
- A **supplement** is guidance used in conjunction with DO-178C/ED-12C that addresses the unique nature of a specific technology or a specific method.
    - A supplement adds, deletes or otherwise modifies: objectives, activities, explanatory text, and software life cycle data in DO-178C/ED-12C.

SC-205: The Game

- Players
  - Anyone can play!
  - But you must be interested enough show up
  - All players are equal (but some are more equal than others)
- Rules
  - Form teams (subgroups)
  - Propose text (Information Papers)
  - Write and resolve comments
  - Vote in plenary sessions
  - Achieve consensus: "I can live with it."
  - Watch out for illegal moves
    - "I'm just here to learn."
    - "Let me tell you about our product."
- Strategy
  - Don't raise the bar
  - Don't lower the bar
- Winning the Game
  - FAA issues advisory circular

DO-178C: Supplements

- Tool Qualification
  - More detailed, stand-alone document, complete with objectives
  - 3 tool criteria (1 = development, 3 = verification, 2 = something in between)
  - [criteria] + [SW level] => tool qual. level (1-5) => required objectives
- Model-Based Development
  - Model = HLR/LLR/SW architecture
  - New guidance for model execution/simulation
  - Model coverage requirements
- Object-Oriented Design
  - Additional requirements for unique aspects of OO software
- Formal Methods
  - Facilitate applicant/certifier communication (definitions, expected evidence)
  - Define new objectives/activities/documentation (abstractions, assumptions)
  - Avoid common errors (check false hypotheses)

Survey

- Which Technology Supplement was the most difficult to produce?
  - Tool Qualification
  - [✓ Model-Based Development]
  - Object Oriented Software
  - Formal Methods

