MATLAB EXPO 2018

Automating Best Practices to Improve Design Quality

Daniel Martins

Why do 71% of Embedded Projects Fail?

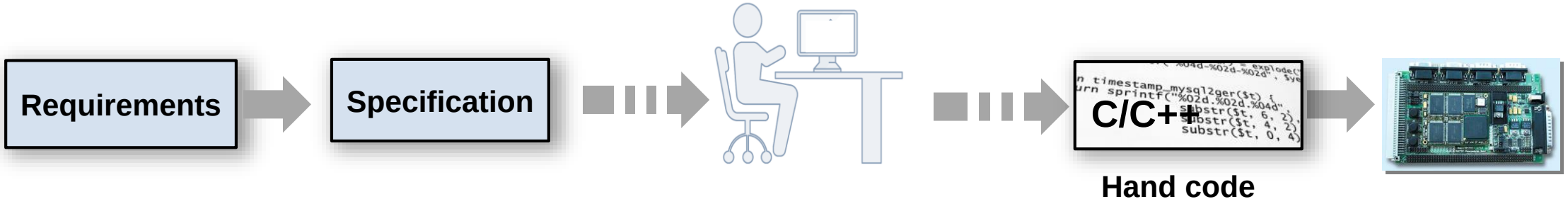
Poor Requirements Management

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

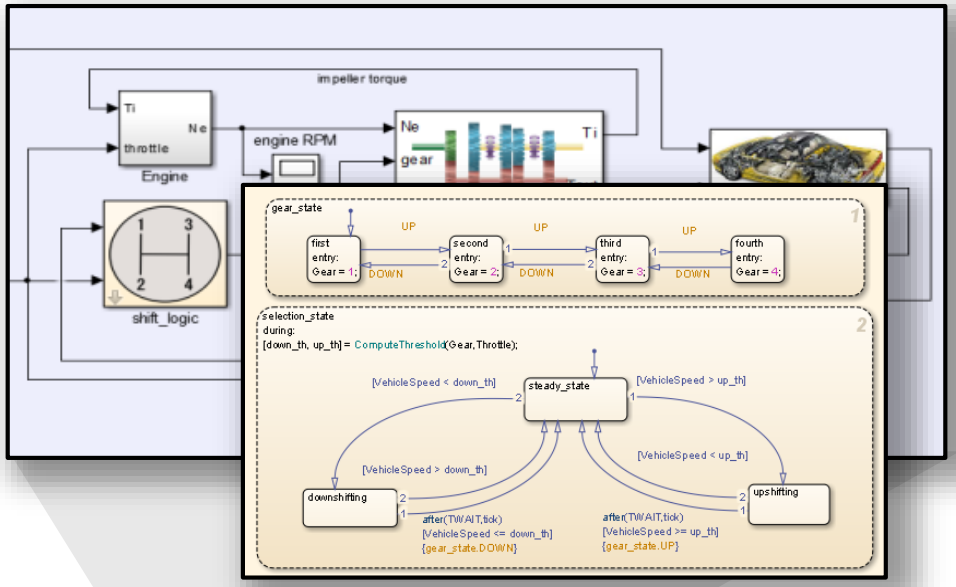
Key Takeaways

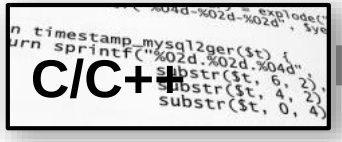
- Author, manage requirements in Simulink
- Early verification to find defects sooner
- Automate manual verification tasks
- Workflow that conforms to safety standards

Challenge with Traditional Development Process



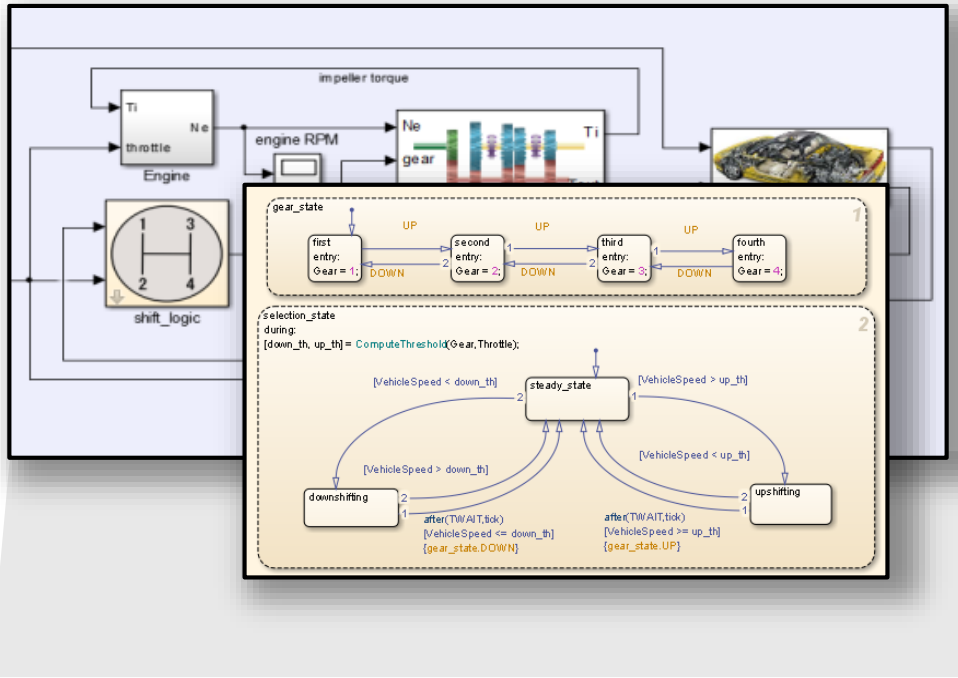
Simulink Models for Specification



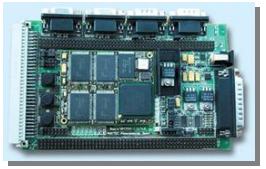
Requirements → **Executable Specification** →  → 

