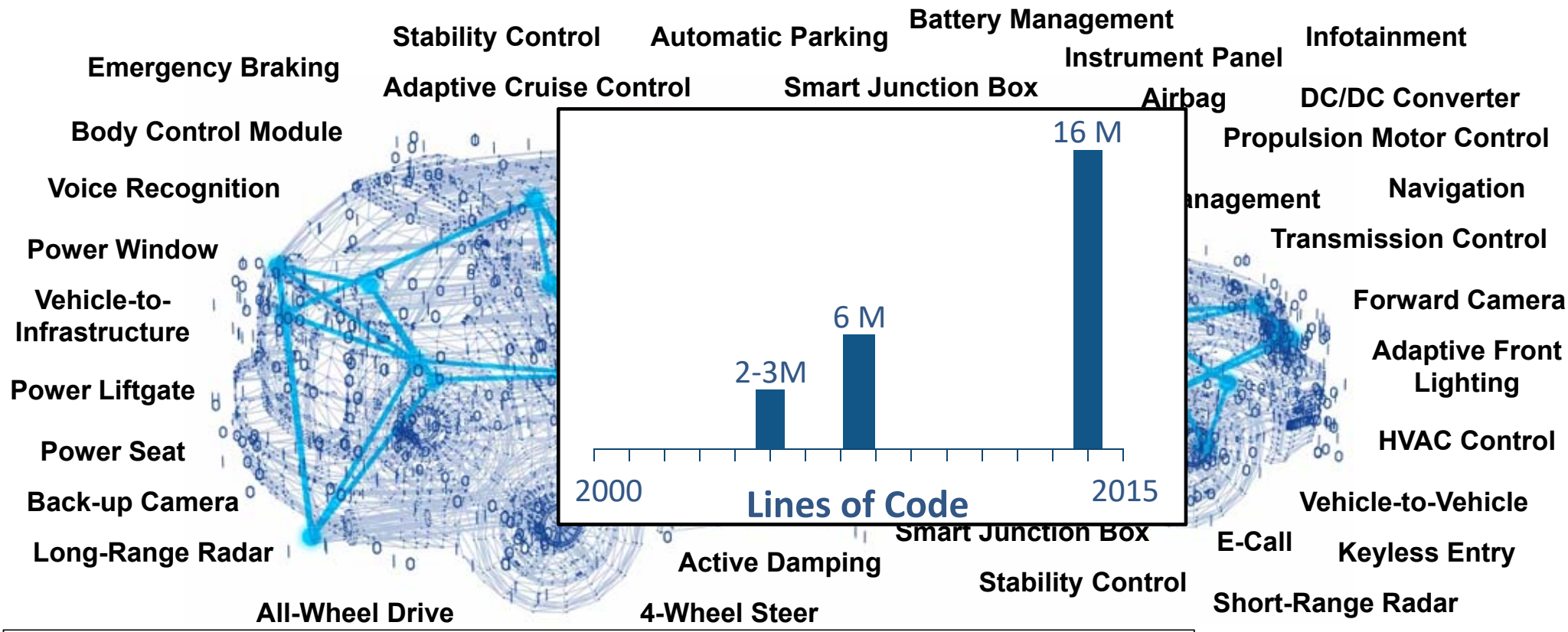
MATLAB EXPO 2018

Verification & Validation: Automating Best Practices to Improve Design Quality

Chuck Olosky

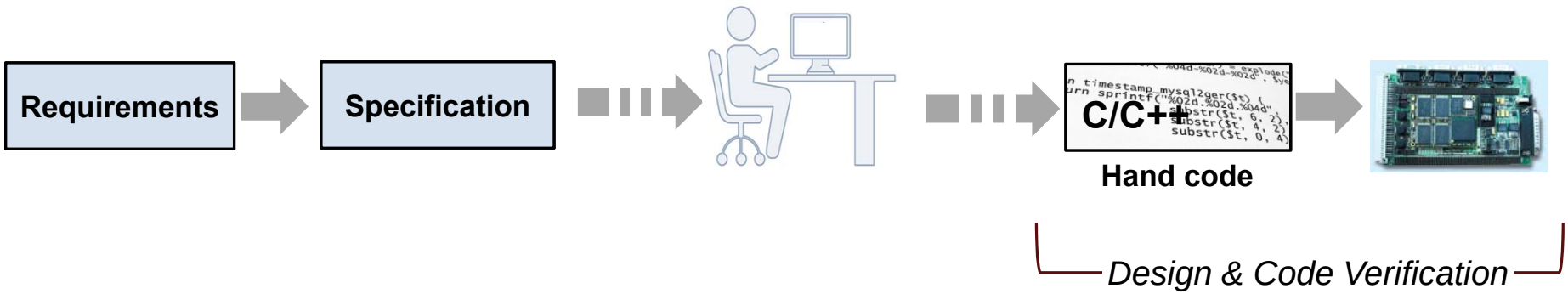
Growing Complexity of Embedded Systems



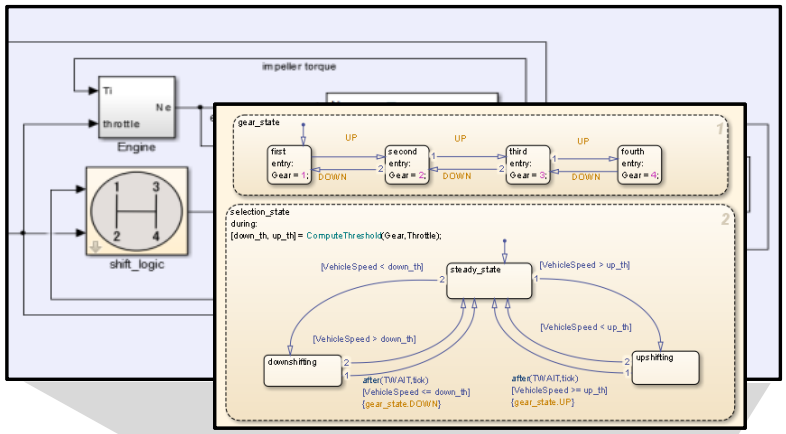
Siemens, "Ford Motor Company Case Study," Siemens PLM Software, 2014
McKendrick, J. "Cars become 'datacenters on wheels', carmakers become software companies," ZDJNet, 2013

Growing Complexity
Challenges the Traditional Development Process

- Find requirements defects later in the process
- Find specification issues later in the process
- Find design issues later in the process



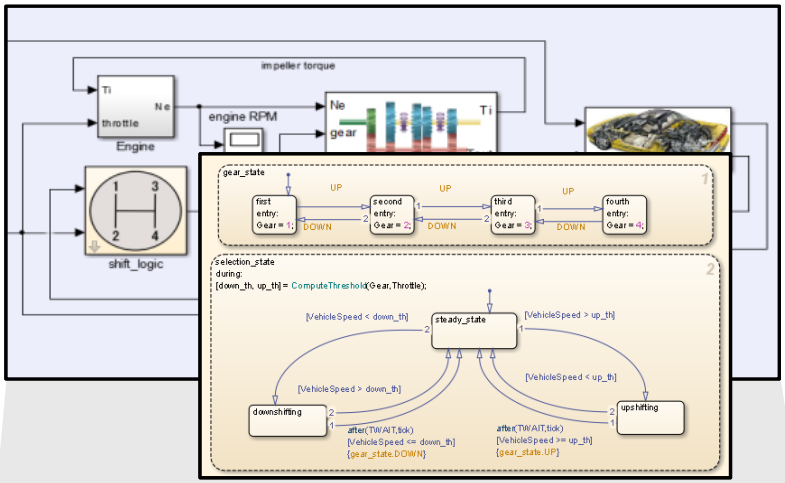
Using Simulink Models for Specification



- Find requirements defects earlier in the process
- Find specification issues earlier in the process
- Find design issues later in the process

Design & Code Verification

Complete Model Based Design



- Find requirements defects earlier in the process
- Find specification issues earlier in the process
- Find design issues earlier in the process

Simulink Models

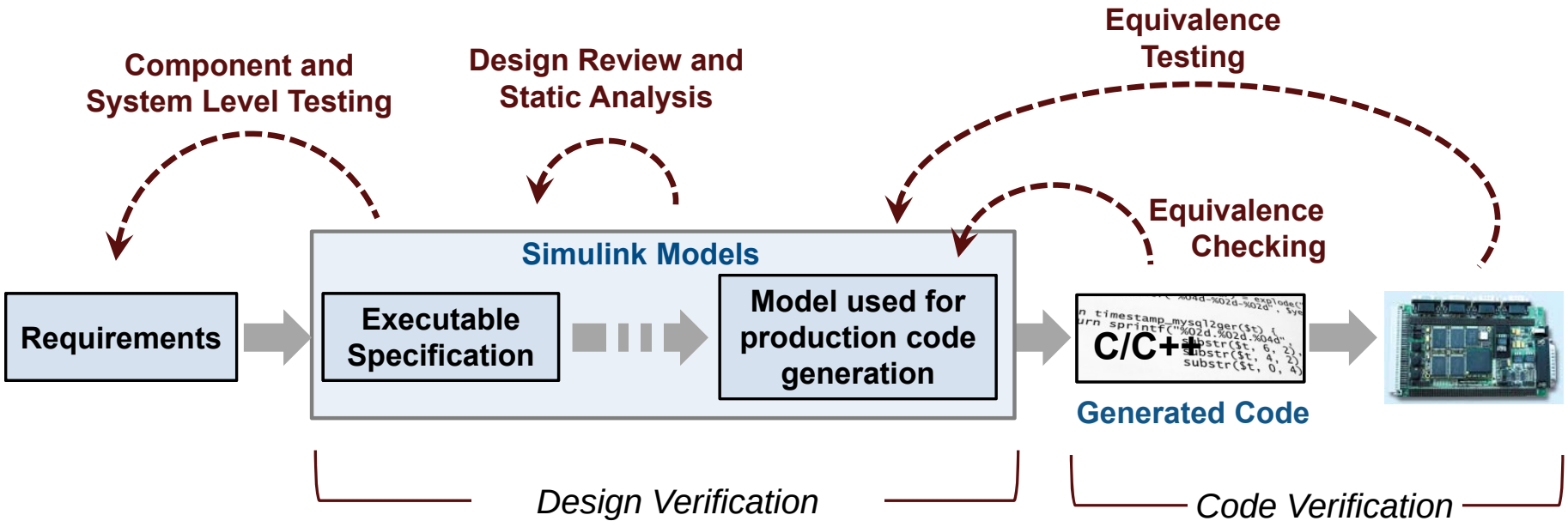
Requirements → **Executable Specification** → **Model used for production code generation** → C/C++ **Generated Code** → 

Design Verification

Code Verification

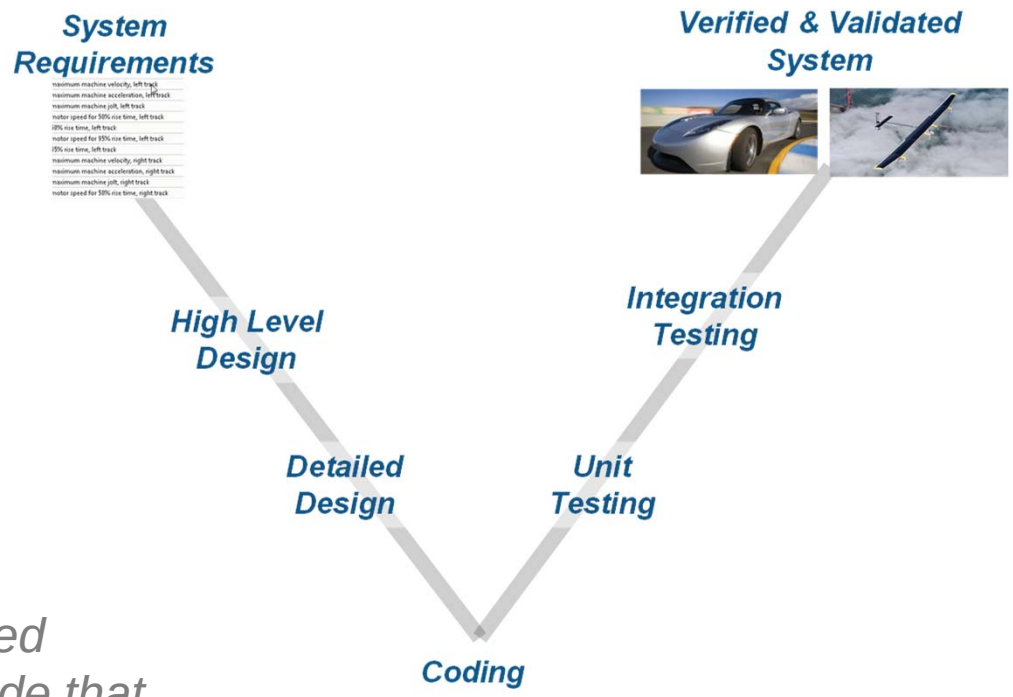
Code Generation

Model Based Design Verification Workflow



Key Takeaways

- Author, manage requirements in Simulink
- Early verification to find defects sooner
- Automate static and dynamic verification
- Workflow that conforms to safety standards



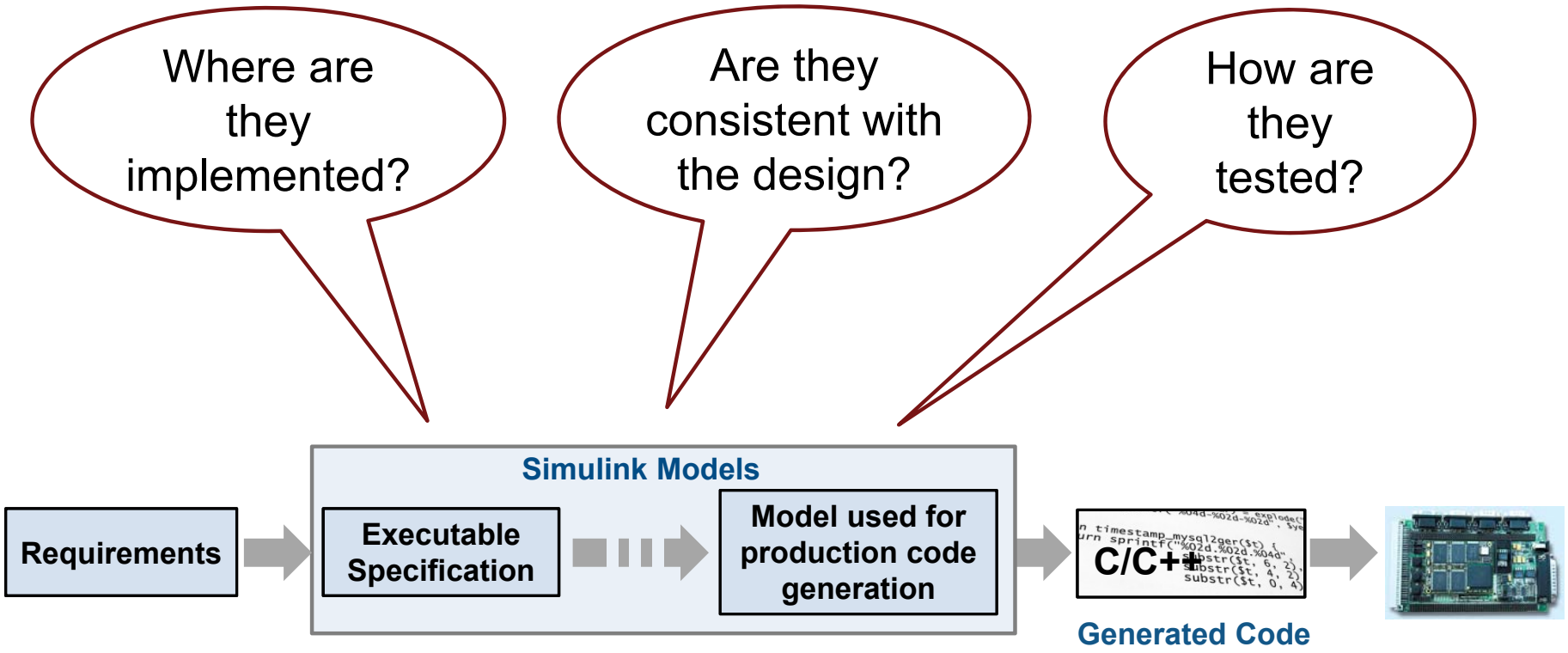
"Reduce costs and project risk through early verification, shorten time to market on a certified system, and deliver high-quality production code that was first-time right" Michael Schwarz, ITK Engineering

Why do 71% of Embedded Projects Fail?