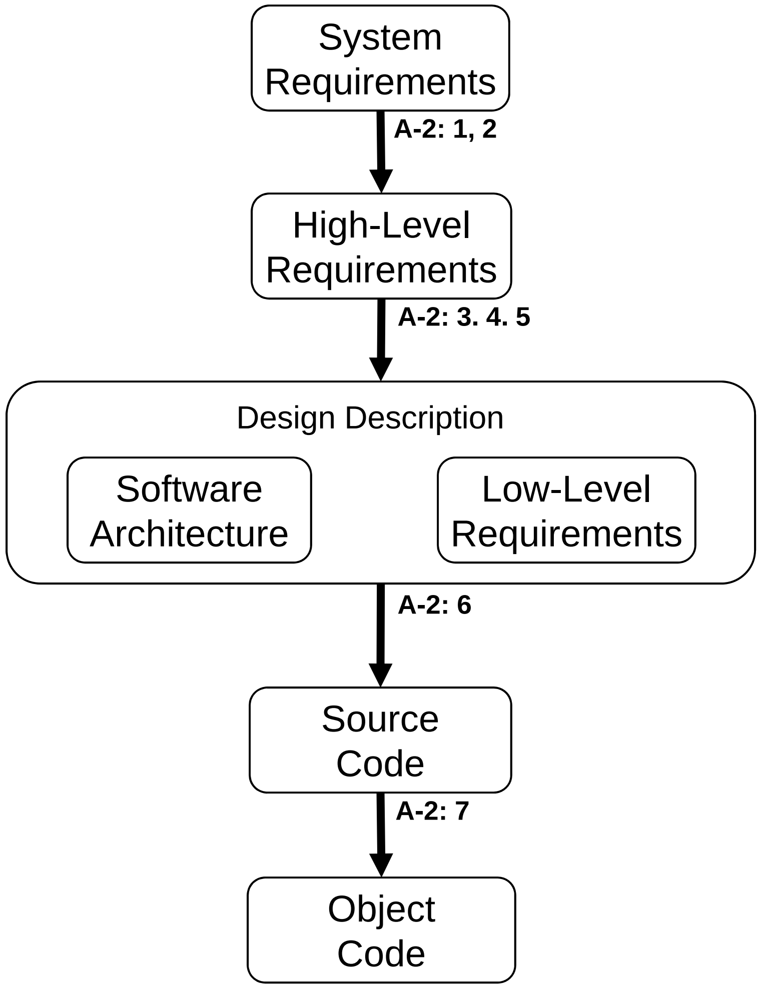
Formal Methods

- Objectives
  - No longer an “alternate method”
  - Provide basis for communication between applicants & certification authorities
  - Focus on verification (DO-178 section 6)
  - Partial use is OK
  - What should formal methods evidence look like?
  - Define new objectives/activities/documentation (abstractions, assumptions)
  - Avoid common errors (check false hypotheses)
- Key issues
  - Capturing assumptions used in analysis (constraints, assertions, environment…)
  - If analysis replaces unit testing, what constitutes “completeness” of analysis? (analog of MC/DC coverage metric)
  - How should formal analysis tools be <u>qualified</u>?
- Keep the bar high enough
  - Applicants with sufficient expertise

Section 6 – SOFTWARE VERIFICATION PROCESS

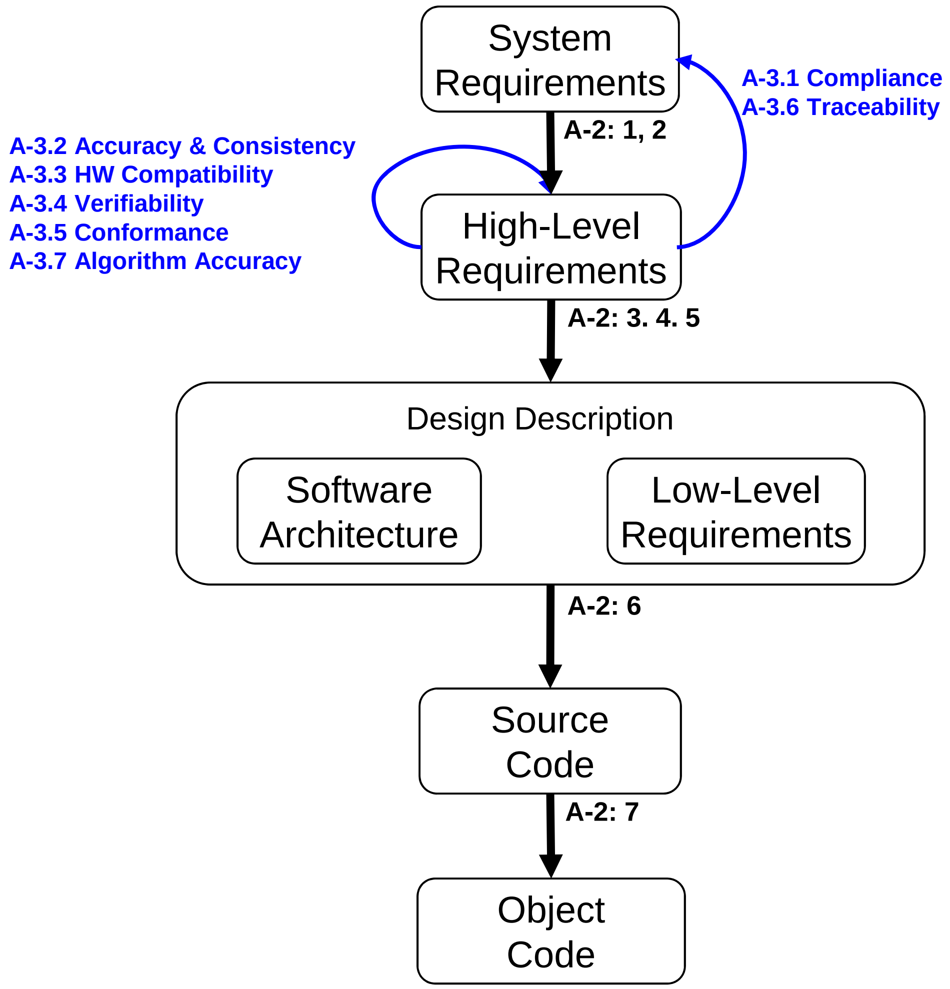
- Focus of FM guidance is on Verification Process
- General guidance:
  - All *formal notations* used must have unambiguous, mathematically defined syntax and semantics.
  - The soundness of each *formal analysis* method should be documented. A sound method never asserts that a *property* is true when it may not be true.
  - All assumptions related to the *formal analysis* should be described and justified (e.g. those associated with the target computer, or those about the data range limits).