Hand code → 

Complete Model Based Design



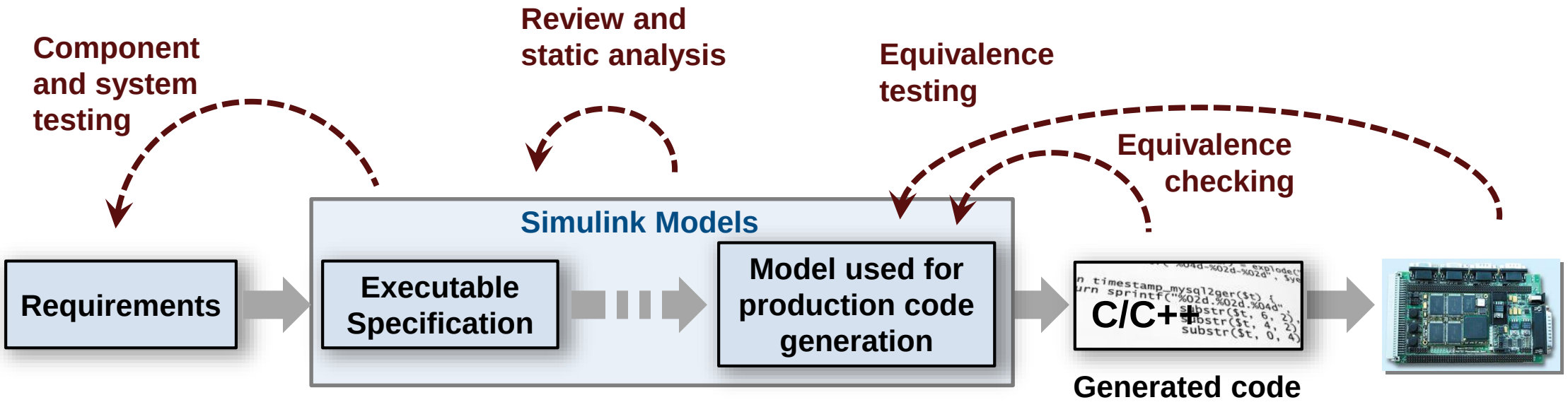
Simulink Models

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

Code Generation

Generated code

Model Based Design Verification Workflow

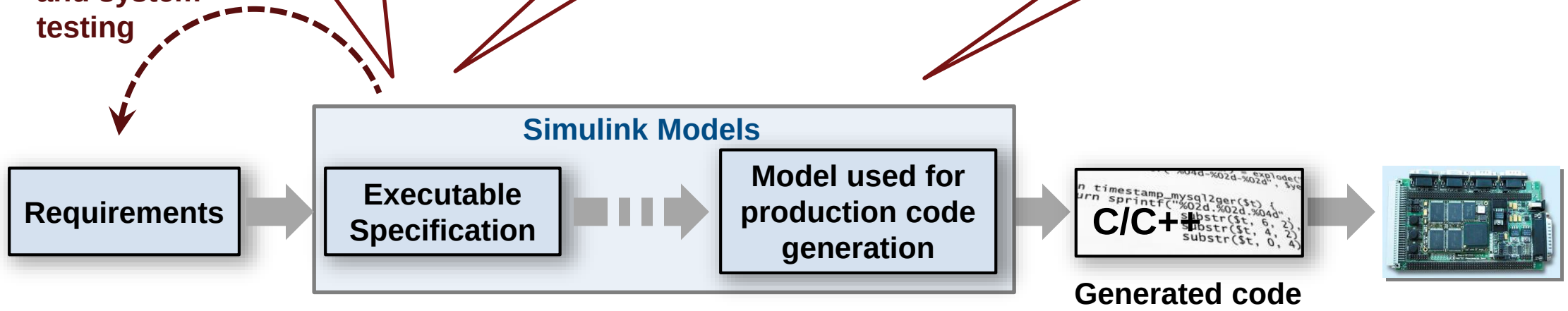


Challenges with Requirements

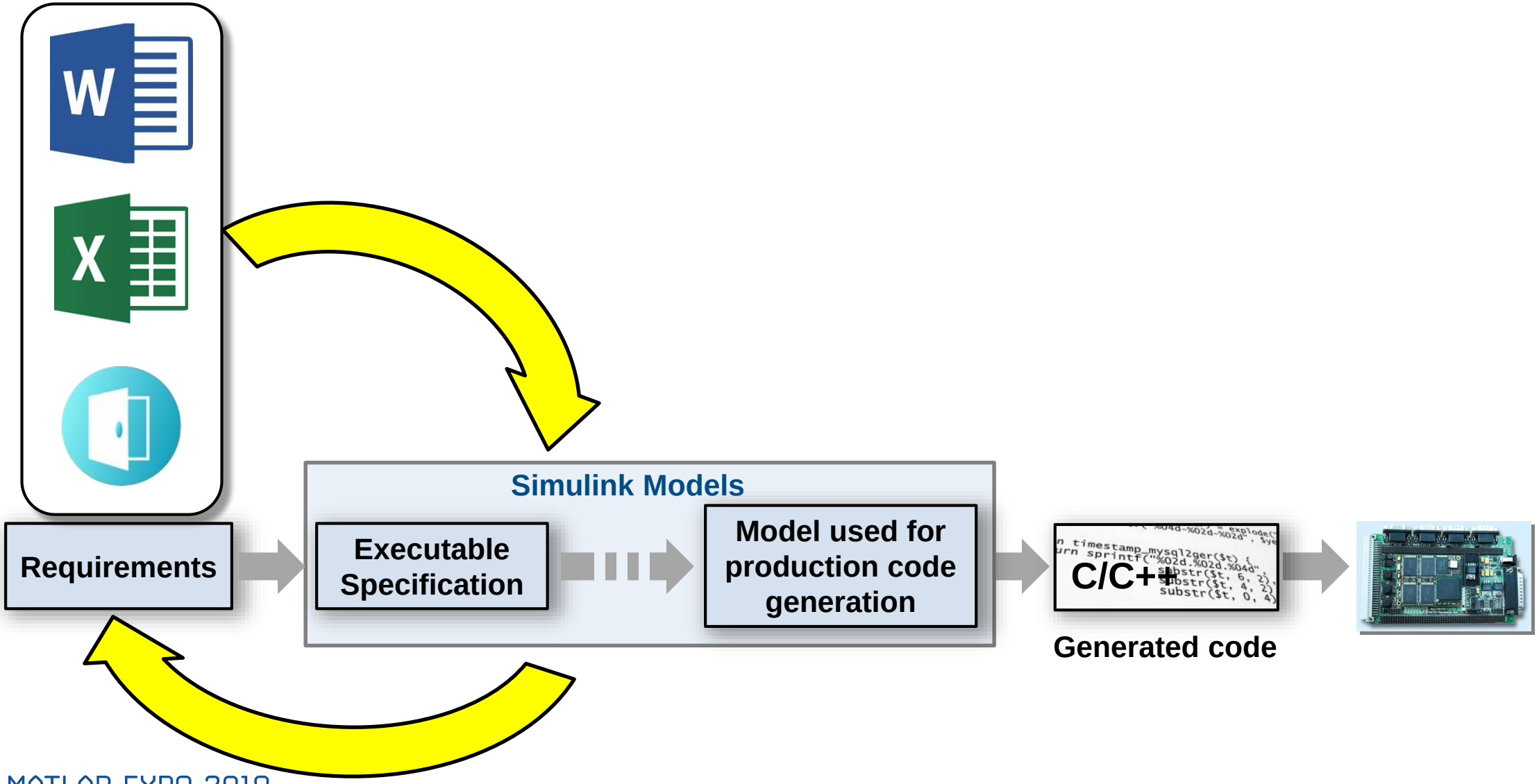
Where are requirements implemented?

Is design and requirements consistent?

How are they tested?