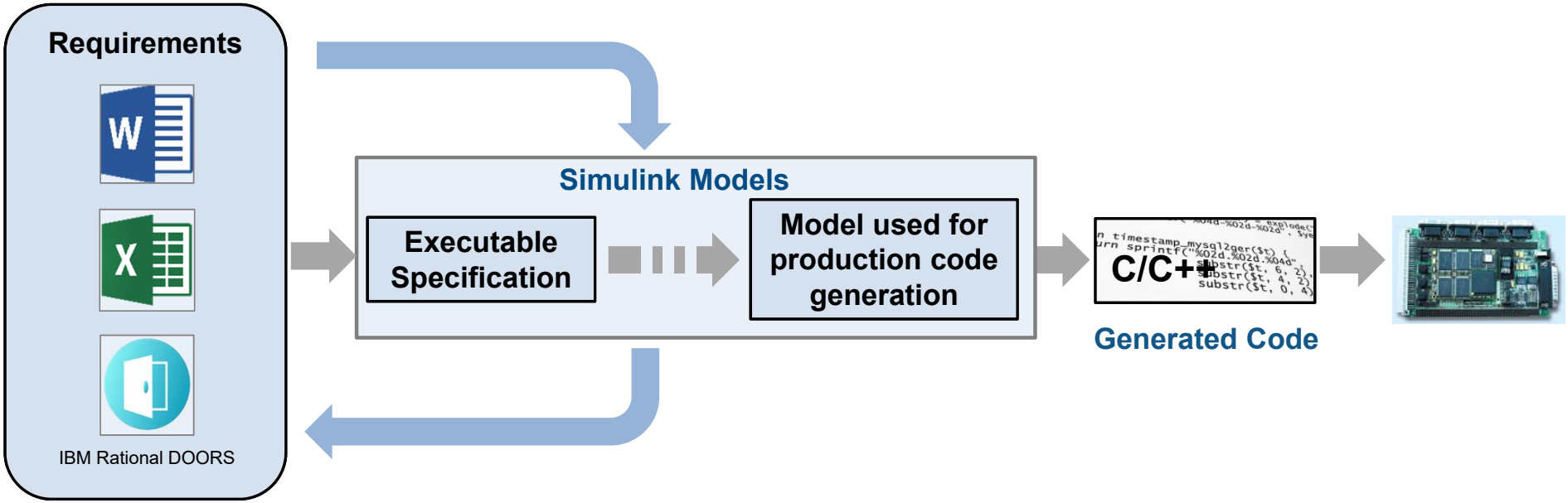
Poor Requirements Management

Sources: Christopher Lindquist, Fixing the Requirements Mess, CIO Magazine, Nov 2005