Software Development Process



- **No new guidance for FM**
- **Nothing prevents use of formal specifications**
- **Benefit shows up in verification activities**

Verification of High-Level SW Requirements



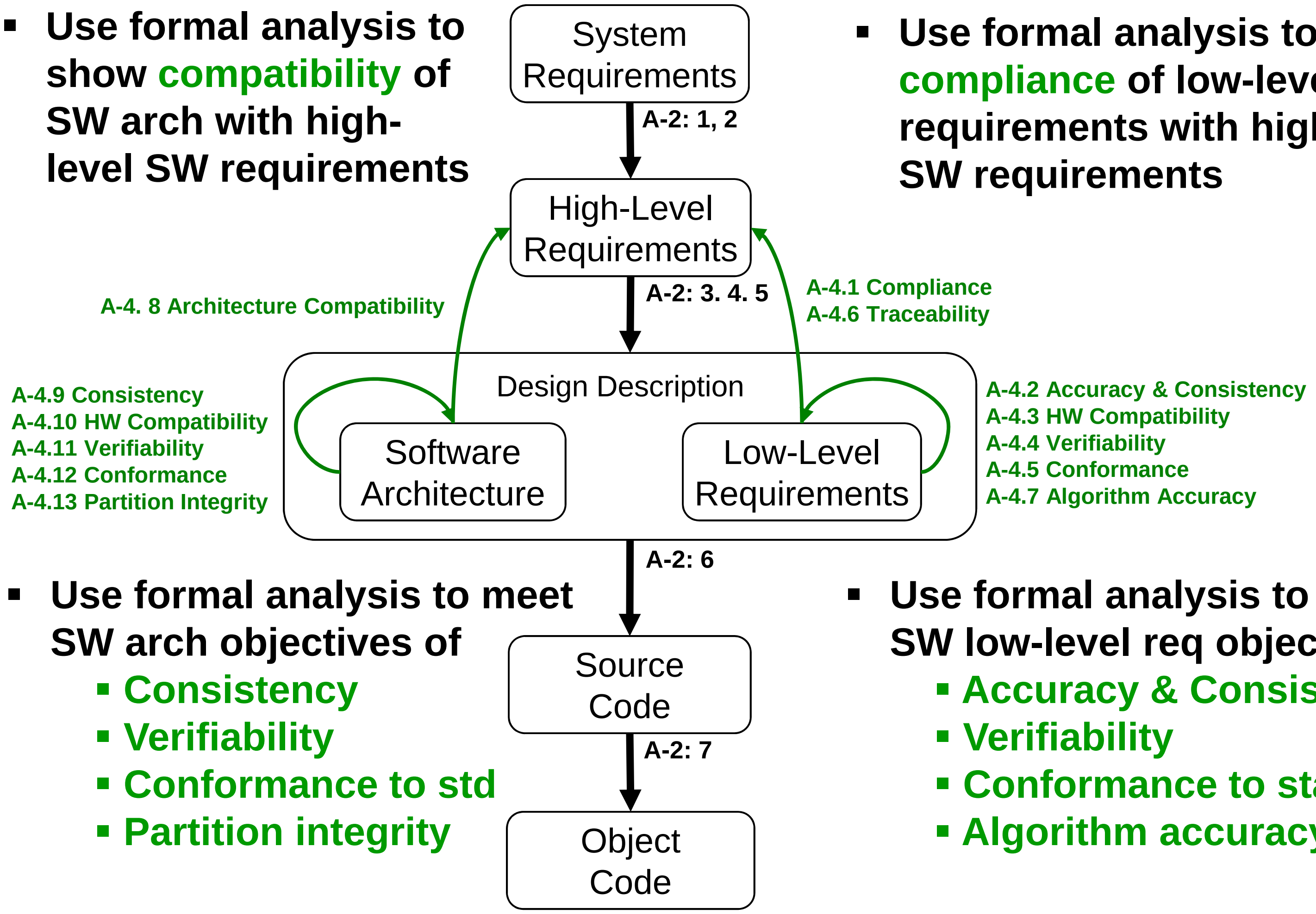
- **If system requirements and high-level SW requirements are formal, may be able to use formal analysis to show compliance**