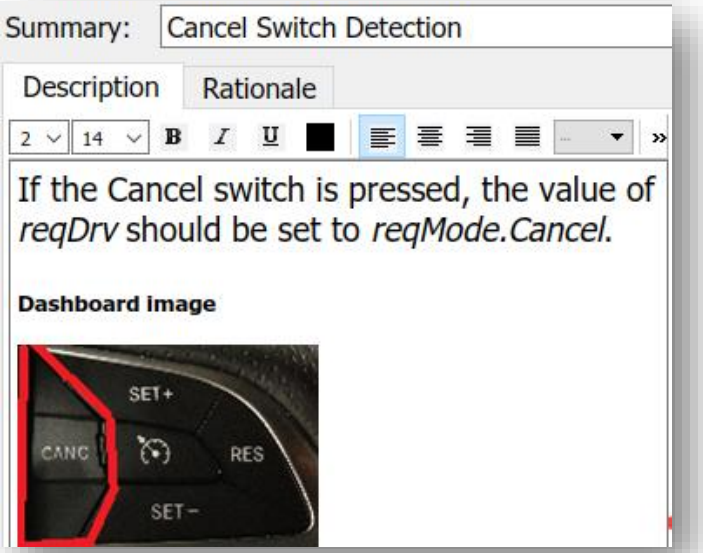
Gap Between Requirements and Design

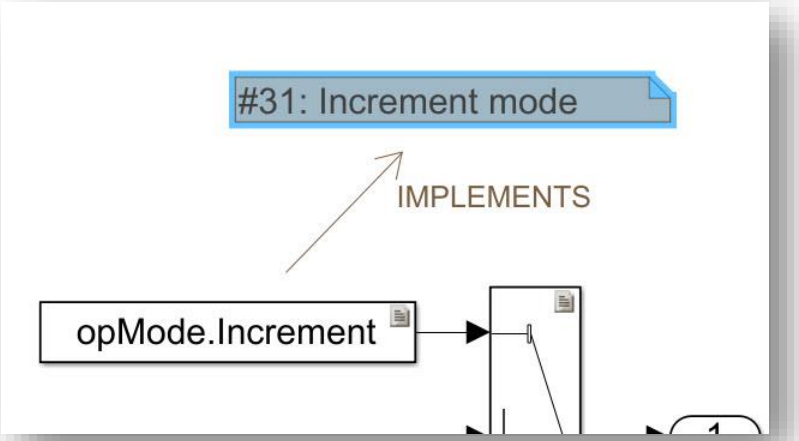


Simulink Requirements

R2017b

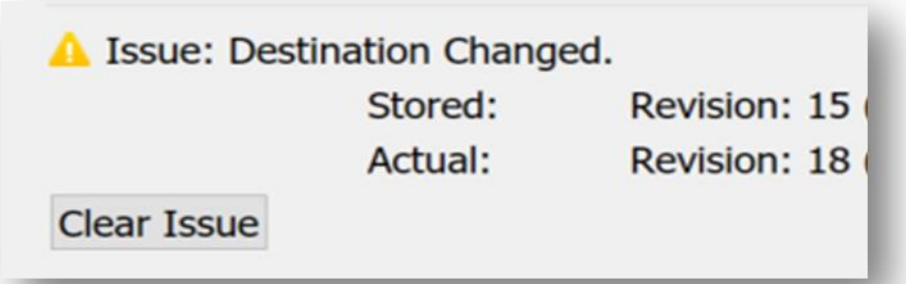