Challenges with Requirements



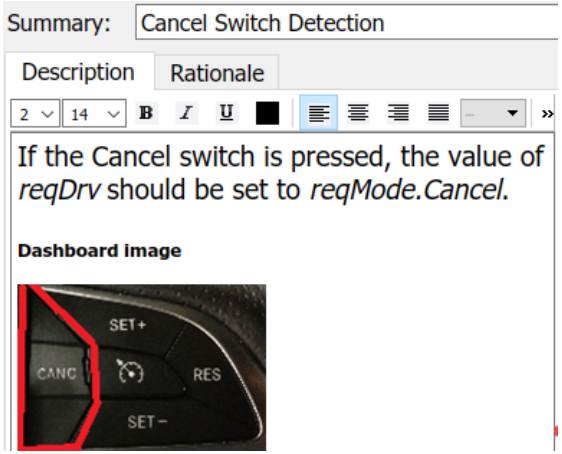
Gap Between Requirements and Design



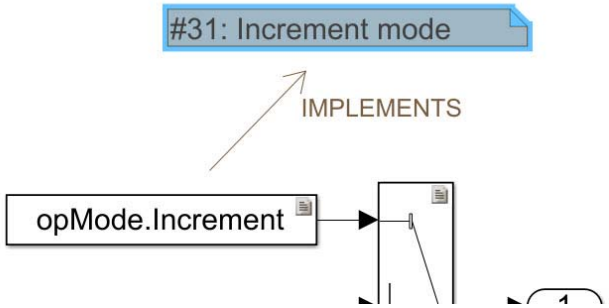
Simulink Requirements

R2017b

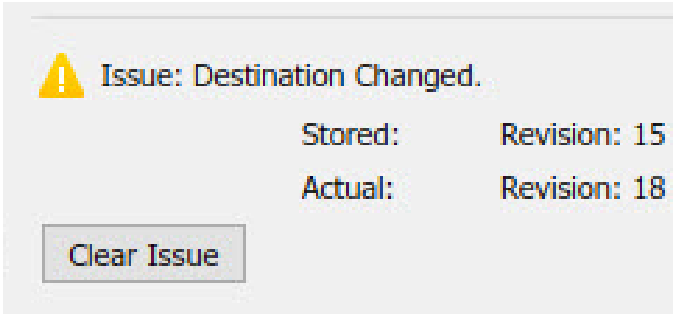
Author



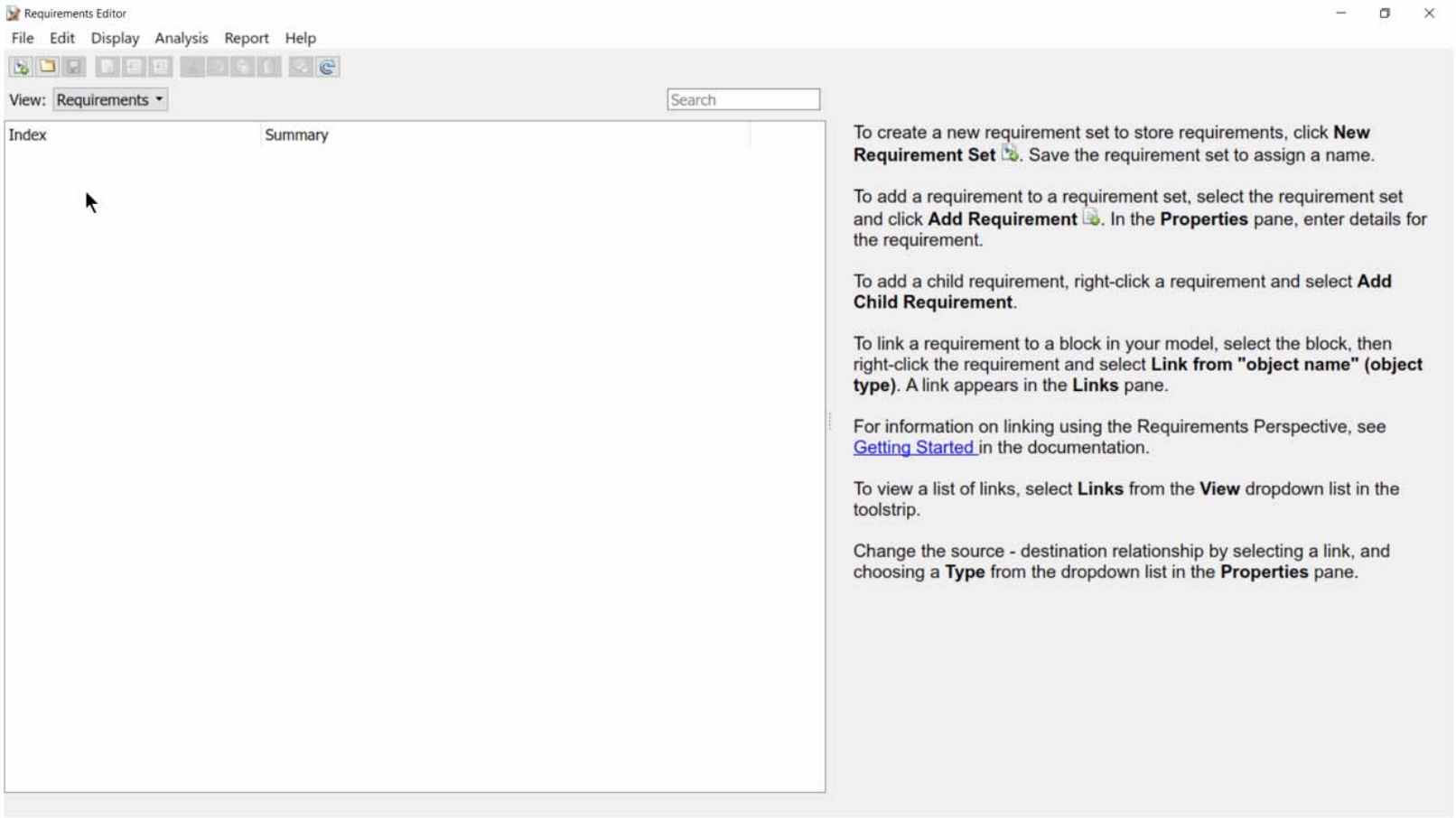
Link & Track



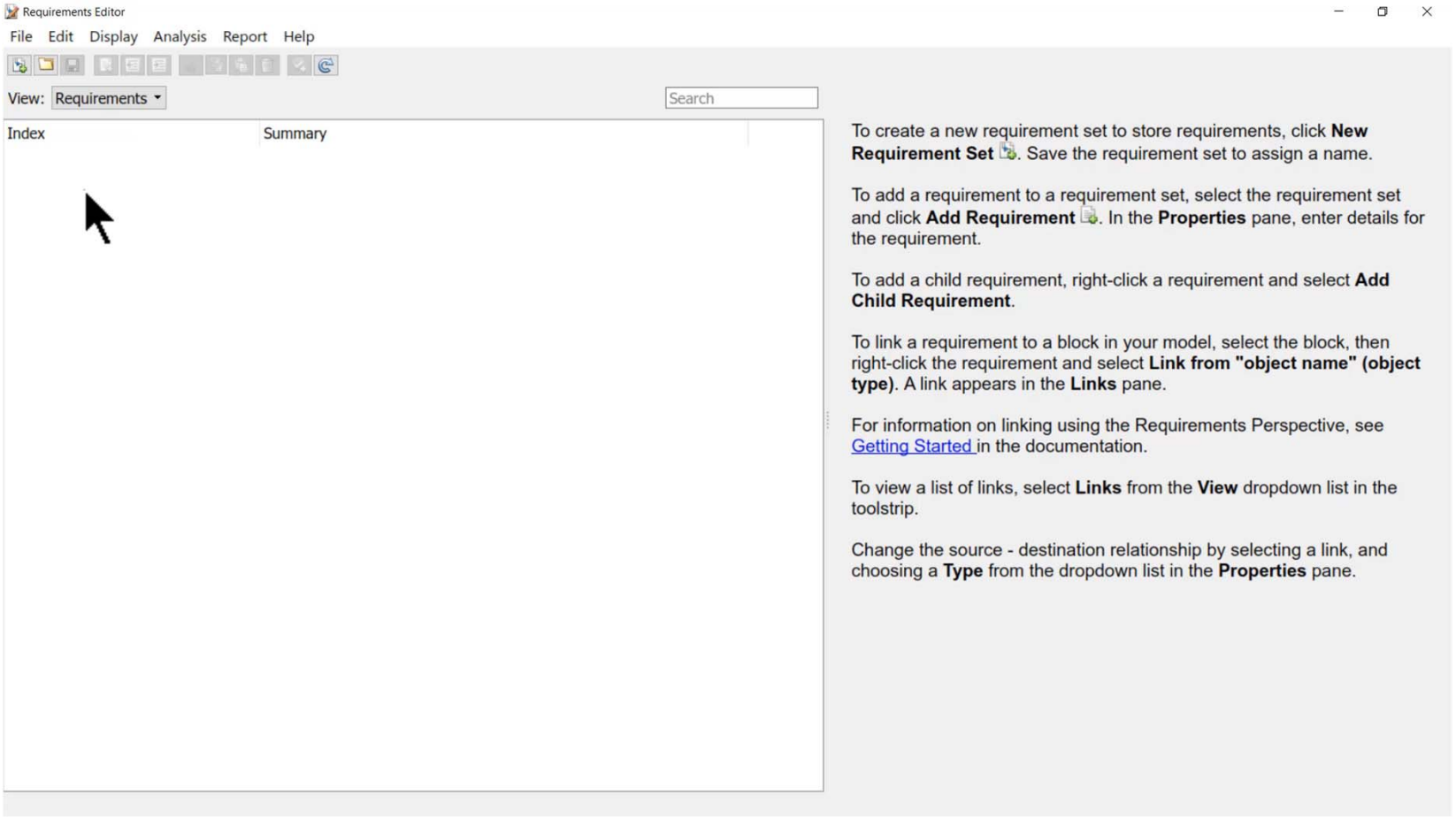
Manage Updates



Requirements Editor



Requirements Editor

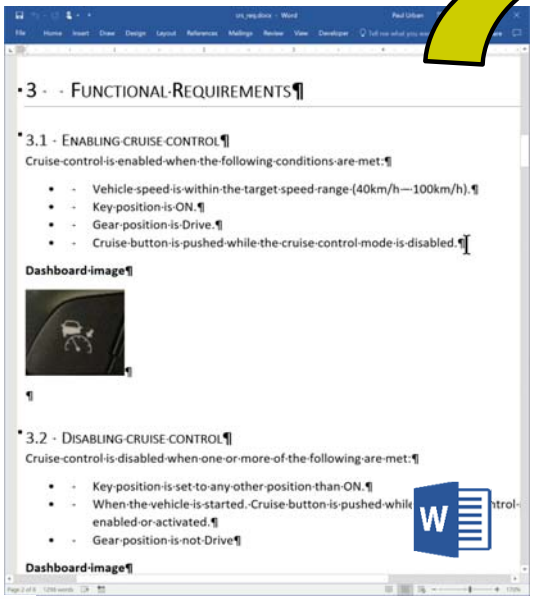


Import Requirements from External Sources

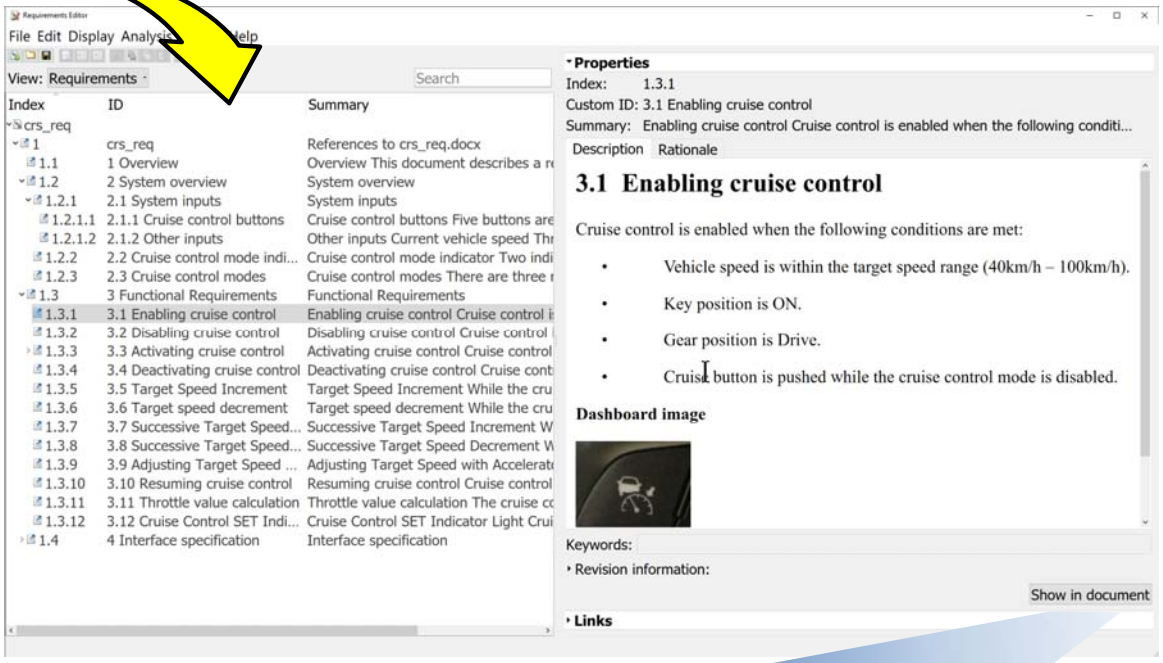
IBM Rational DOORS

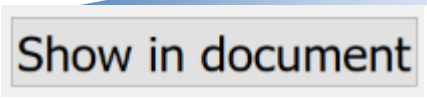
ReqIF R2018a

Import

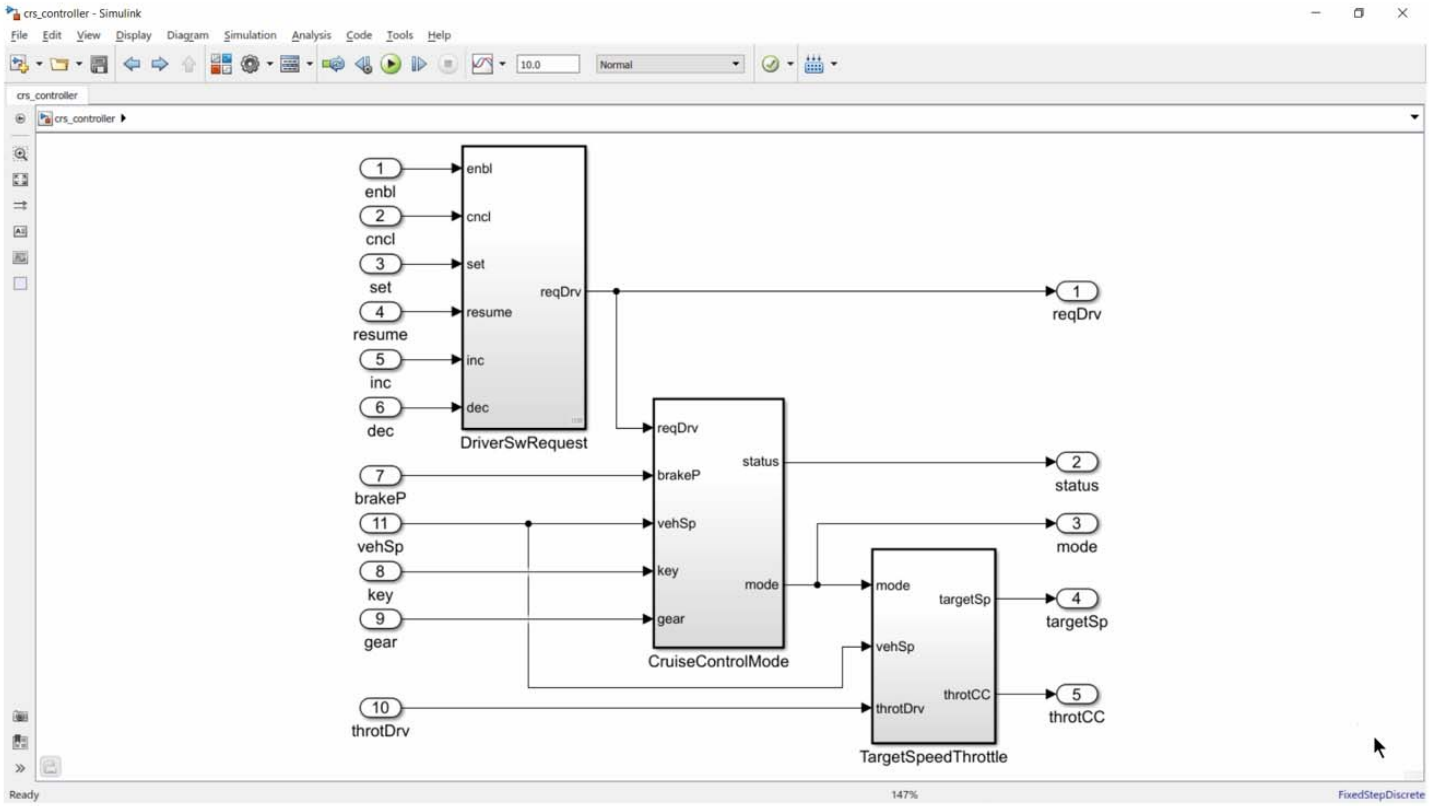


Simulink Requirements Editor

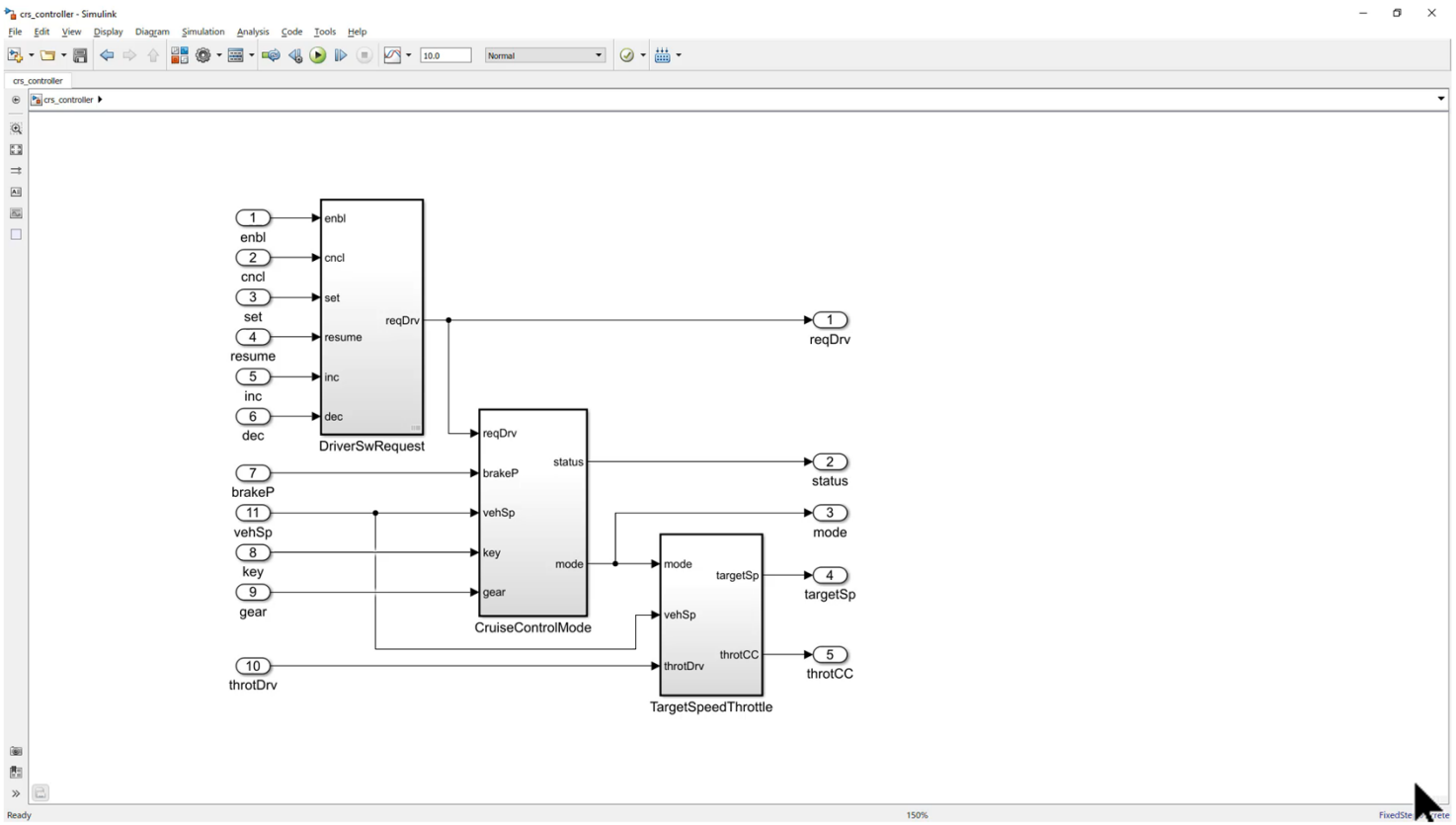




Requirements Perspective View of Model

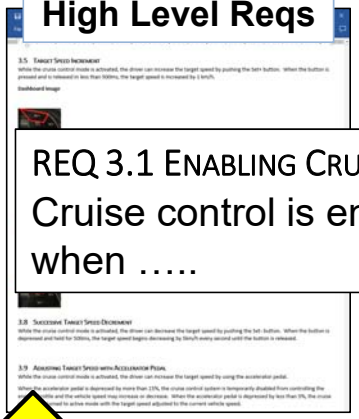


Requirements Perspective View of Model



Requirements Traceability

High Level Reqs



REQ 3.1 ENABLING CRUISE CONTROL
Cruise control is enabled when …..

Derives

Low Level Req

ENABLE SWITCH DETECTION
If the Enable switch is pressed ……

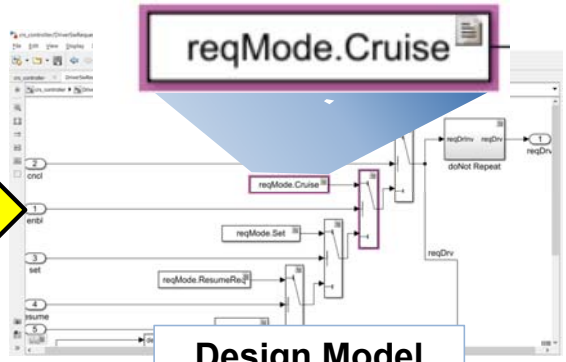
Implemented By

Verified By

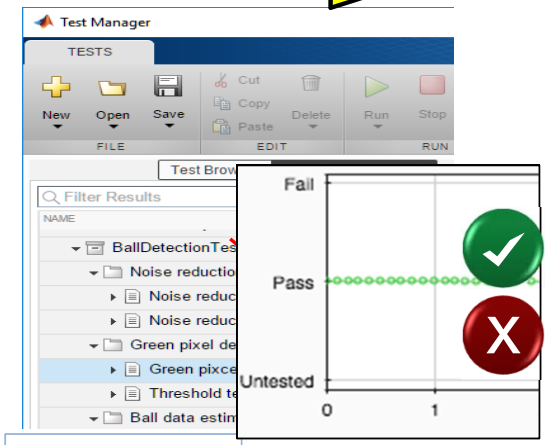
```
void CruiseControl_Req_step(void)
{
  boolean_T AccelResSw_prev;
  boolean_T CoastSetSw_prev;

  /* Chart: '<Root>/Compute target speed' incorporates:
   *  Inport: '<Root>/AccelResSw'
   *  Inport: '<Root>/Brake'
   *  Inport: '<Root>/CoastSetSw'
   *  Inport: '<Root>/CruiseOnOff'
   *  Inport: '<Root>/Speed'
   *
   * Block requirements for '<Root>/AccelResSw':
   *  1. Resume Speed/Accelerate Button (MW_CruiseControl#4)
   *
   * Block requirements for '<Root>/Brake'
   *  1. Vehicle Brake Input (MW_CruiseControl#6)
```

Generated Code

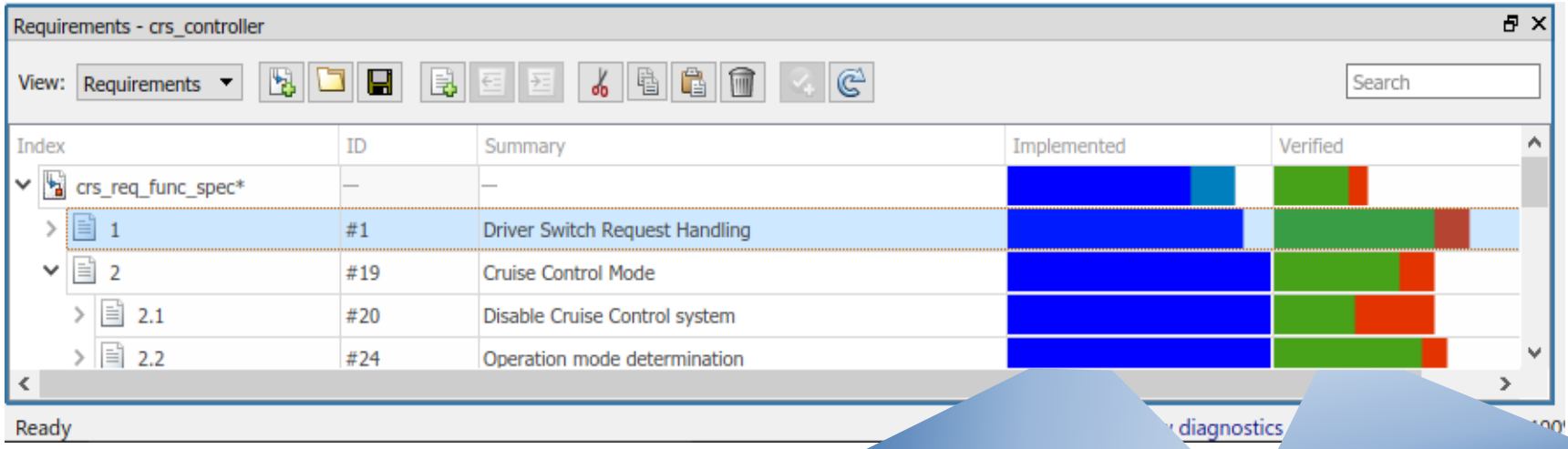


Design Model



Test Case

Track Implementation and Verification Status



Implementation Status

- Implemented
- Justified
- Missing

Verification Status

- Passed
- Failed
- No Result
- Missing

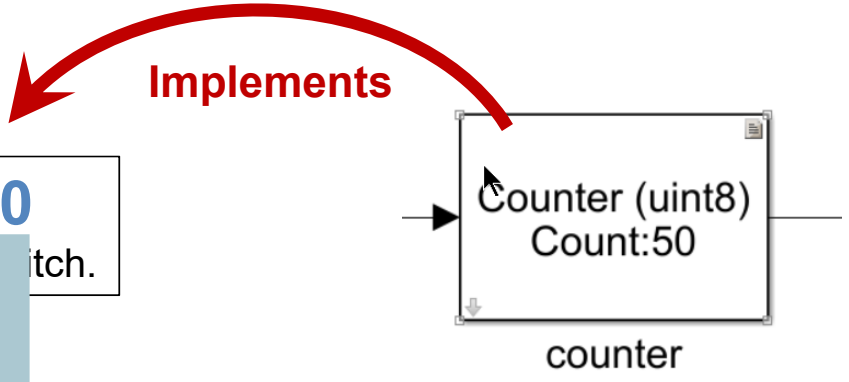
Respond to Requirements Change

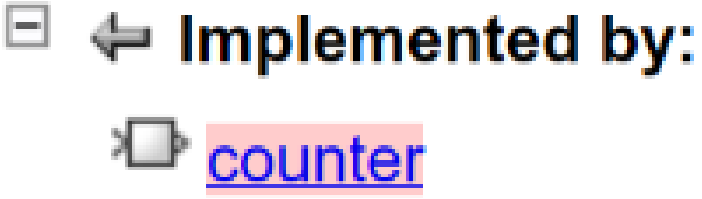
Original Requirement

If the switch is pressed and the counter reaches **50** then it shall be recognized as a long press of the switch.

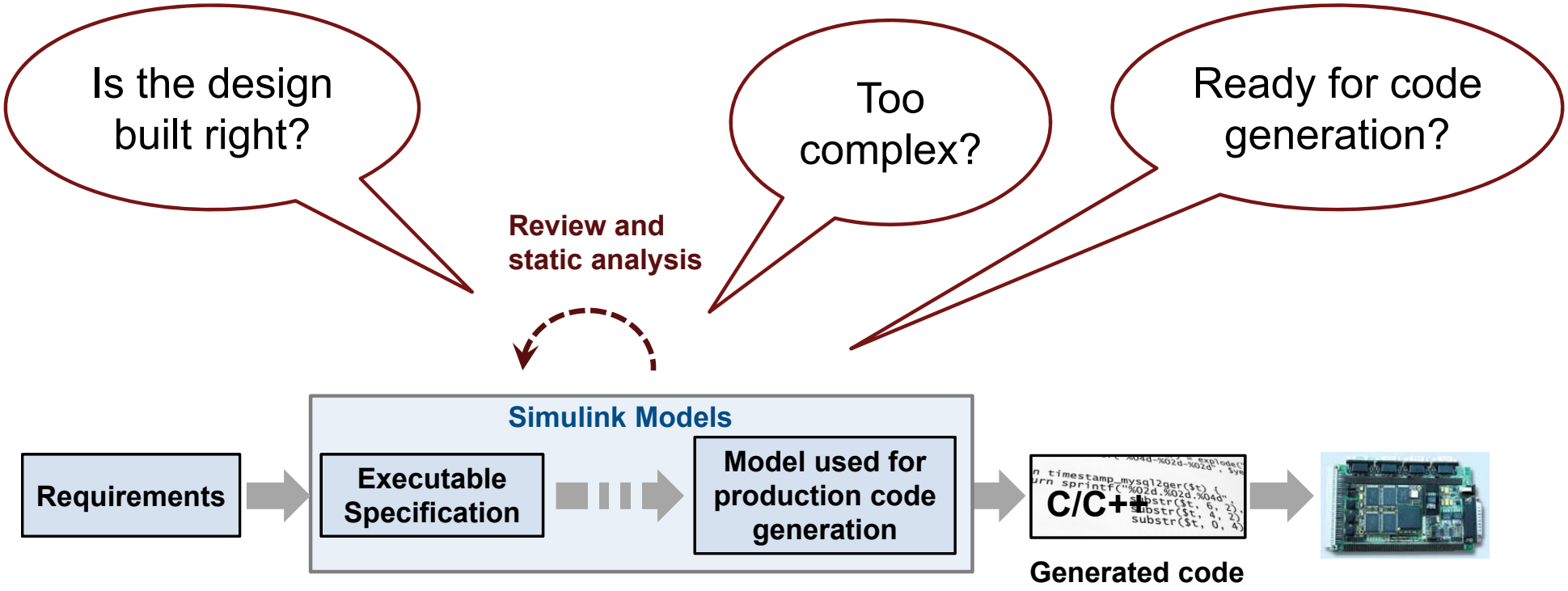
Updated Requirement

If the switch is pressed and the counter reaches **75** then it shall be recognized as a long press of the switch.