- **May be able to use formal analysis to meet objectives of**
  - **Accuracy and consistency**
  - **Verifiability**
  - **Conformance to standards**
  - **Algorithms are accurate**

Verification of the System Design Description

- Use formal analysis to show compatibility of SW arch with high-level SW requirements

- Use formal analysis to show compliance of low-level SW requirements with high-level SW requirements

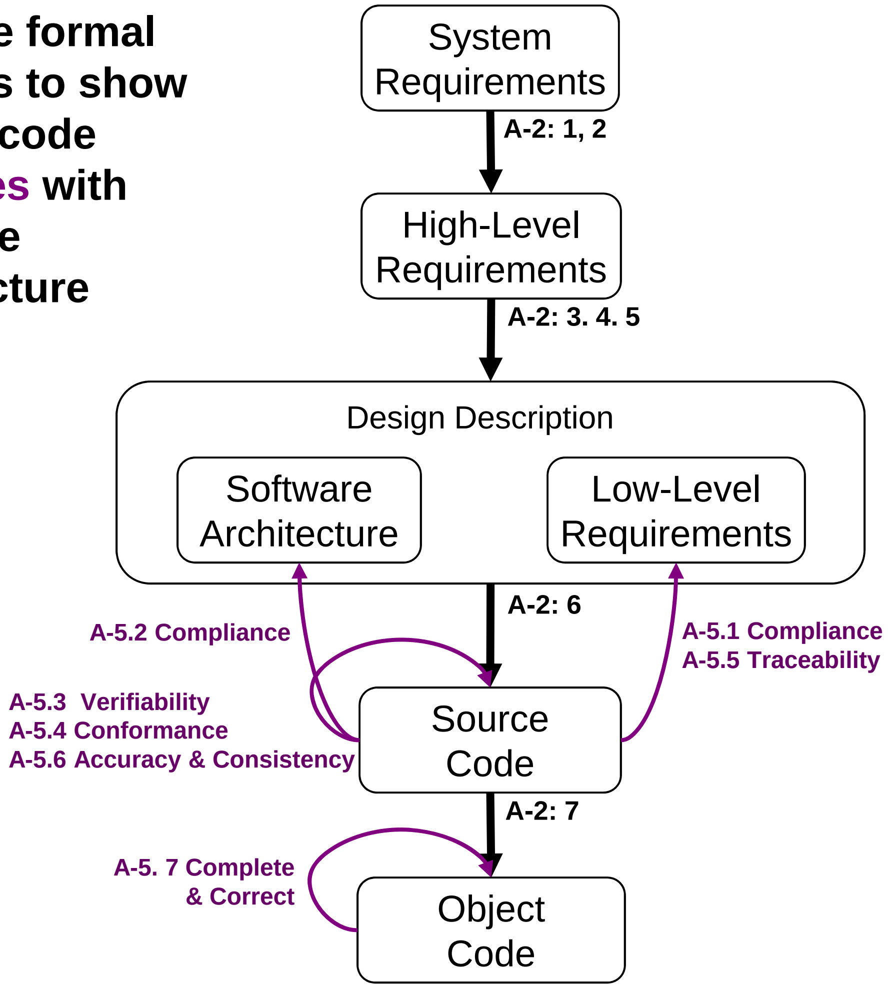


- Use formal analysis to meet SW arch objectives of
  - Consistency
  - Verifiability
  - Conformance to std
  - Partition integrity

- Use formal analysis to meet SW low-level req objectives of
  - Accuracy & Consistency
  - Verifiability
  - Conformance to standards
  - Algorithm accuracy