Author





Track

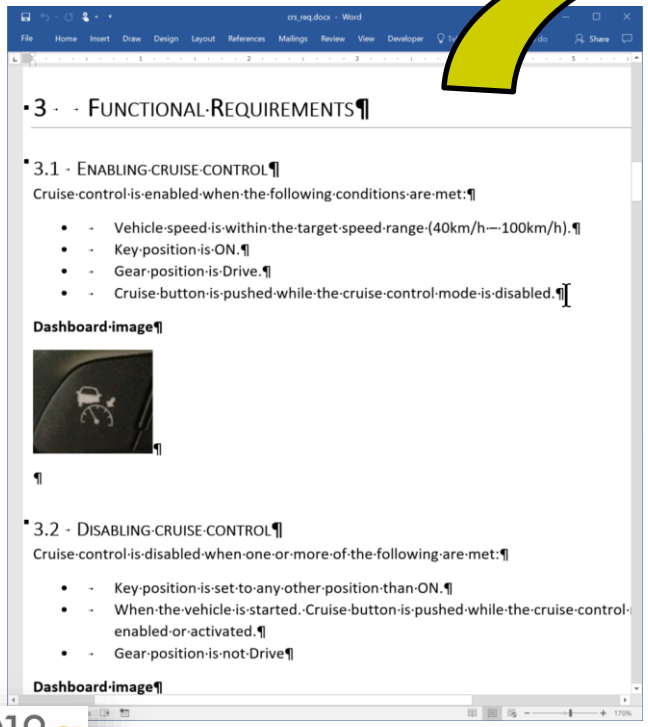
Manage



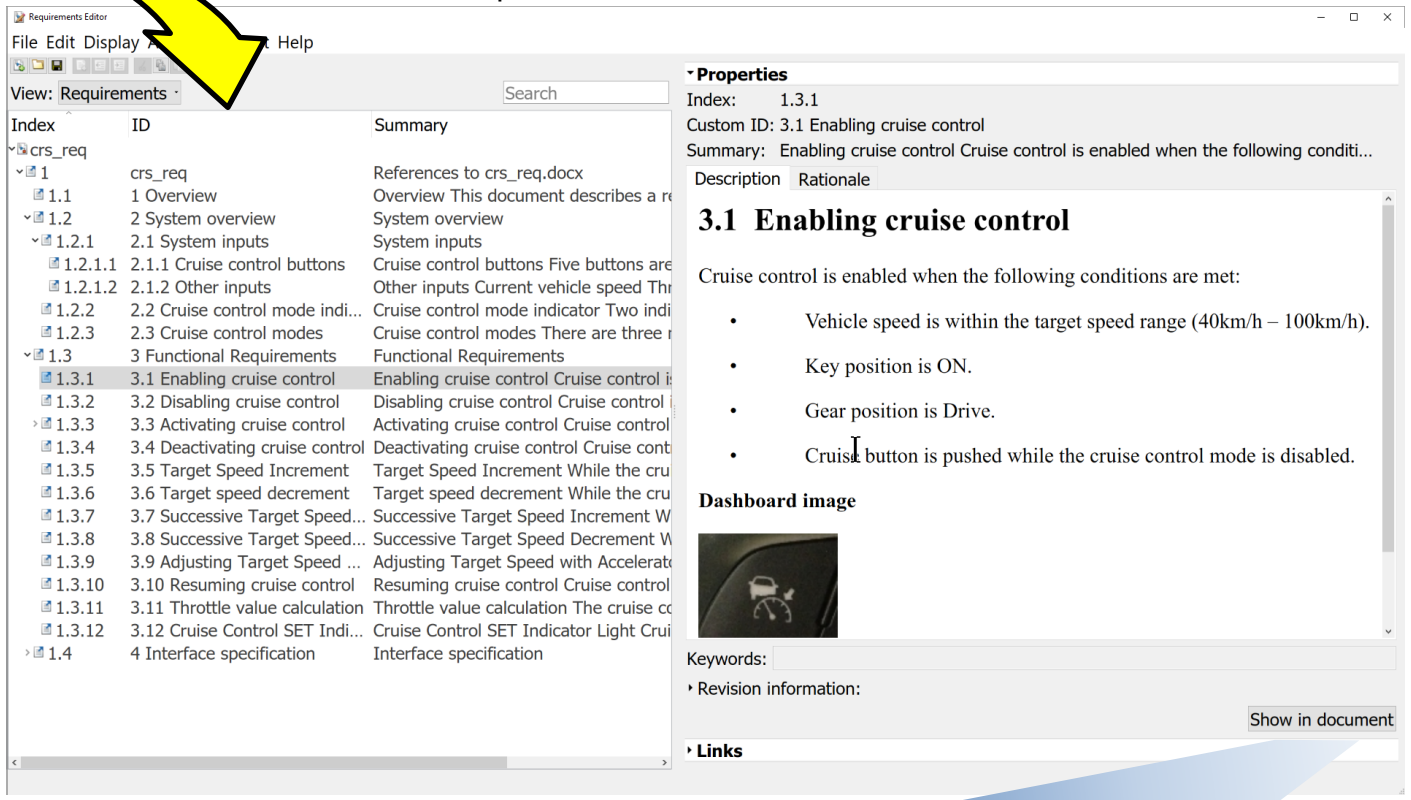
Import Requirements from External Sources