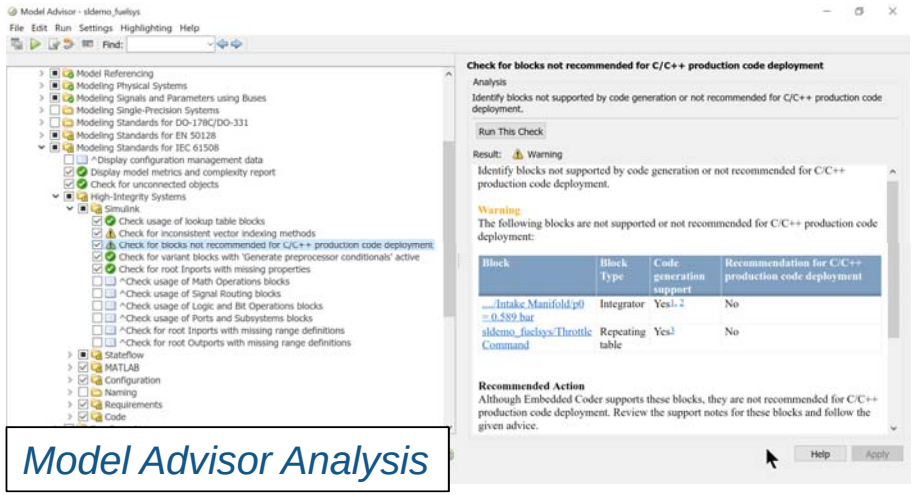




Verify Design to Guidelines and Standards



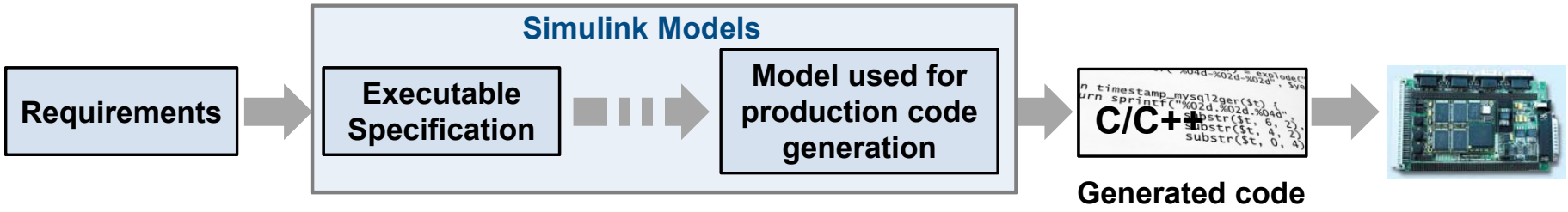
Automated Static Analysis of the Design



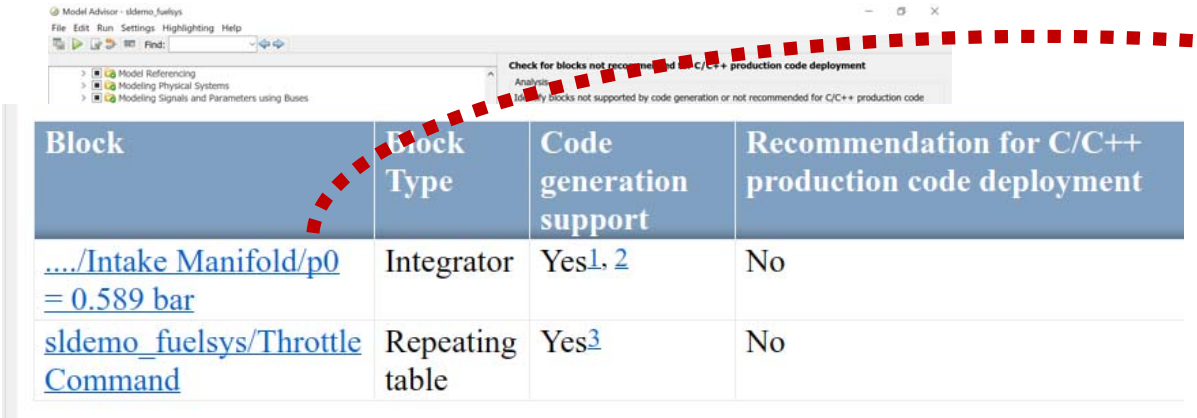
Model Advisor Analysis

Check the model for

- Readability and Semantics
- Performance and Efficiency
- Clones
- And more……

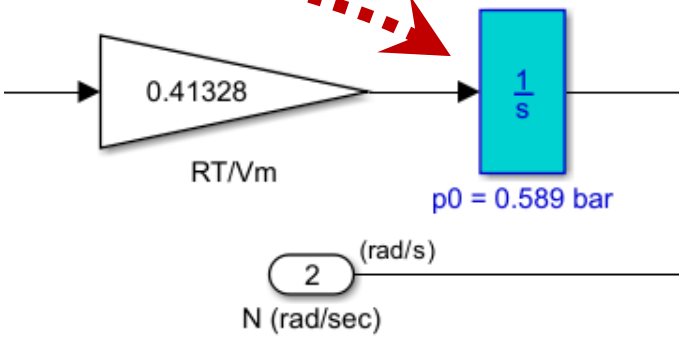


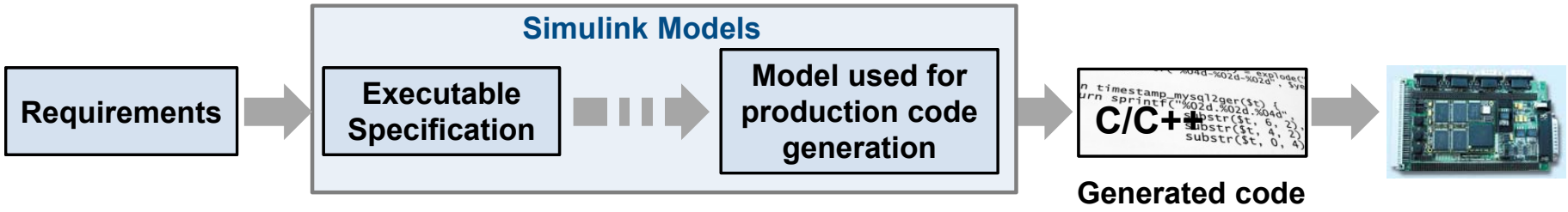
Navigate to Problematic Blocks



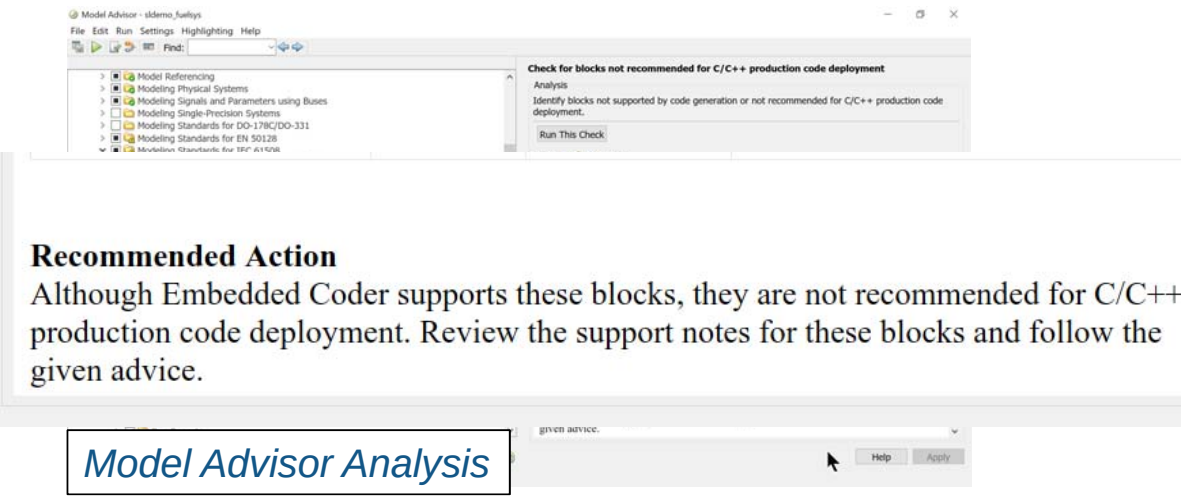
Block	Block Type	Code generation support	Recommendation for C/C++ production code deployment
..../Intake Manifold/p0 = 0.589 bar	Integrator	Yes[1, 2]	No
sldemo_fuelsys/Throttle Command	Repeating table	Yes[3]	No

Model Advisor Analysis

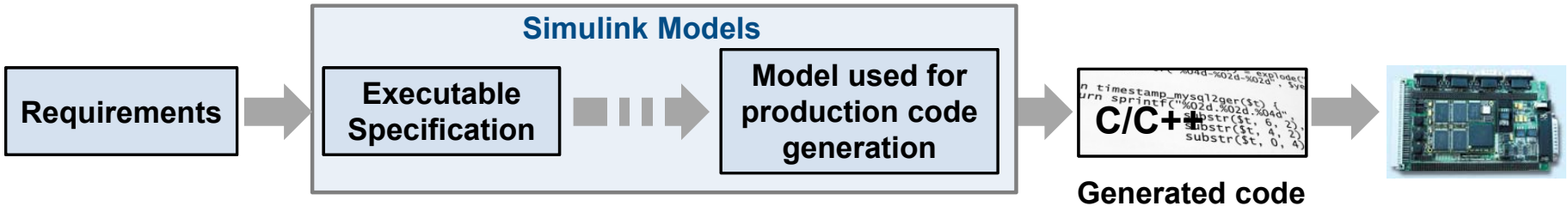




Guidance Provided to Address Issues w/ Auto-Correct

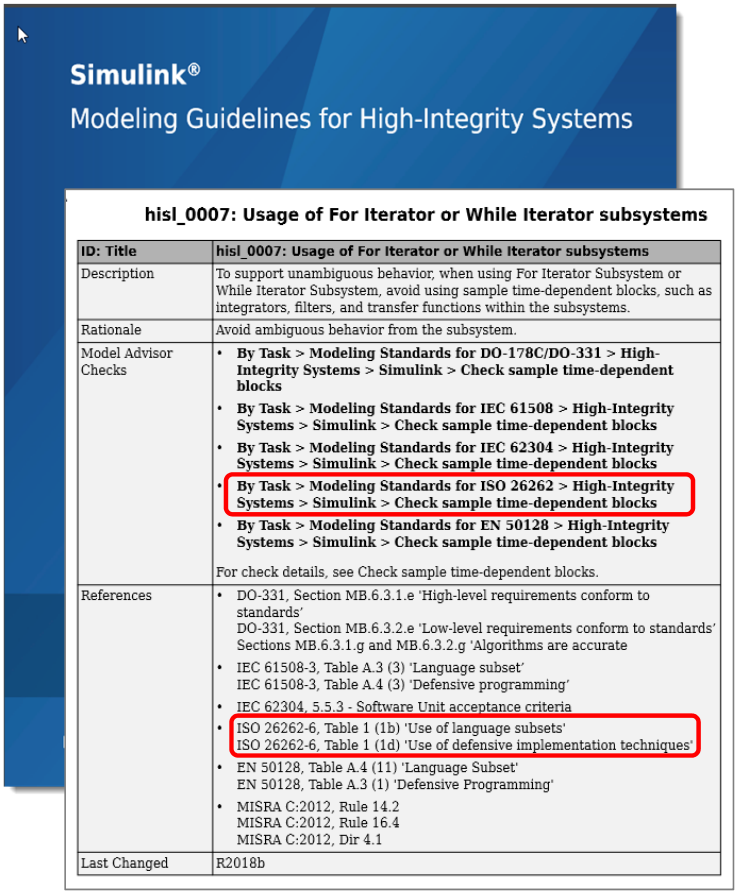


Model Advisor Analysis



Modeling Guidelines for High-Integrity Systems

- Leverage industry-best practices and MathWorks tool expertise when developing high-integrity systems
- Modeling Guidelines with corresponding Model Advisor checks
- Mapped to the modeling standards and guidelines objectives of industry standards like ISO 26262 and MISRA-C



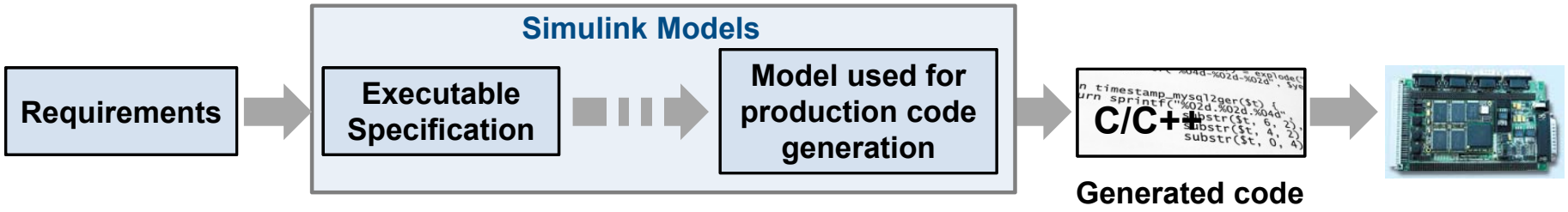
hisl_0007: Usage of For Iterator or While Iterator subsystems

ID: Title	hisl_0007: Usage of For Iterator or While Iterator subsystems
Description	To support unambiguous behavior, when using For Iterator Subsystem or While Iterator Subsystem, avoid using sample time-dependent blocks, such as integrators, filters, and transfer functions within the subsystems.
Rationale	Avoid ambiguous behavior from the subsystem.
Model Advisor Checks	• **By Task > Modeling Standards for DO-178C/DO-331 > High-Integrity Systems > Simulink > Check sample time-dependent blocks** • **By Task > Modeling Standards for IEC 61508 > High-Integrity Systems > Simulink > Check sample time-dependent blocks** • **By Task > Modeling Standards for IEC 62304 > High-Integrity Systems > Simulink > Check sample time-dependent blocks** • **By Task > Modeling Standards for ISO 26262 > High-Integrity Systems > Simulink > Check sample time-dependent blocks** • **By Task > Modeling Standards for EN 50128 > High-Integrity Systems > Simulink > Check sample time-dependent blocks** For check details, see Check sample time-dependent blocks.
References	• DO-331, Section MB.6.3.1.e 'High-level requirements conform to standards' DO-331, Section MB.6.3.2.e 'Low-level requirements conform to standards' Sections MB.6.3.1.g and MB.6.3.2.g 'Algorithms are accurate • IEC 61508-3, Table A.3 (3) 'Language subset' IEC 61508-3, Table A.4 (3) 'Defensive programming' • IEC 62304, 5.5.3 - Software Unit acceptance criteria • ISO 26262-6, Table 1 (1b) 'Use of language subsets' ISO 26262-6, Table 1 (1d) 'Use of defensive implementation techniques' • EN 50128, Table A.4 (11) 'Language Subset' EN 50128, Table A.3 (1) 'Defensive Programming' • MISRA C:2012, Rule 14.2 MISRA C:2012, Rule 16.4 MISRA C:2012, Dir 4.1
Last Changed	R2018b

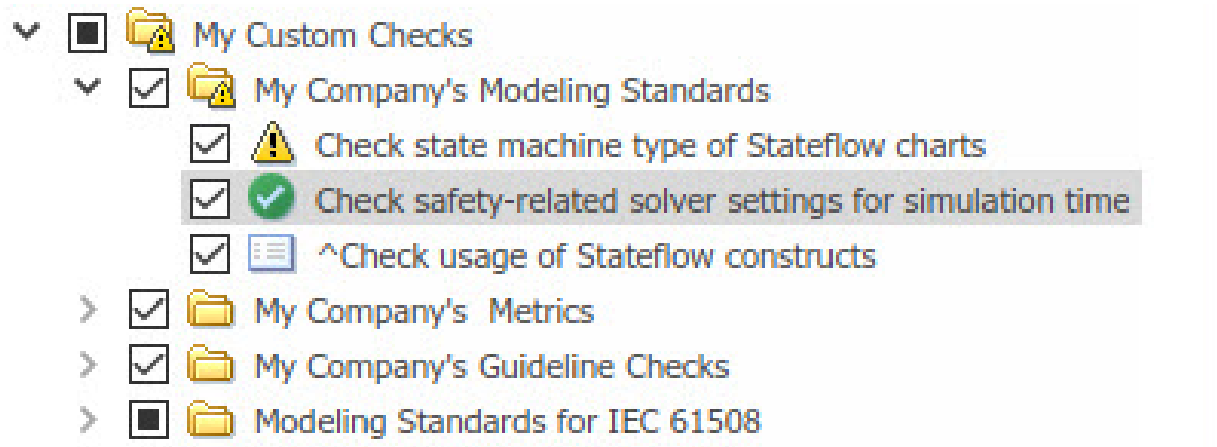
http://www.mathworks.com/help/pdf_doc/simulink/hi_guidelines.pdf