Verification of Source Code

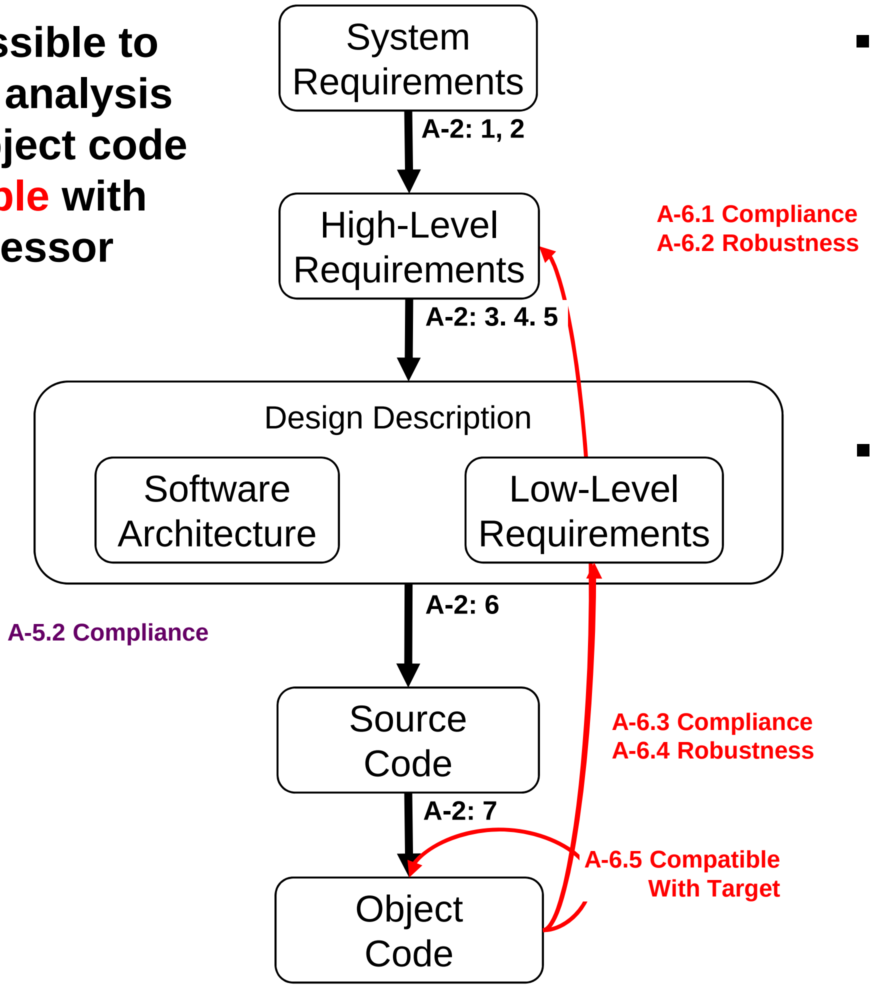
- May use formal analysis to show source code complies with software architecture



- May use formal analysis to show source code complies with low-level SW requirements
- Use formal analysis to meet source code objectives of
  - Verifiability
  - Conformance to standards
  - Accuracy & consistency
- Use formal analysis to meet show output of software integration process is
  - Complete & correct

Verification of Object Code

- May be possible to use formal analysis to show object code is compatible with target processor