Import

Microsoft Word



Simulink Requirements Editor



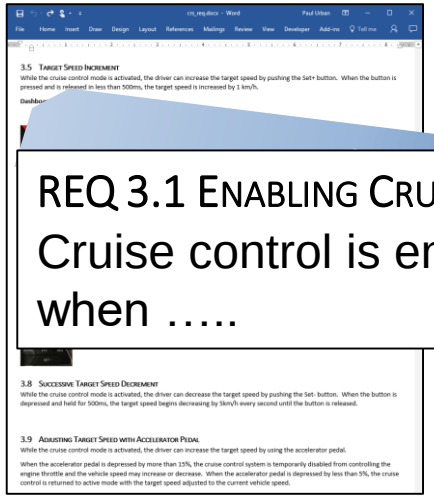
IBM Rational DOORS

R2018a

ReqIF Requirements Interchange Format

Show in document

Link Requirements, Designs and Tests

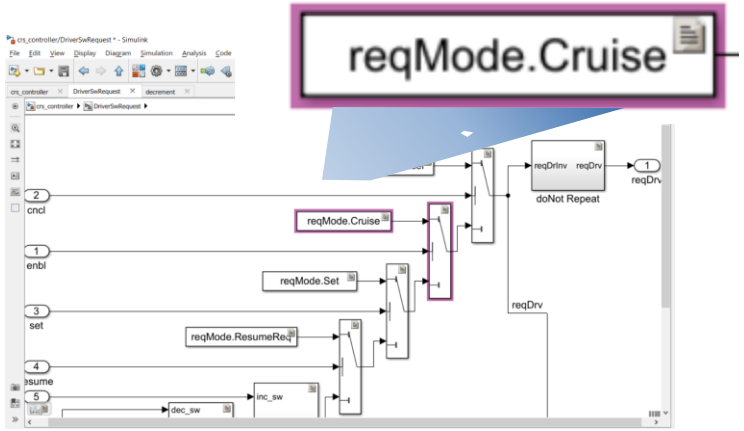


REQ 3.1 ENABLING CRUISE CONTROL
Cruise control is enabled when …..