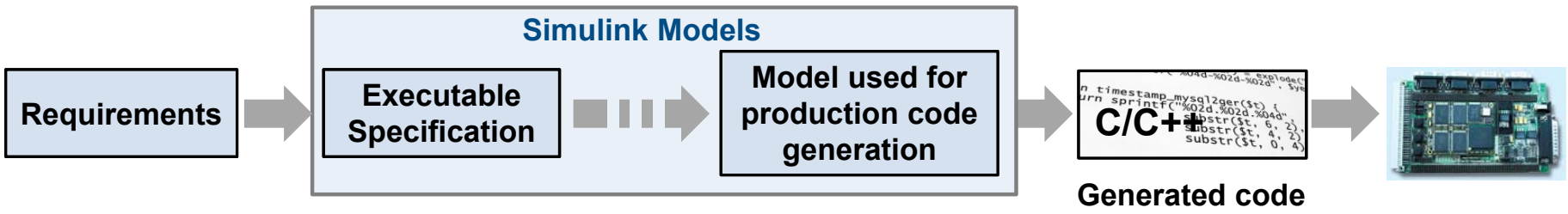
Built in Checks for Industry Standards and Guidelines

- DO-178/DO-331
- ISO 26262
- IEC 61508
- IEC 62304
- EN 50128
- MISRA C:2012
- CERT C, CWE, ISO/IEC TS 17961
- MAAB (MathWorks Automotive Advisory Board)
- JMAAB (Japan MATLAB Automotive Advisory Board)

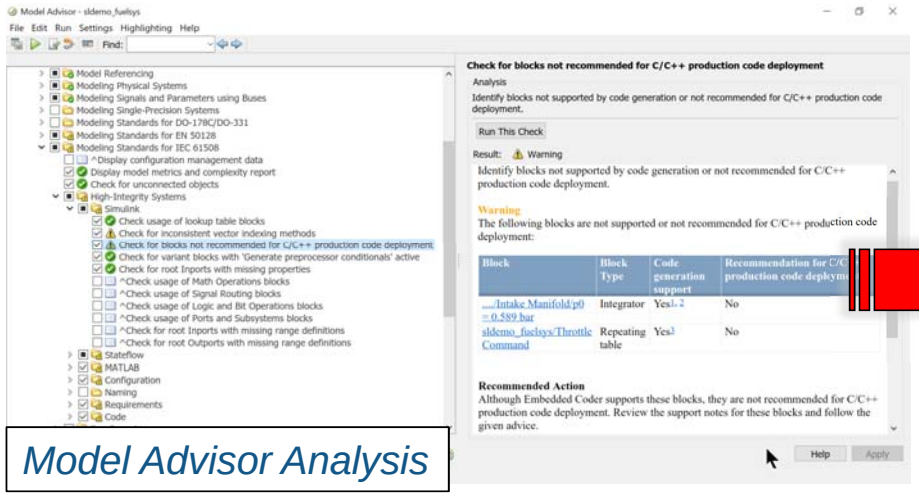


Configure and Customize Static Analysis

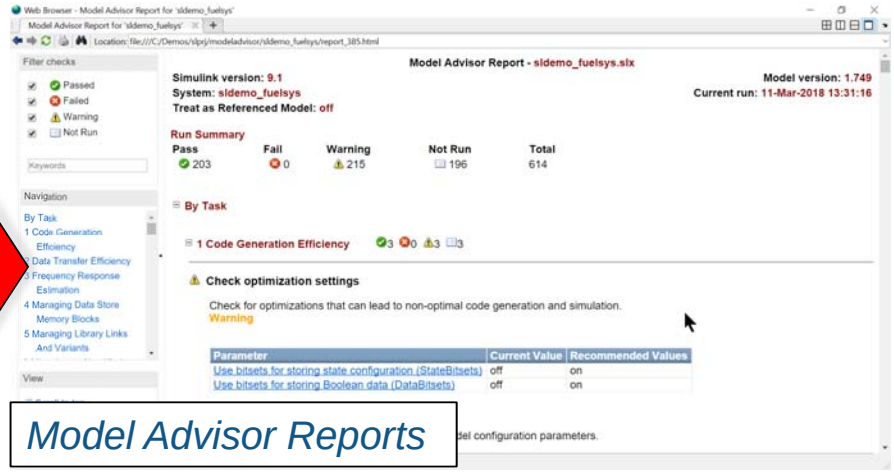




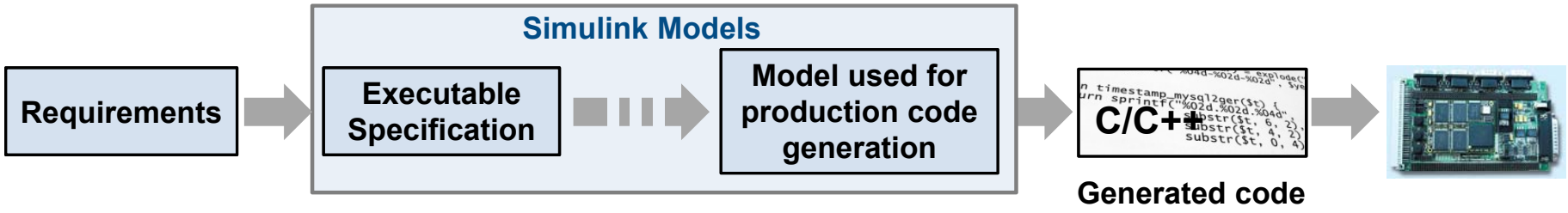
Generate Reports for Reviews and Documentation



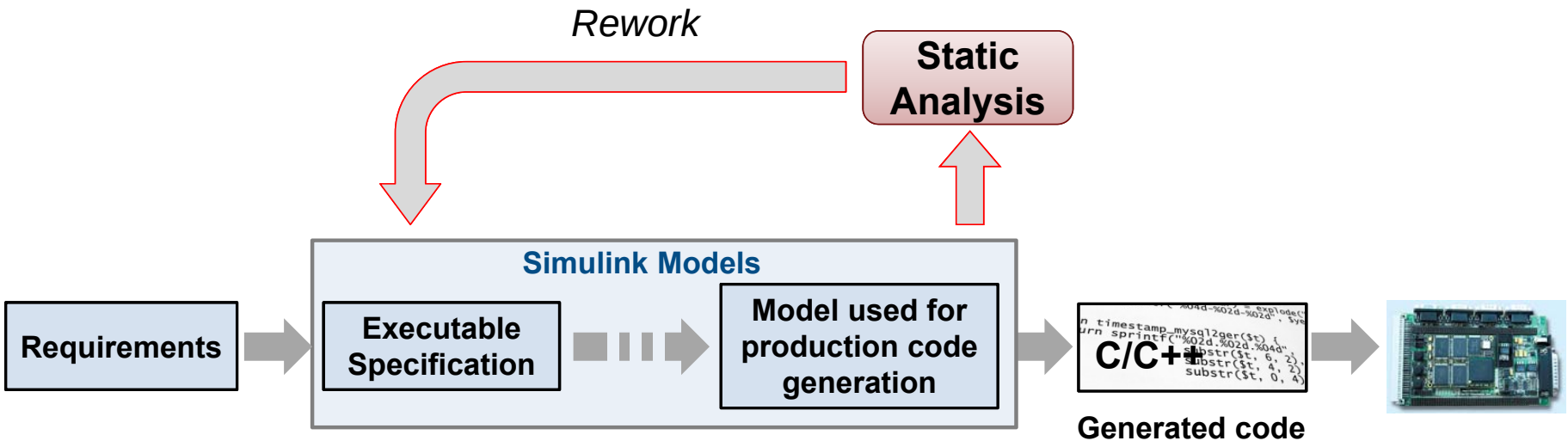
Model Advisor Analysis



Model Advisor Reports

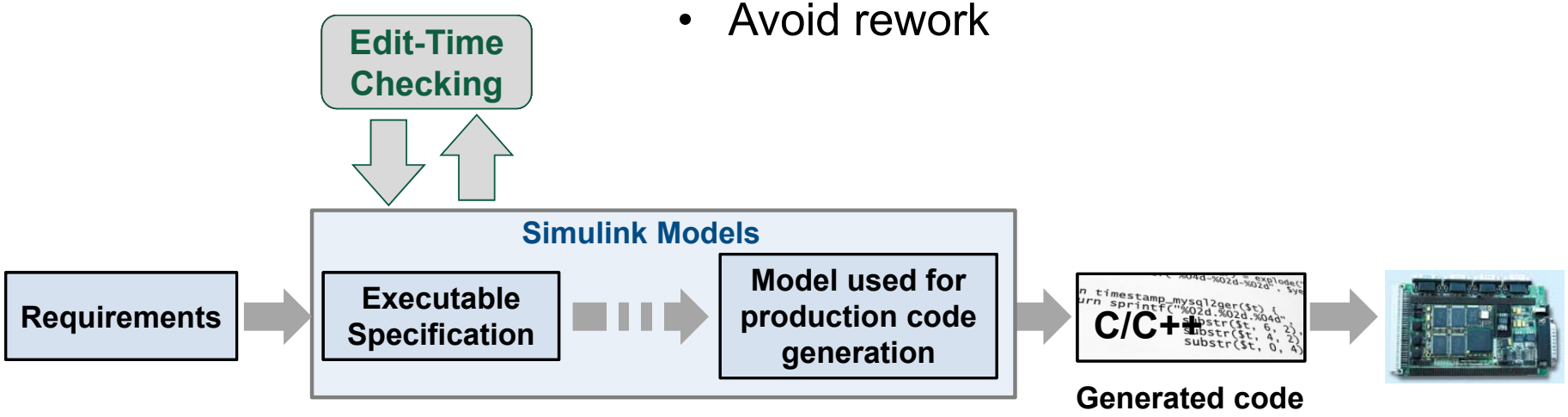


Checks for Standards and Guidelines are often Performed Late

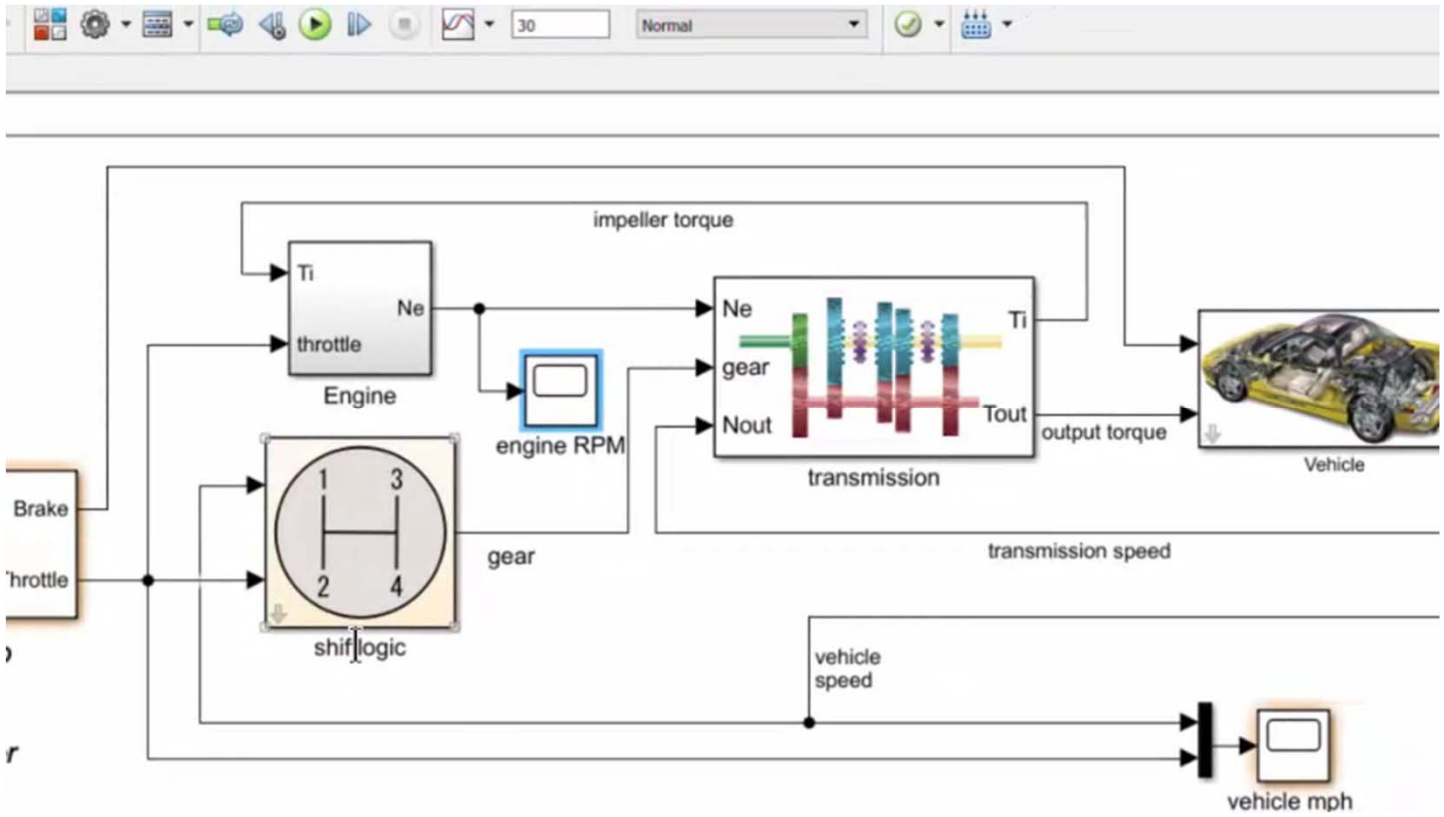


Shift Verification Earlier with Edit-Time Checking

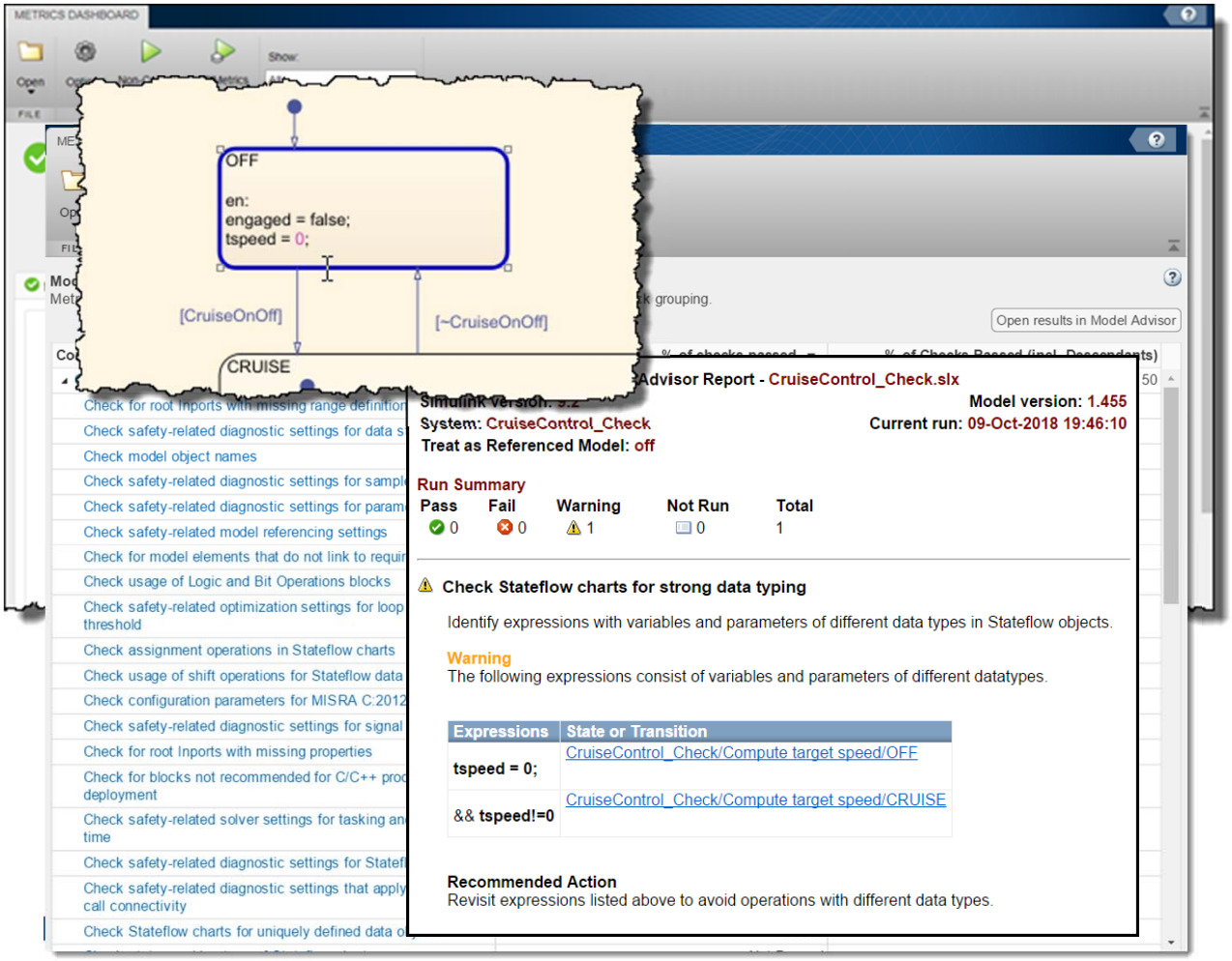
- Highlight violations as you edit
- Fix issues earlier
- Avoid rework