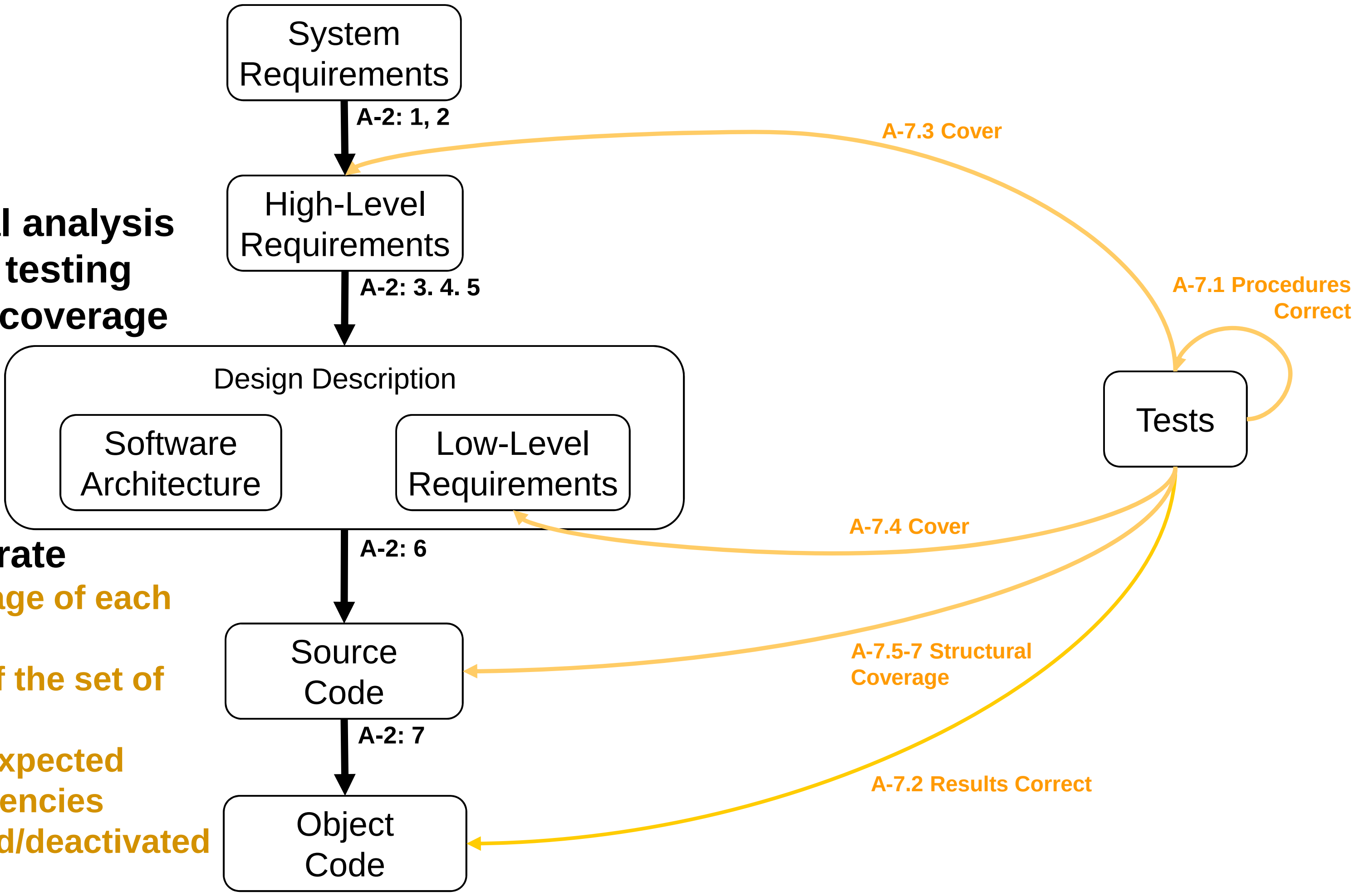


- Use formal analysis to show object code complies with low-level and high-level SW requirements

- Use formal analysis to show object code is robust with low-level and high-level requirements
  - Abstract interpretation?

Verification of Verification Results

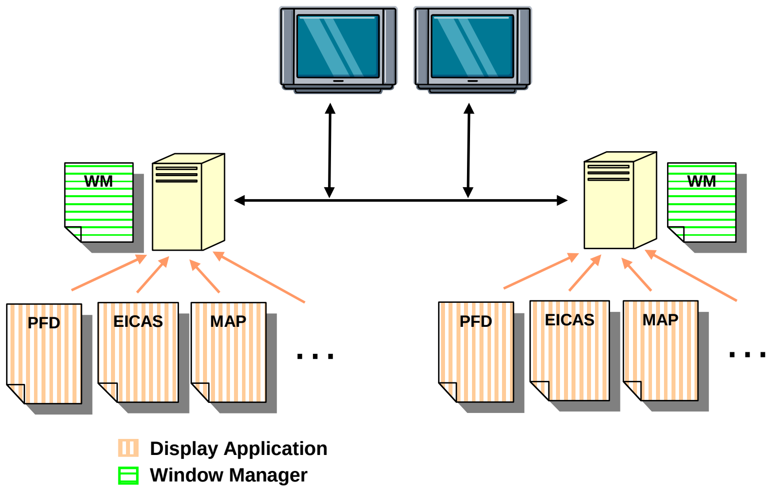
- **May use formal analysis to replace unit testing and structural coverage**