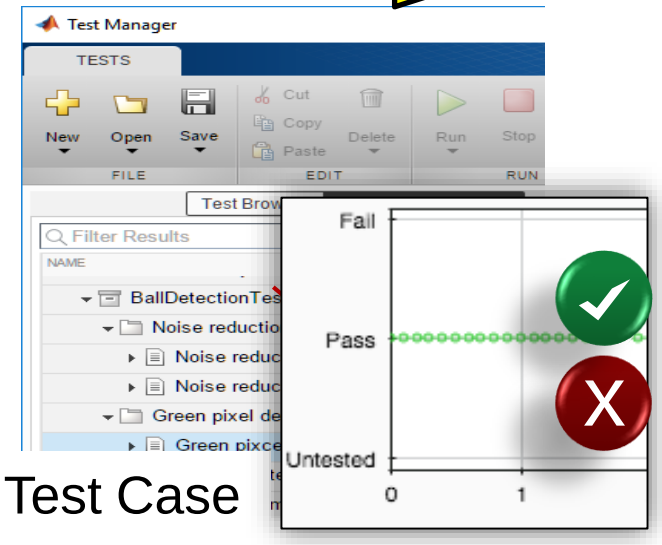
Derives

ENABLE SWITCH DETECTION
If the Enable switch is pressed ……

Implemented By

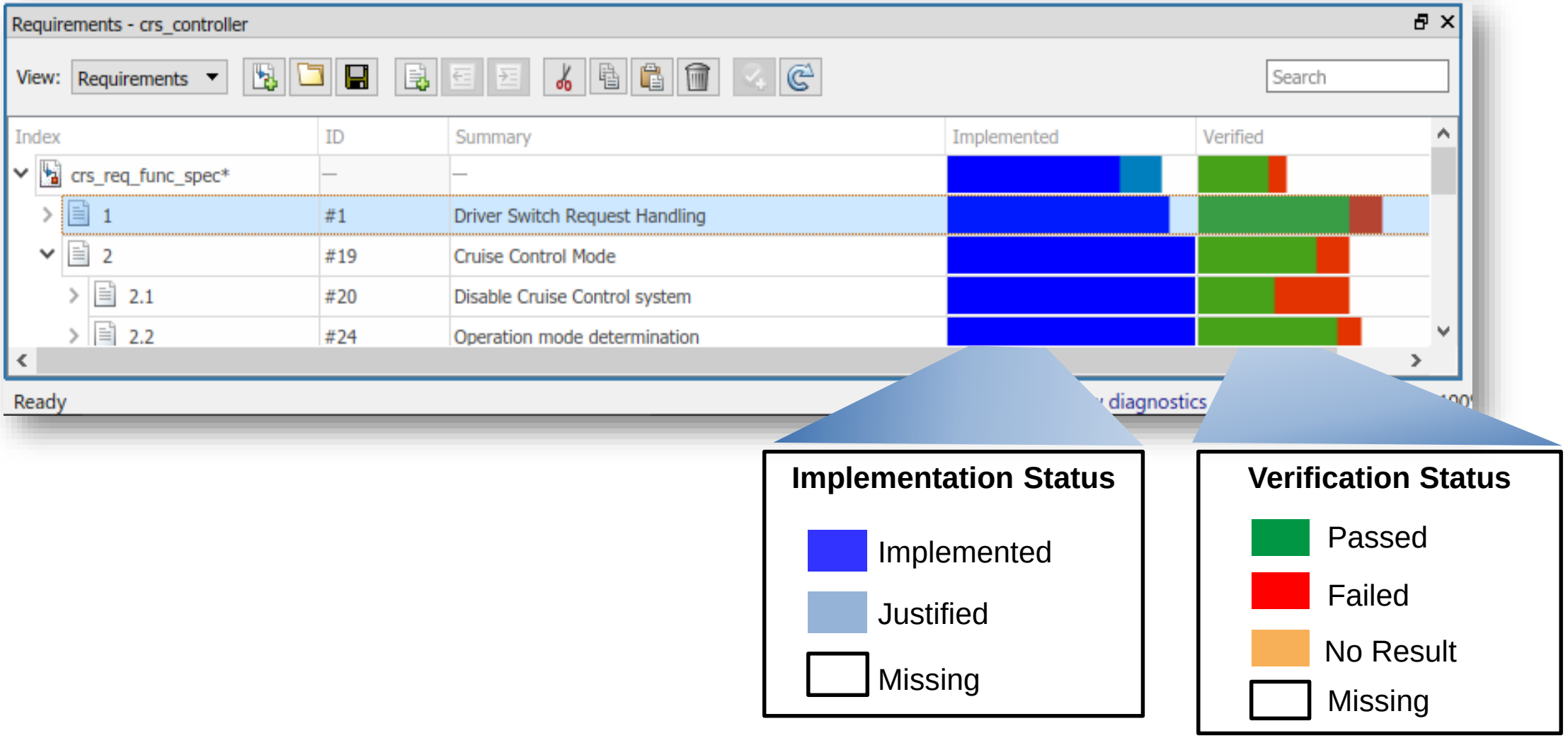


Verified By



Test Case

Track Implementation and Verification



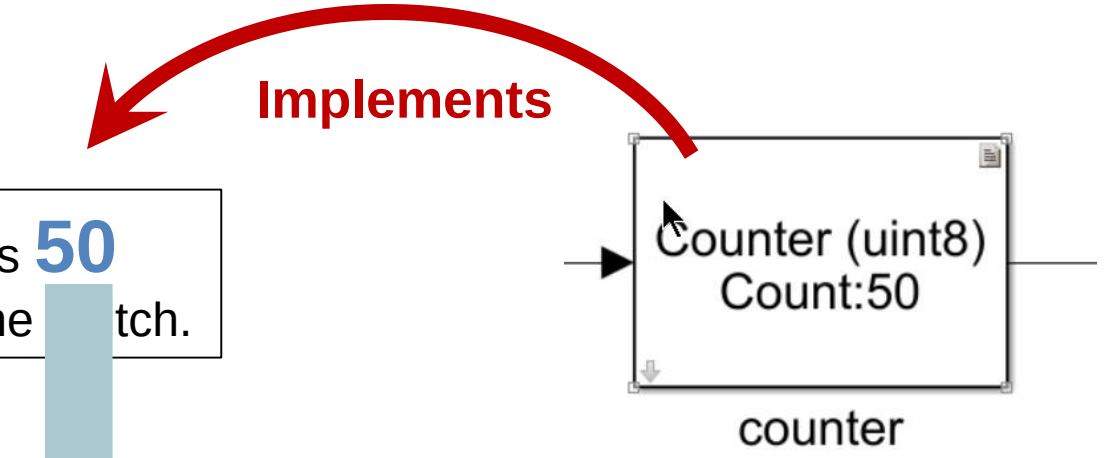
Respond to Change

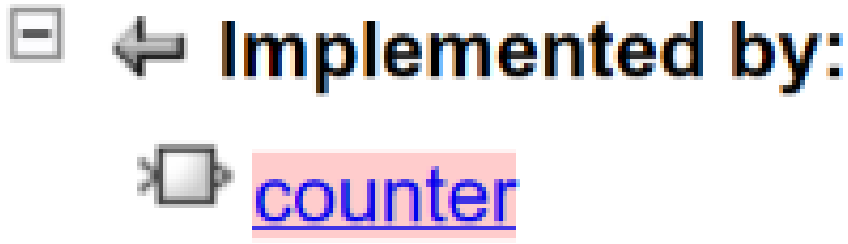
Original Requirement

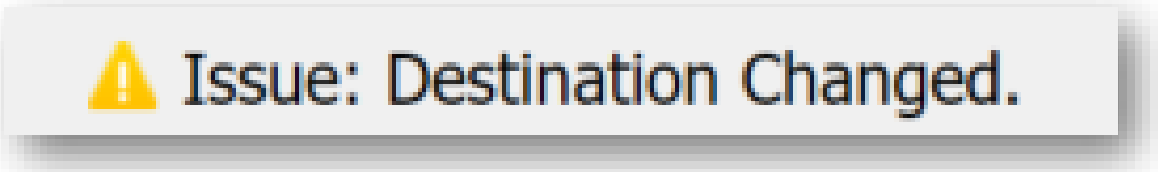
If the switch is pressed and the counter reaches **50** then it shall be recognized as a long press of the tch.

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

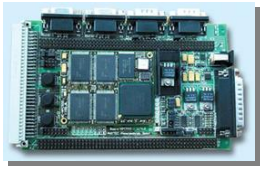
Is the design built right?

Is it too complex?

Is it ready for code generation?