Find Compliance Issues while you Design

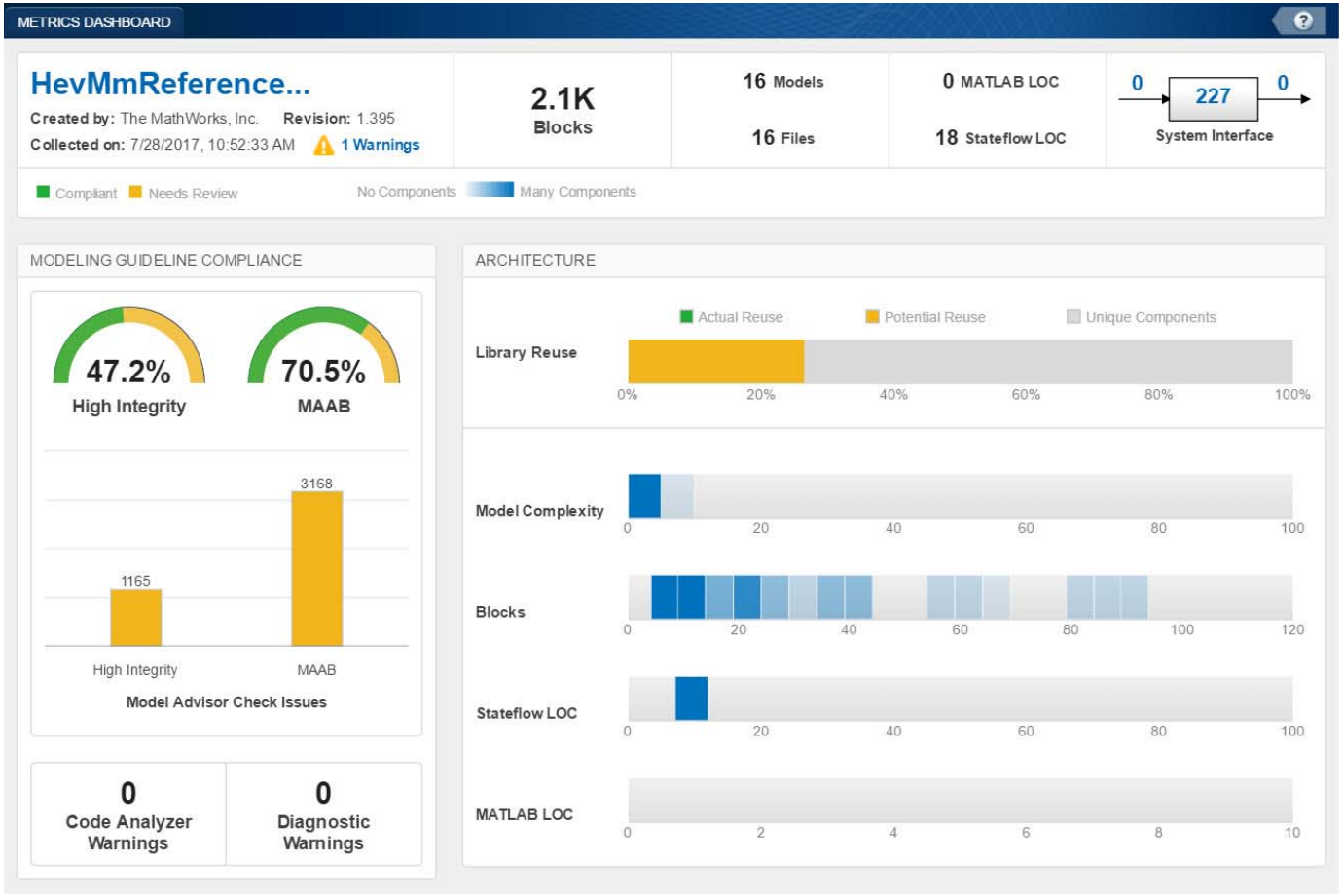


Modeling Standards Checking with Simulink Check



- Static analysis of models against a set of checks
- Modeling Standards Checks
  - MAAB Style Guidelines V3.0
  - ISO 26262
  - MISRA C:2012
- Additional Checks
  - Model Metrics
  - Tool Bug Reports (Cert Kit)
  - Requirements Consistency

Assess Quality with Metrics Dashboard

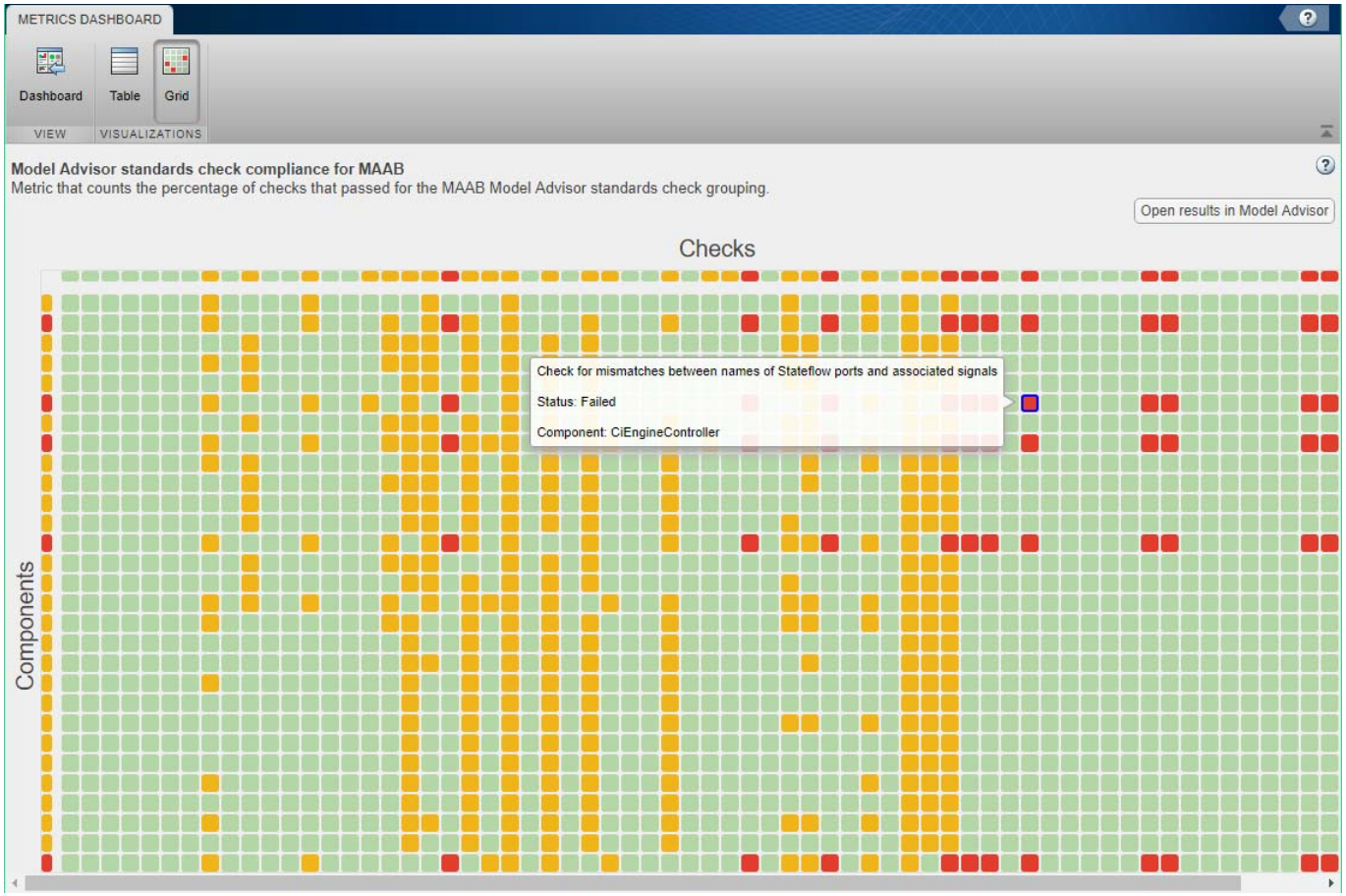


- Consolidated view of metrics
  - Size
  - Compliance
  - Complexity
- Identify where problem areas may be

Grid View for Metrics Analysis

MathWorks

R2018a

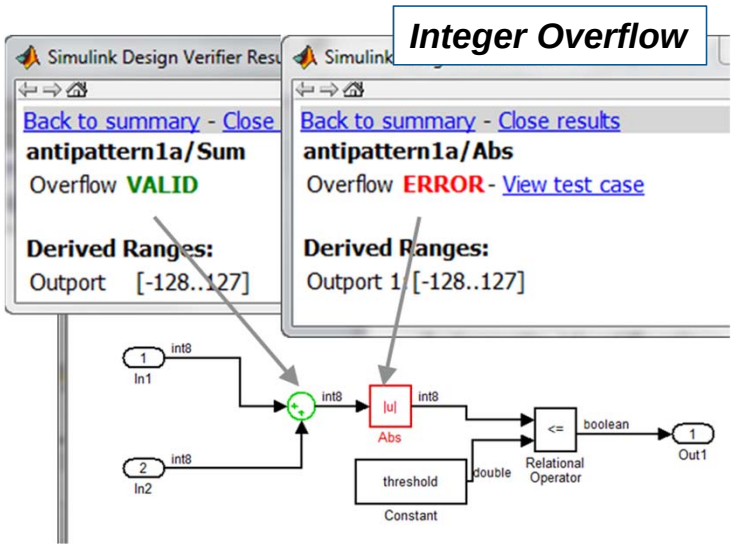


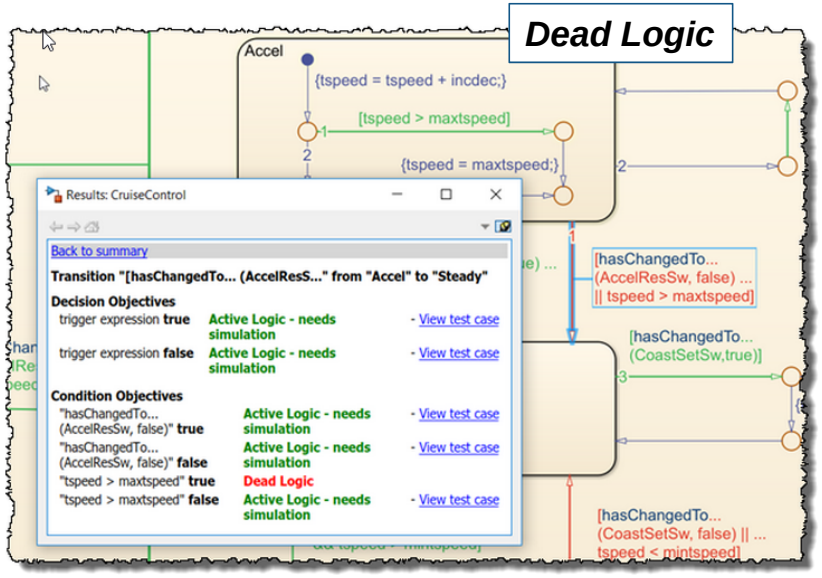
- Visualize Standards Check Compliance
  - Find Issues
  - Identify patterns
  - See hot spots

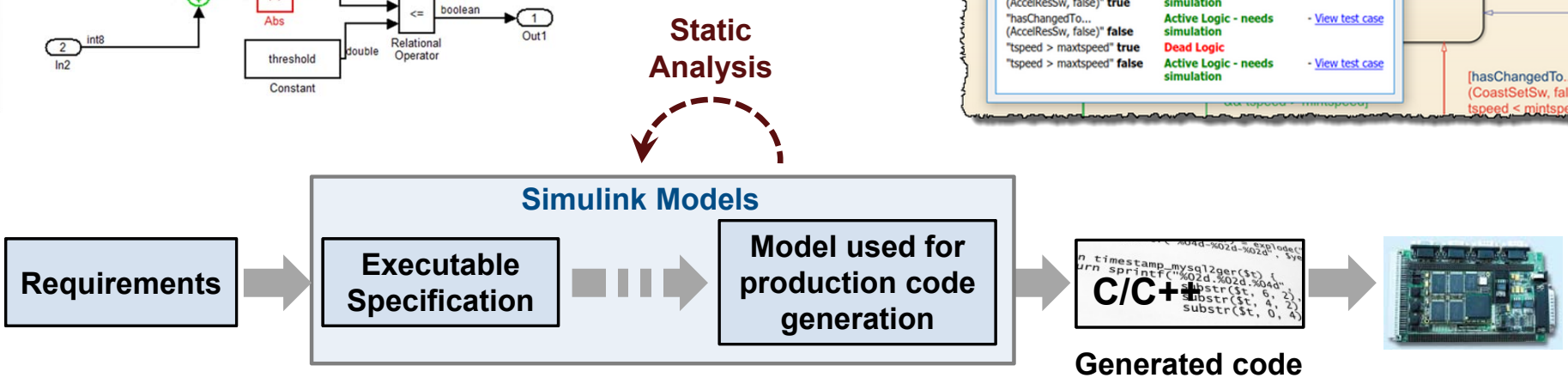
Legend:

Red:	Fail
Orange:	Warning
Green:	Pass
Gray:	Not run

Static Analysis for Detecting Design Errors

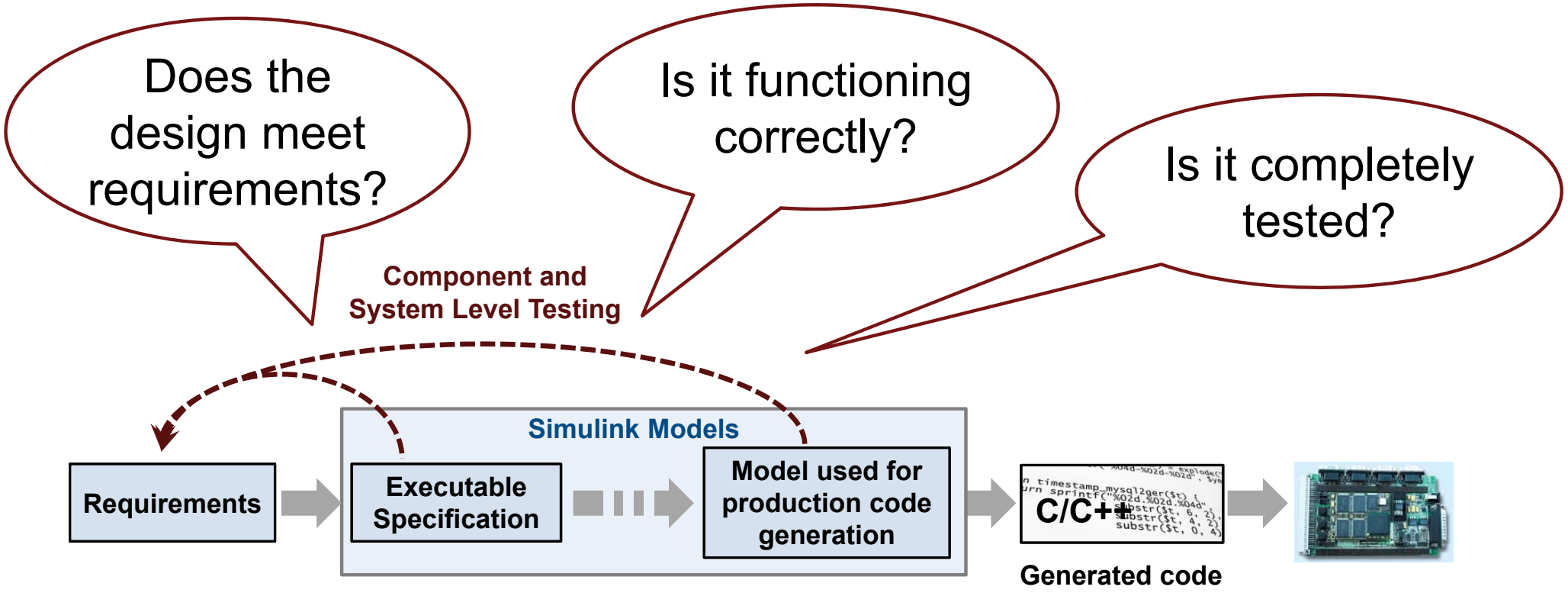




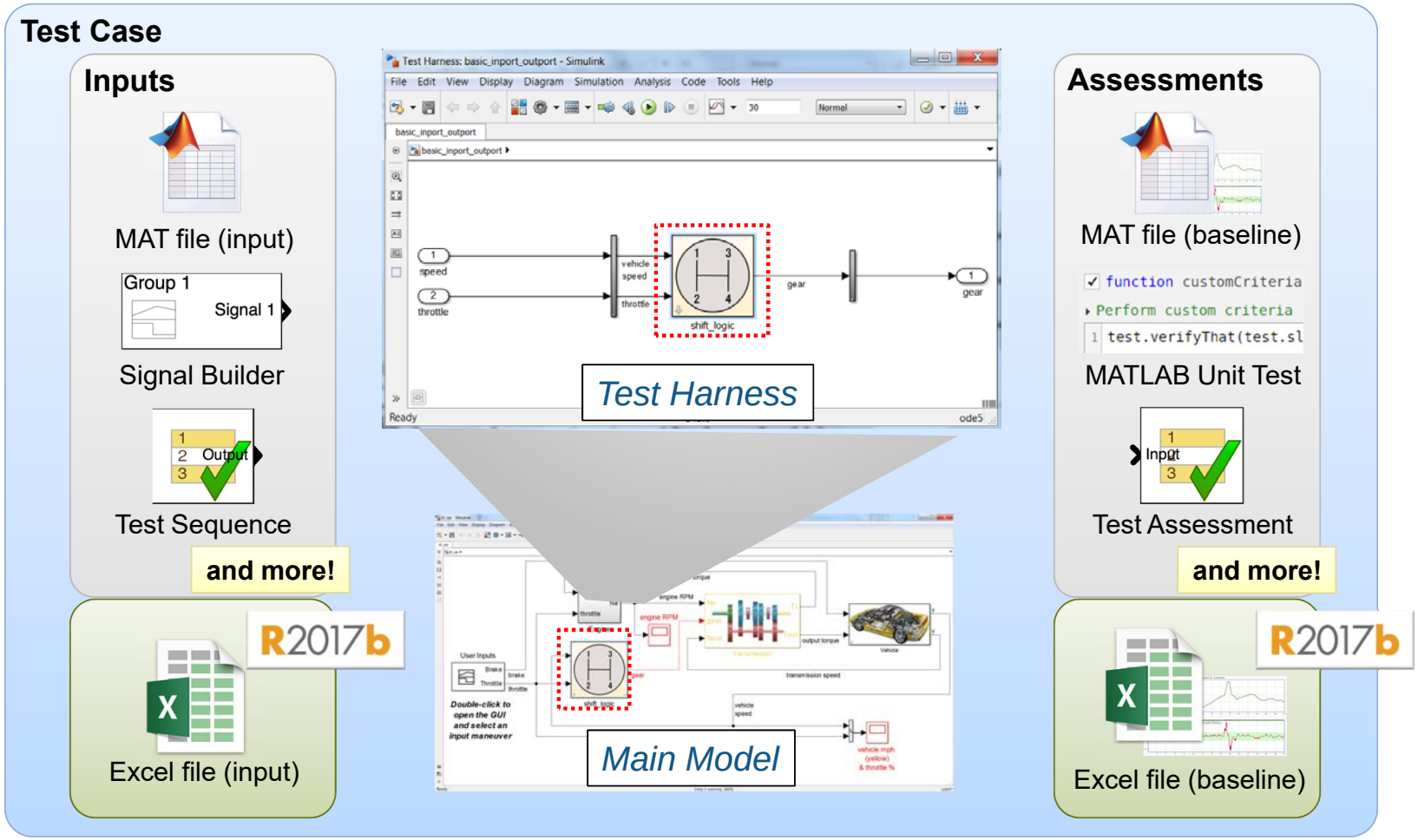


- Find run-time design errors
- Generates a test case to reproduce the issue for debugging

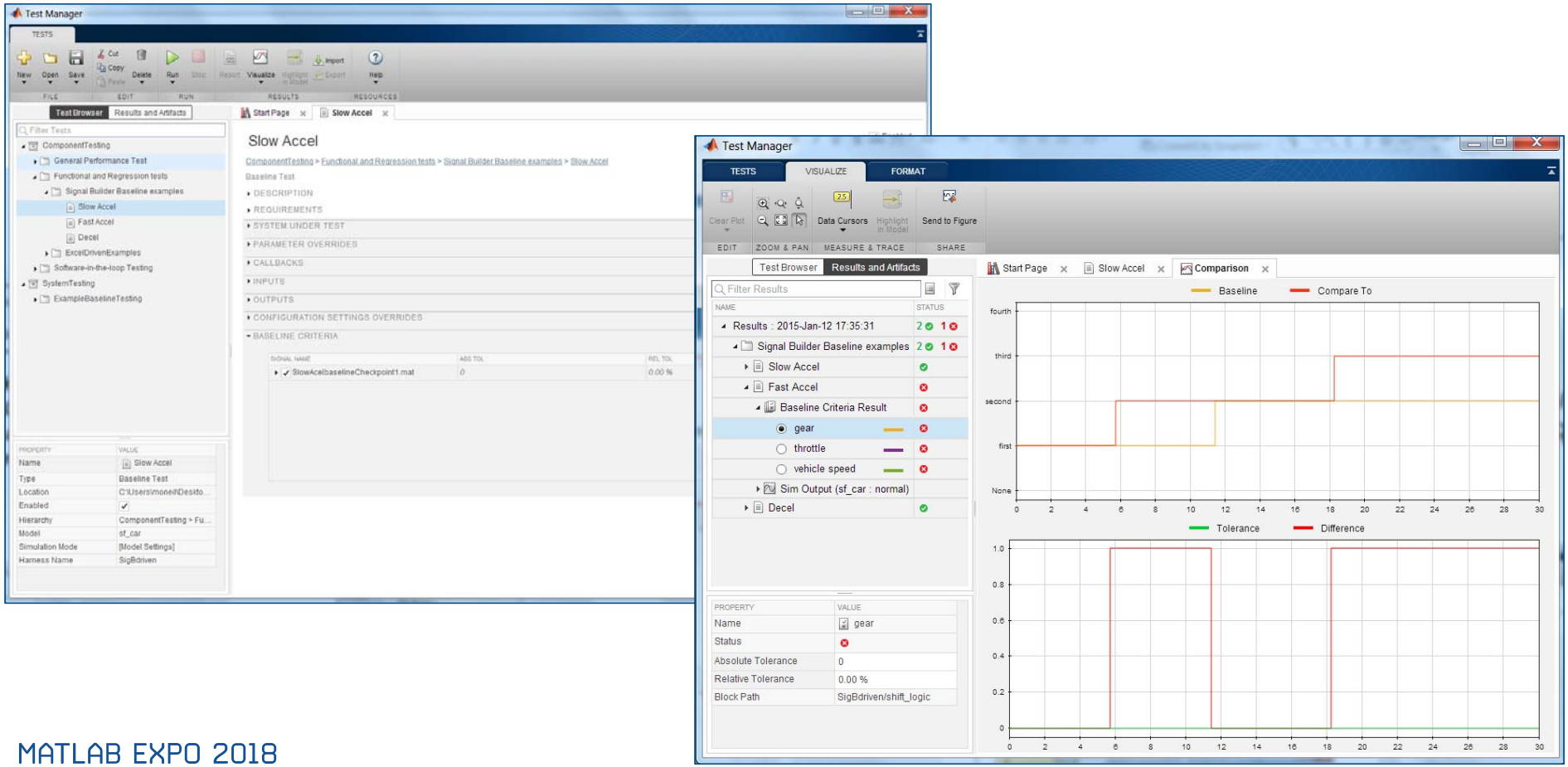
Functional Testing



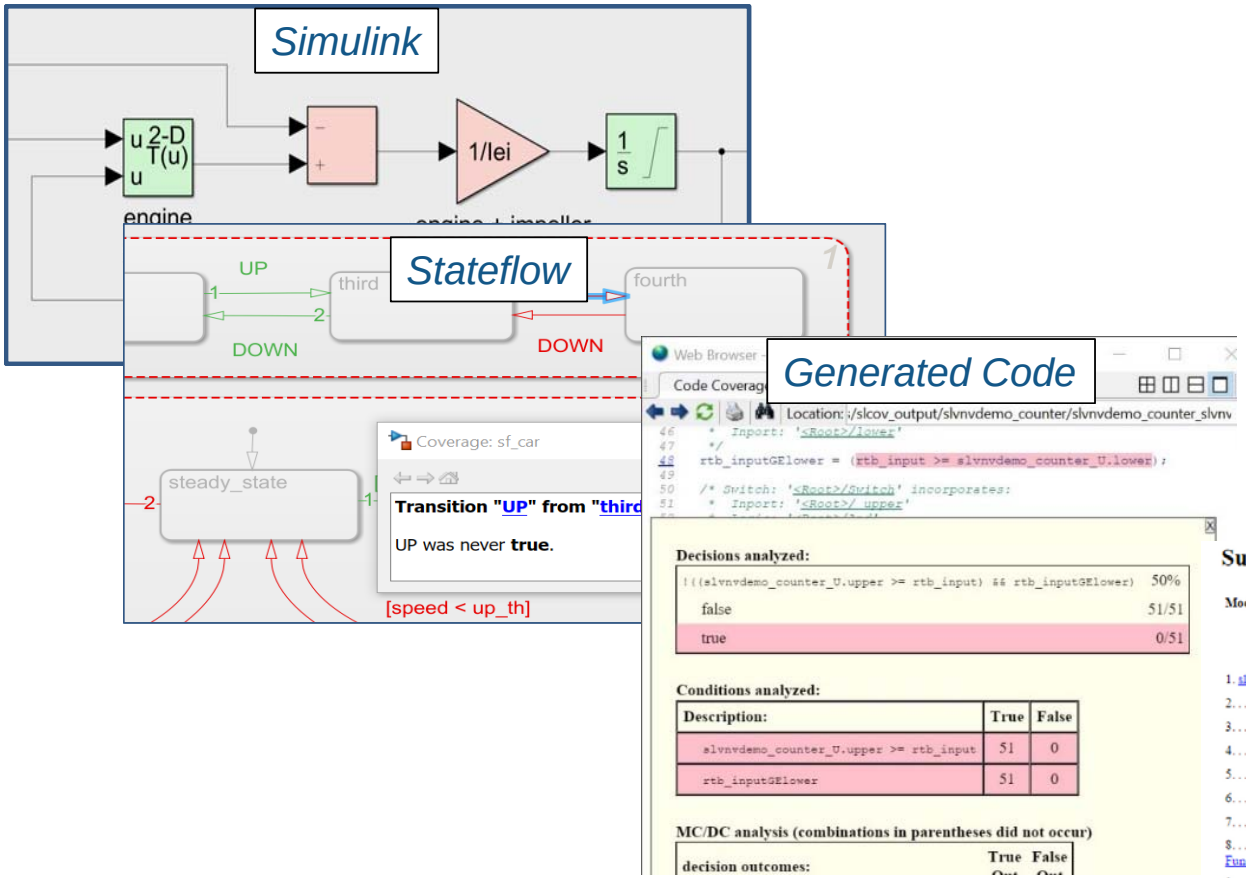
Systematic Functional Testing



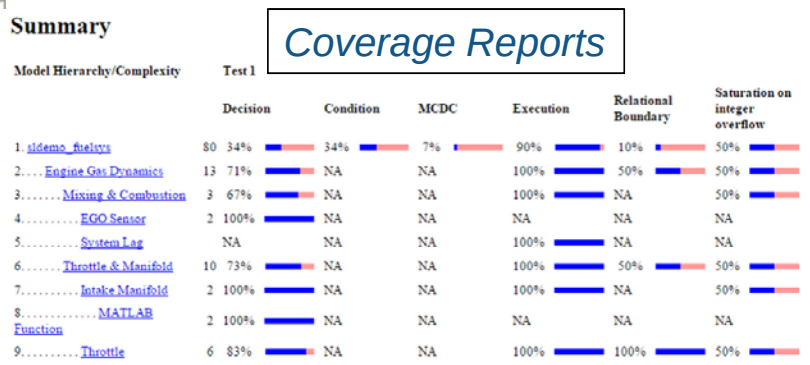
Test Execution and Results Analysis



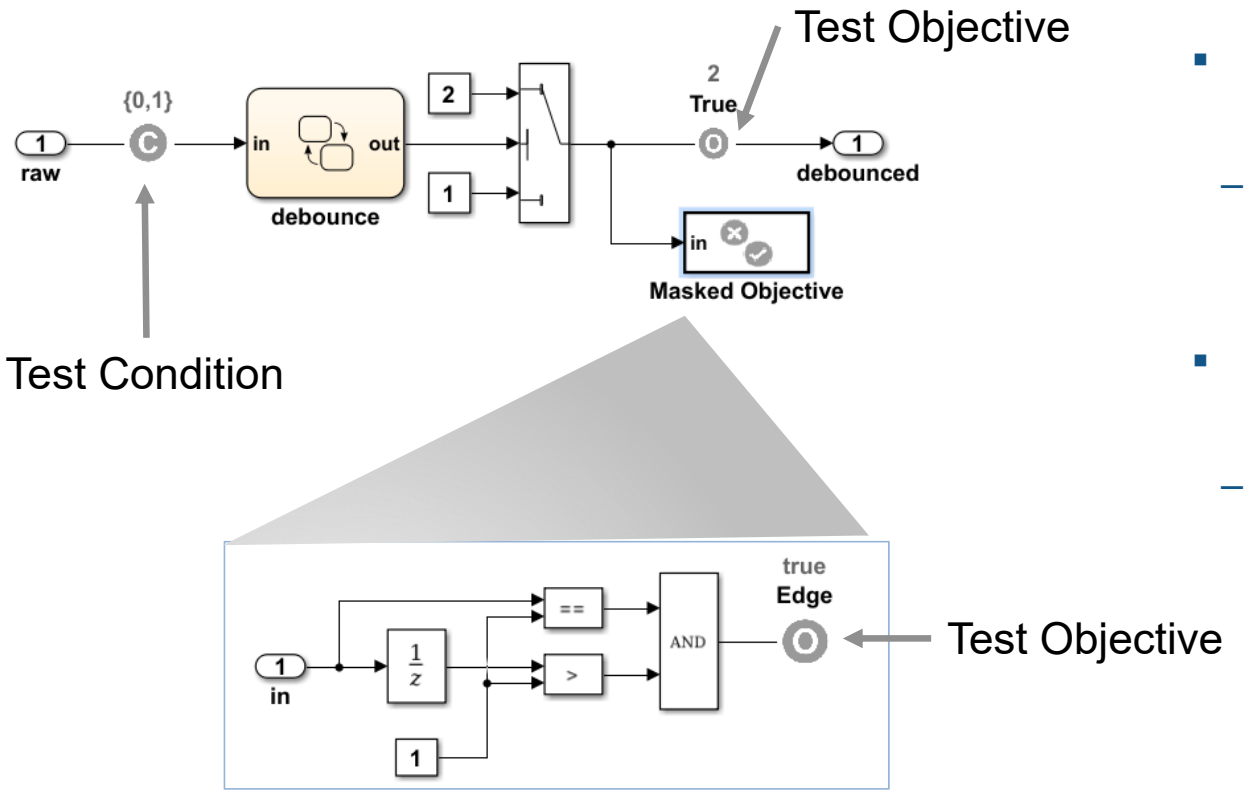
Coverage Analysis to Measure Test Completeness



- Identify testing gaps
- Missing requirements
- Unintended Functionality
- Design Errors

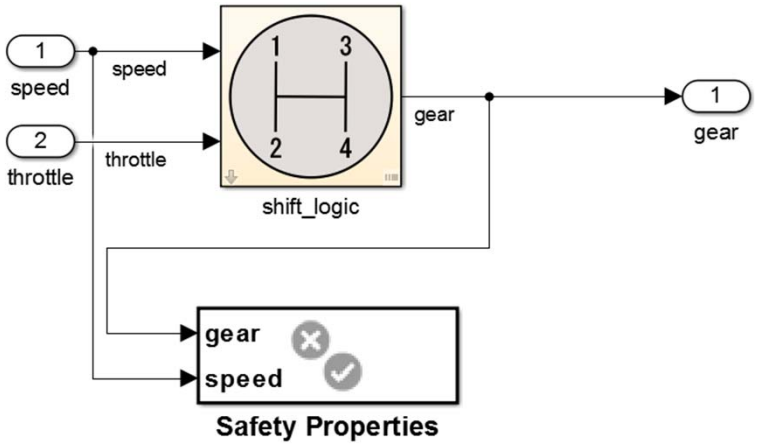


Test Case Generation for Functional Testing

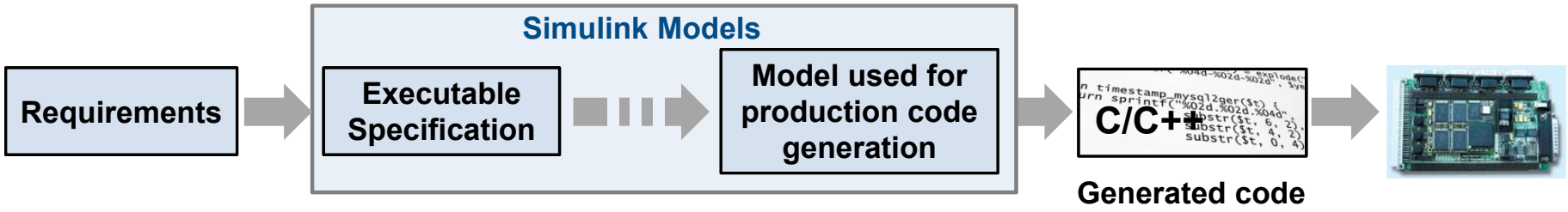


- Specify functional test objectives
  - Define custom objectives that signals must satisfy in test cases
- Specify functional test conditions
  - Define constraints on signal values to constrain test generator