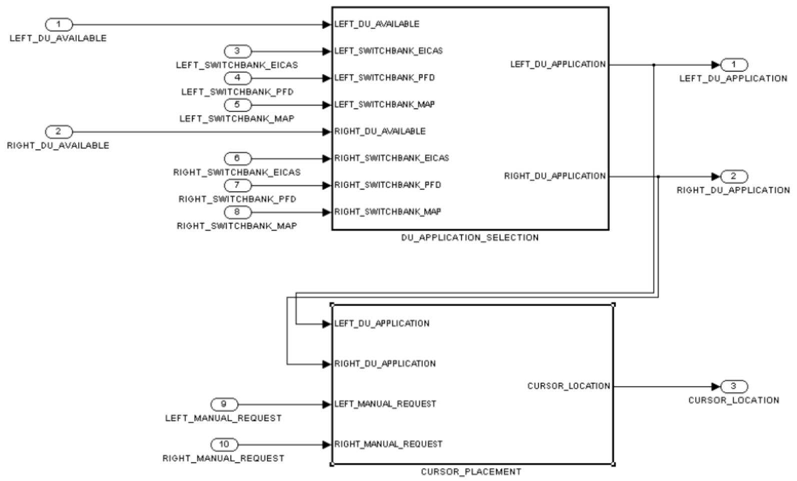


- **Must demonstrate**
- **Complete coverage of each requirement**
- **Completeness of the set of requirements**
- **Detection of unexpected dataflow dependencies**
- **Detection of dead/deactivated code**

Rockwell Collins

Example: Window Manager SW (cockpit display)





ADGS-2100 Adaptive Display & Guidance System

Subsystem	Simulink Diagrams	Simulink Blocks	State Space	Properties	Errors found
GG	2,831	10,669	9.8×10^{9}	43	56
PS	144	398	4.6×10^{23}	152	10
CM	139	1,009	1.2×10^{17}	169	10
DUF	879	2941	1.5×10^{37}	115	8
MFD	302	1,100	6.8×10^{31}	84	14
Totals	4295	16,117	n/a	563	98

Certification credit?

- Development process
  - HLRs are initially expressed as English “shall” statements that are subsequently formalized as CTL for analysis.
  - LLRs are software models developed using model-based design tools (Simulink and Stateflow).
  - The LLR models are analyzed using a model checker to verify whether or not they satisfy the HLRs.
  - Source code is automatically generated from the LLRs and tested in conformance with a conventional test-based process.
- What DO-178C objectives could be satisfied using the Formal Methods Supplement?
  - Just a sample…

FM for Certification

FM6.2 Software Verification Process Activities

a. **Formal notations**: Properties to be verified were specified in CTL. Formal definition of CTL may be found in [Em90] and [Hu04]. The models analyzed were specified in Simulink and Stateflow. These models were given formal definition through the translation process, which includes a formal syntax and translation rules for each model element.

b. **Soundness**: The BDD and SAT algorithms are known to be sound. Details of the BDD algorithm used for model checking and its soundness can be found in [Mc93]. Application of satisfiability solving to the model checking problem and its soundness is described in [BCCZ99],[CBRZ01].

c. **Assumptions**: All assumptions necessary for the analysis are justified.

FM for Certification

<u>FM6.3</u> Software Reviews and Analysis

i. **Requirement formalization correctness**: In this project, all requirements were captured and managed using the DOORS tool. For each requirement, the corresponding formalization was captured in DOORS with one or more CTL statements. Independent reviews were conducted to ensure that the CTL statements accurately described the original English-language requirement.

<u>FM6.3.1</u> Reviews and Analyses of the High-Level Requirements

d. **Verifiability of HLR**: The ability to express the high-level requirements for the system in CTL is a sufficient demonstration of verifiability in this example.

e. **Conformance to standards**: Requirements that do not conform to the standard for CTL syntax will be identified and rejected by the analysis tools. This feature of the tool would need to be qualified. Alternatively, conformance to CTL syntax can be easily checked by a manual review.

<u>FM6.3.2</u> Reviews and Analyses of the Low-Level Requirements

a. **Compliance with HLR**: Analysis by model checking demonstrated that low-level requirements (the system model) complied with high-level requirements. This feature of the tool would need to be qualified.

Conclusion

- Model checking can be used now to
  - Reduce the cost of avionics software through early detection and elimination of defects
  - Increase confidence in avionics software
- Model checking will soon be able to be used to
  - Satisfy certification objectives for avionics software (DO-178C)
- But…
  - There is still plenty of work to do to support larger and more complex systems