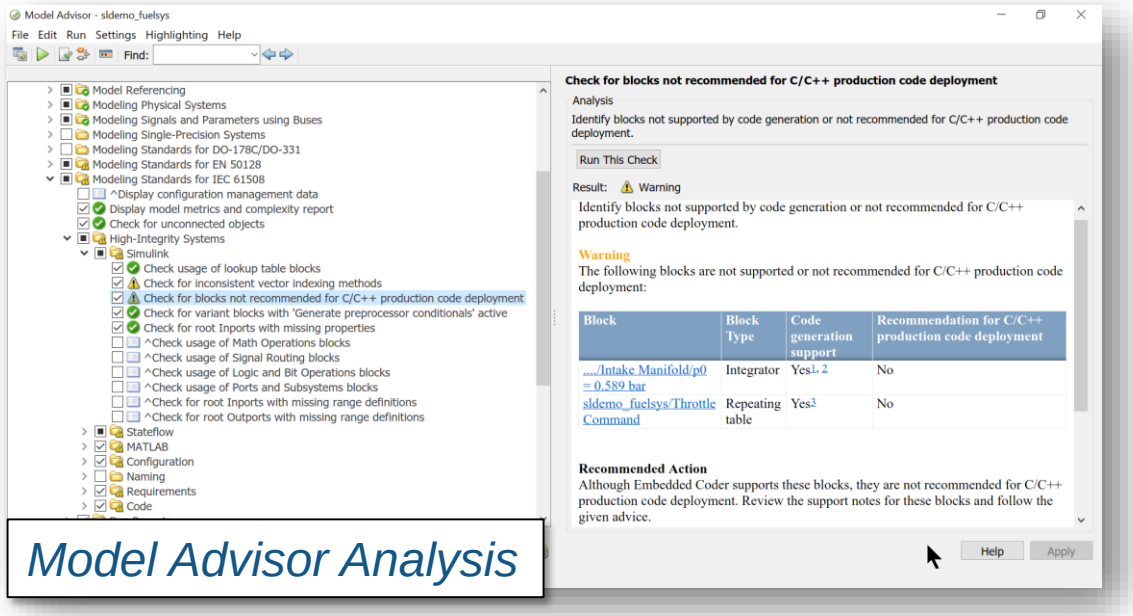
Review and static analysis

Simulink Models

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

Generated code

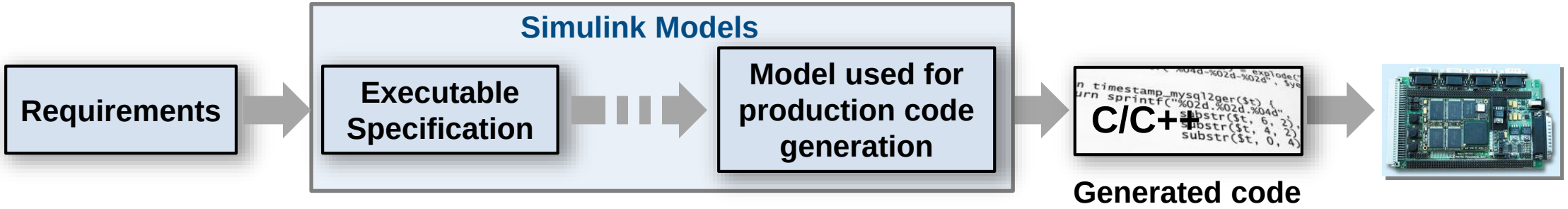
Automate verification with static analysis



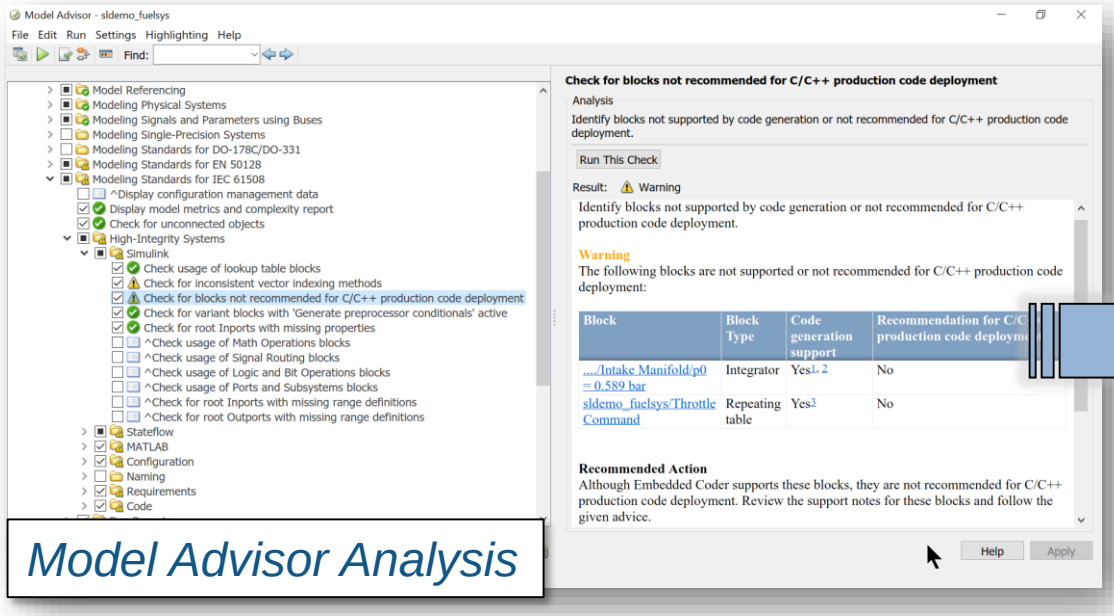
Model Advisor Analysis

Check for:

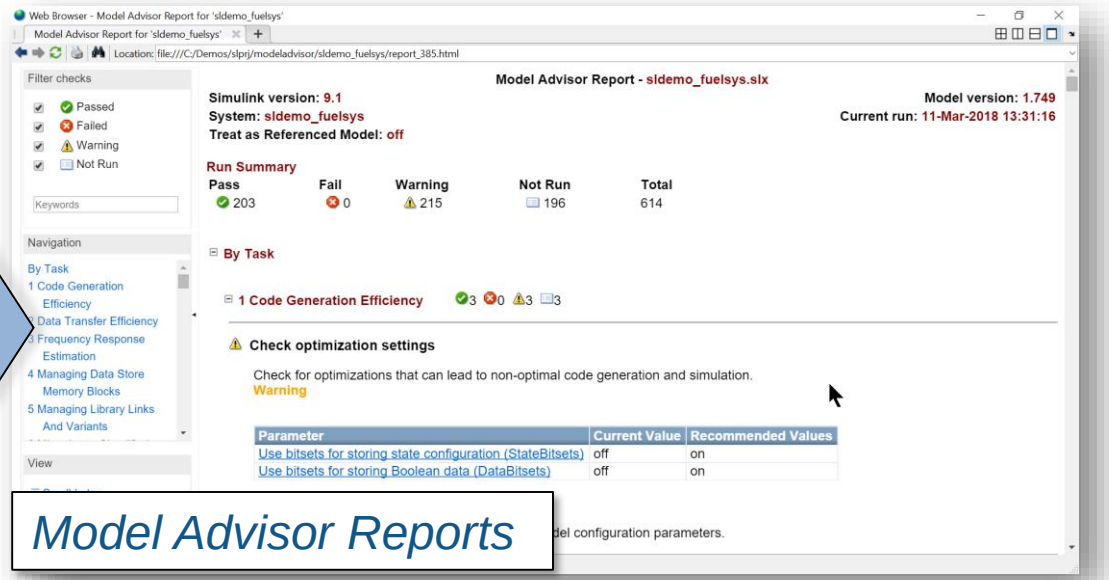
- Readability and Semantics
- Performance and Efficiency
- Clones
- And more……