Prove Design Meets Requirements



- Prove design properties using formal requirement models
- Model functional and safety requirements
- Generates counter example for analysis and debugging

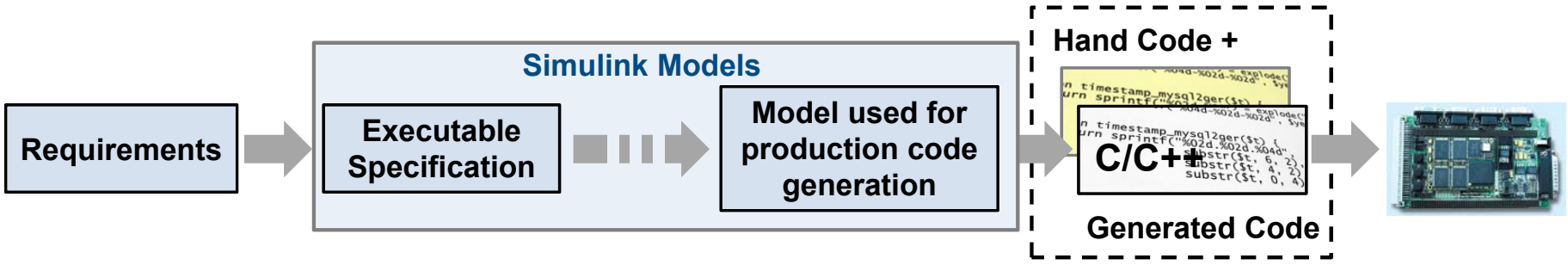


Static Code Analysis

Is the code compliant to MISRA?

Is integrated code free of run-time errors?

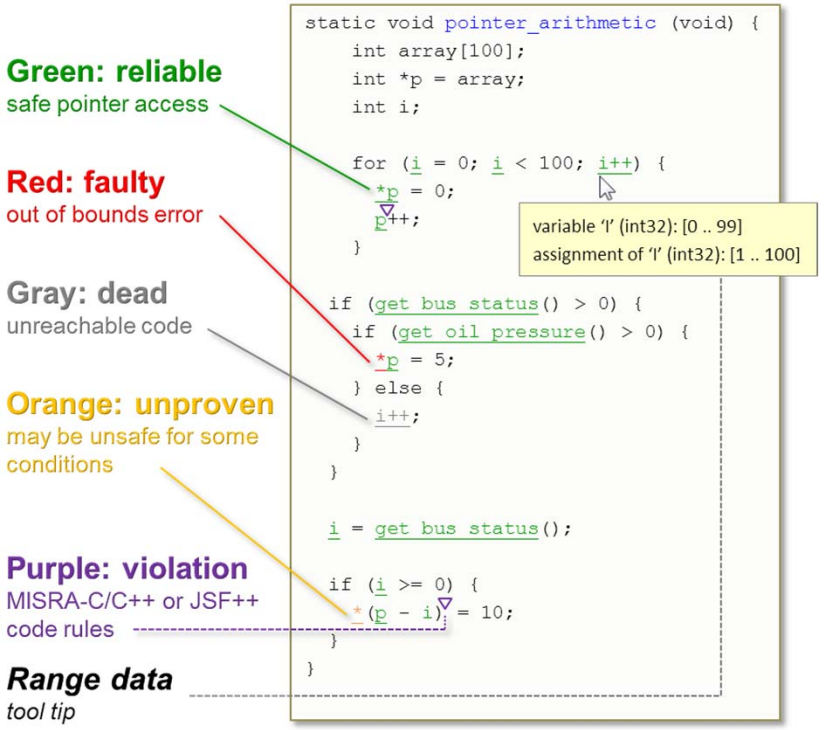
Is interface between generated and other code fully tested?



Generated Code is integrated with Hand Code

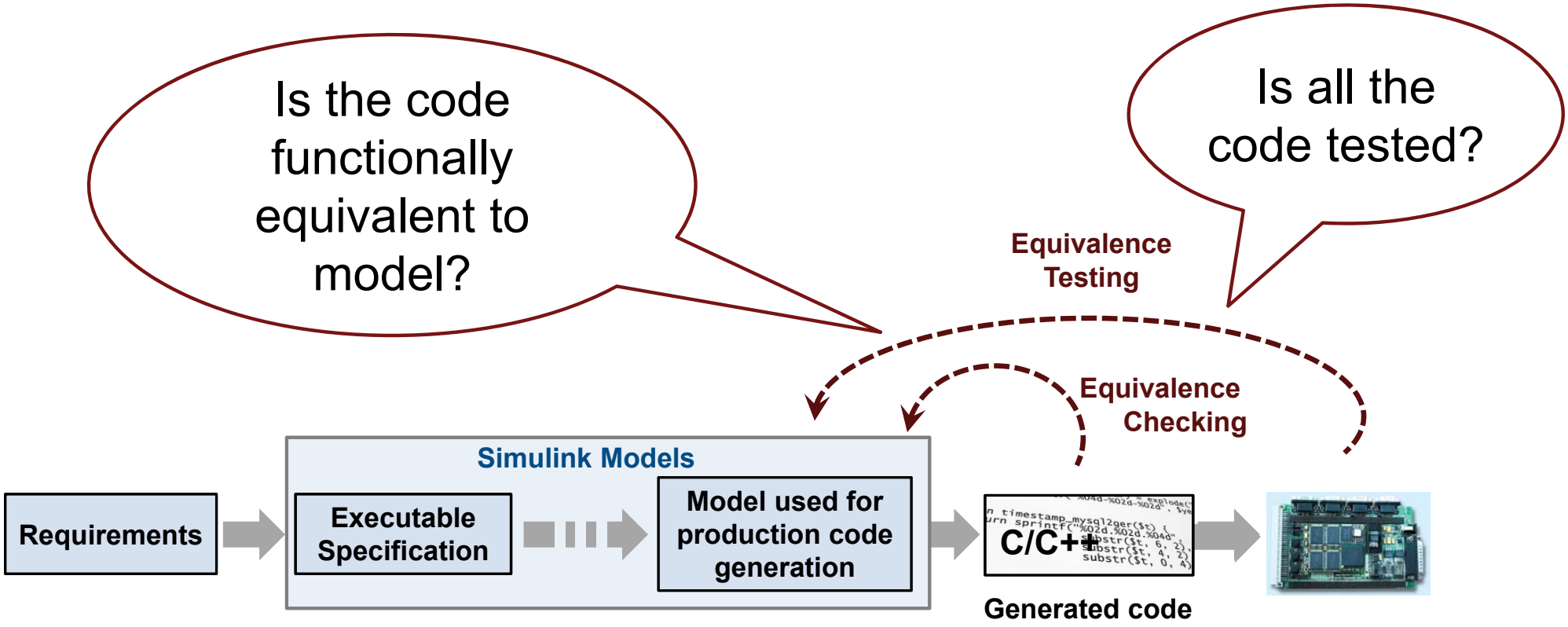
Static Code Analysis with Polyspace

- Code metrics and standards
  - Comment density, cyclomatic complexity,…
  - MISRA and Cybersecurity standards
  - Support for DO-178, ISO 26262, ….
- Bug finding and code proving
  - Check data and control flow of software
  - Detect bugs and security vulnerabilities
  - Prove absence of runtime errors



Results from Polyspace Code Prover

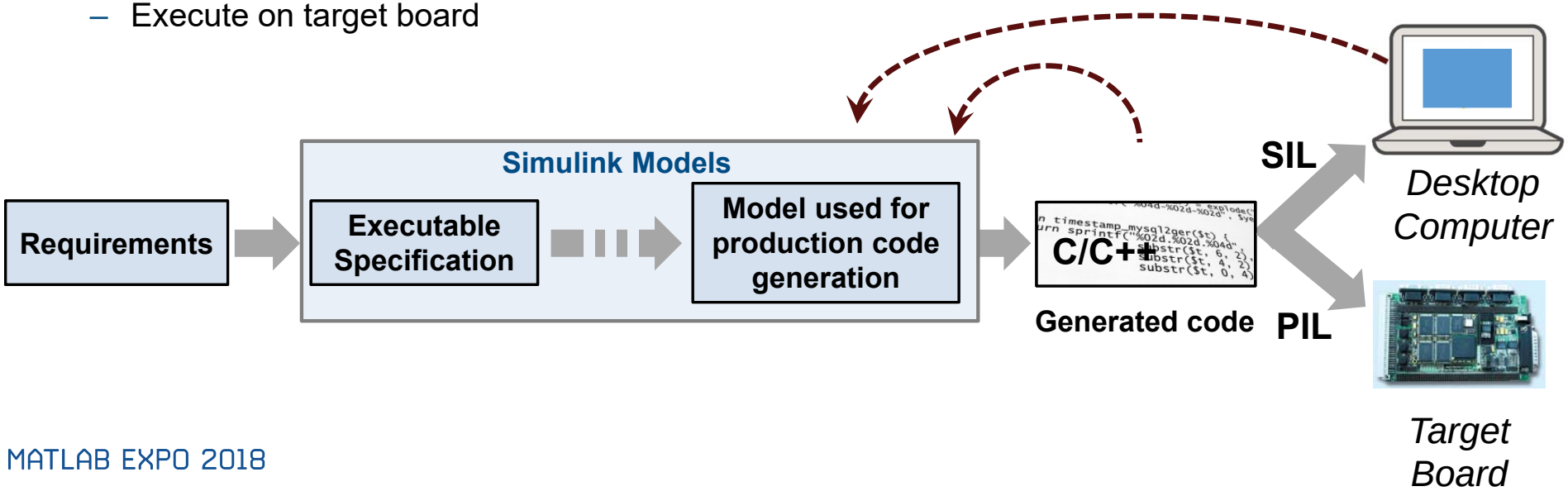
Equivalence Testing



Equivalence Testing

- Software in the Loop (SIL)
  - Show functional equivalence, model to code
  - Execute on desktop / laptop computer
- Processor in the Loop (PIL)
  - Numerical equivalence, model to target code
  - Execute on target board

- Re-use tests developed for model to test code
- Check for equivalent outputs model to code
- Collect code coverage, compare to model coverage



Qualify tools with IEC Certification Kit and DO Qualification Kit

- Qualify code generation and verification tools
- Includes documentation, test cases and procedures

KOSTAL Asia R&D Center Receives ISO 26262 ASIL D Certification for Automotive Software Developed with Model-Based Design

Kostal's electronic steering column lock module.

BAE Systems Delivers DO-178B Level A Flight Software on Schedule with Model-Based Design

Primary flight control computers from BAE Systems.

Lear Delivers Quality Body Control Electronics Faster Using Model-Based Design

Challenge

Design, verify, and implement high-quality automotive body control electronics

Solution

Use Model-Based Design to enable early and continuous verification via simulation, SIL, and HIL testing

Results

- Requirements validated early. Over 95% of issues fixed before implementation, versus 30% previously
- Development time cut by 40%. 700,000 lines of code generated and test cases reused throughout the development cycle
- Zero warranty issues reported

Lear automotive body electronic control unit.

"We adopted Model-Based Design not only to deliver better-quality systems faster, but because we believe it is a smart choice. Recently we won a project that several of our competitors declined to bid on because of its tight time constraints. Using Model-Based Design, we met the original delivery date with no problem."

- Jason Bauman, Lear Corporation

Link to user story

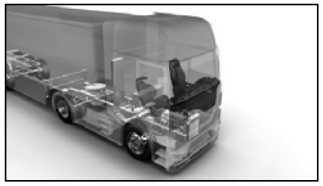
Customer References and Applications

Airbus Helicopters Accelerates Development of DO-178B Certified Software with Model-Based Design

Software testing time cut by two-thirds

LS Automotive Reduces Development Time for Automotive Component Software with Model-Based Design

Specification errors detected early



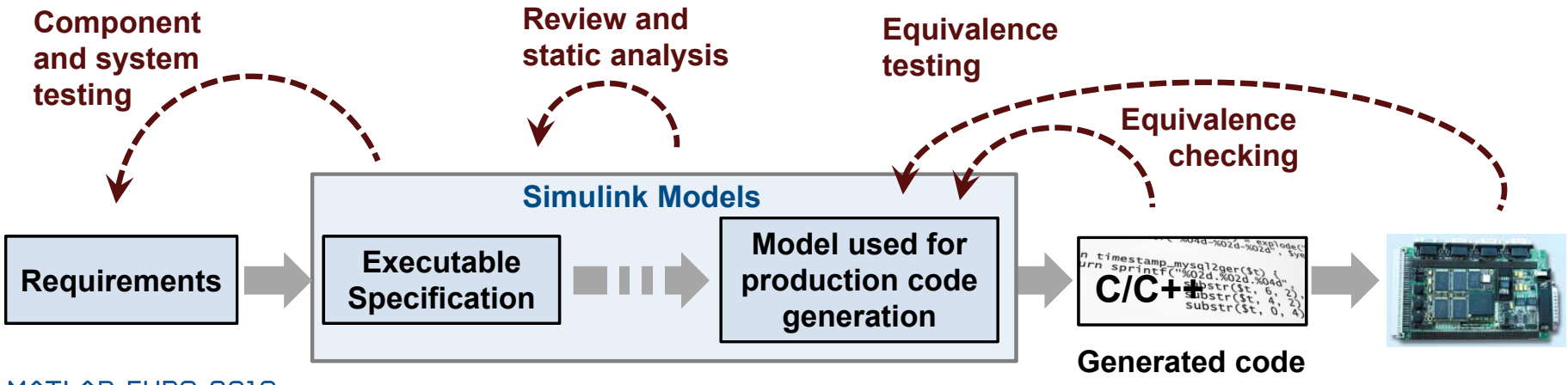
Continental Develops Electronically Controlled Air Suspension for Heavy-Duty Trucks

Verification time cut by up to 50 percent

More User Stories: www.mathworks.com/company/user_stories.html

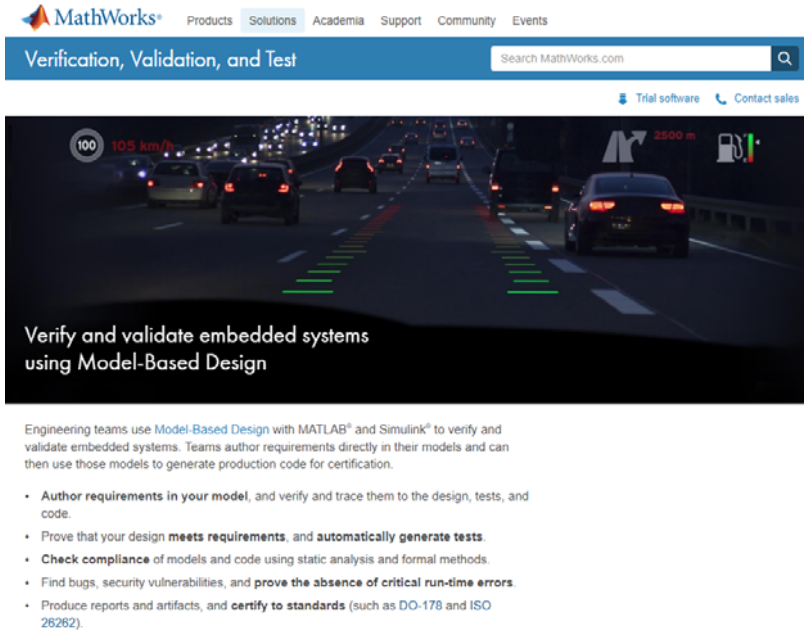
Summary

1. Author and manage requirements within Simulink
2. Find defects earlier
3. Automate static and dynamic verification
4. Reference workflow that conforms to safety standards



Learn More

Visit MathWorks Verification, Validation and Test Solution Page:
mathworks.com/solutions/verification-validation.html



Thank You!