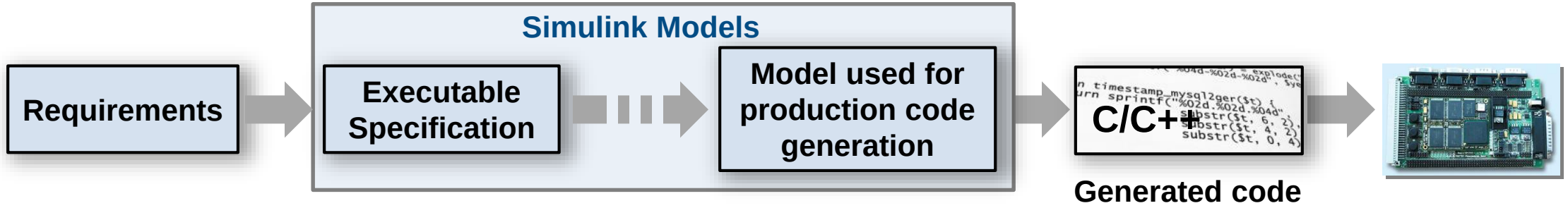
Generate reports for reviews and documentation



Model Advisor Analysis

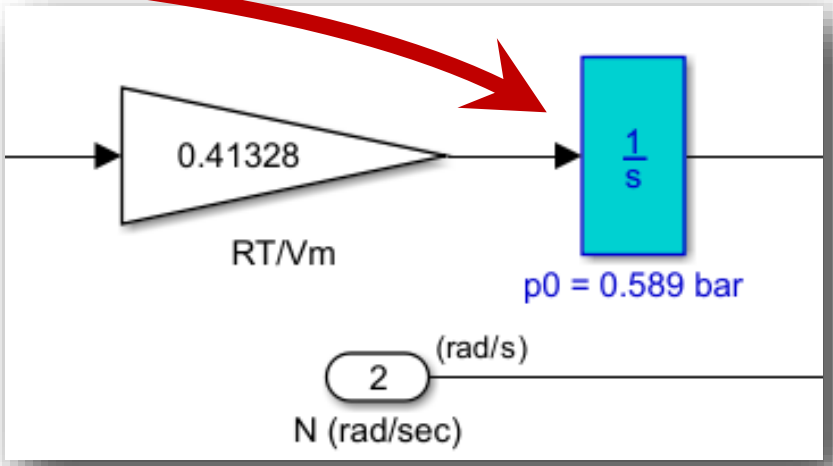


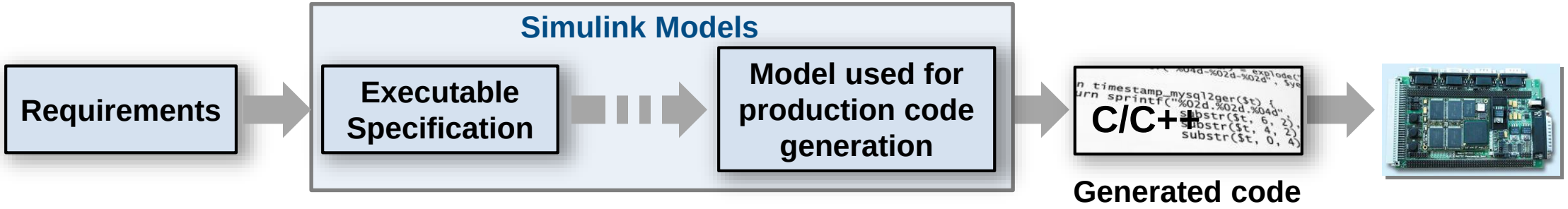
Model Advisor Reports



Navigate to Problematic Blocks

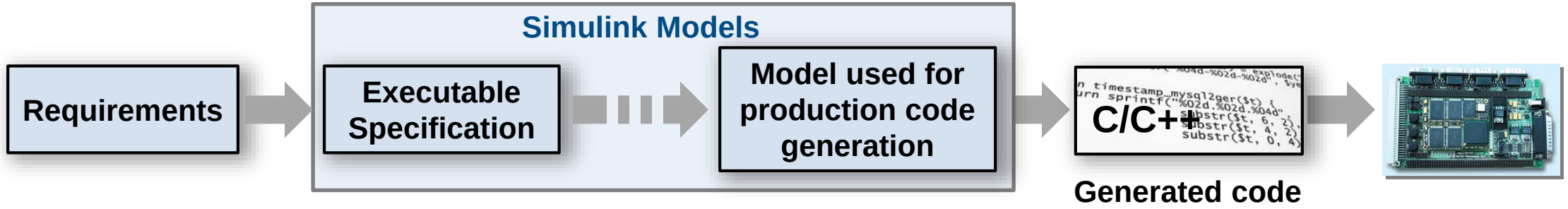
Block	Block Type	Code generation support	Recommendation for C/C++ production code deployment
..../Intake Manifold/p0 = 0.589 bar	Integrator	Yes[1], [2]	No
sldemo_fuelsys/Throttle Command	Repeating table	Yes[3]	No





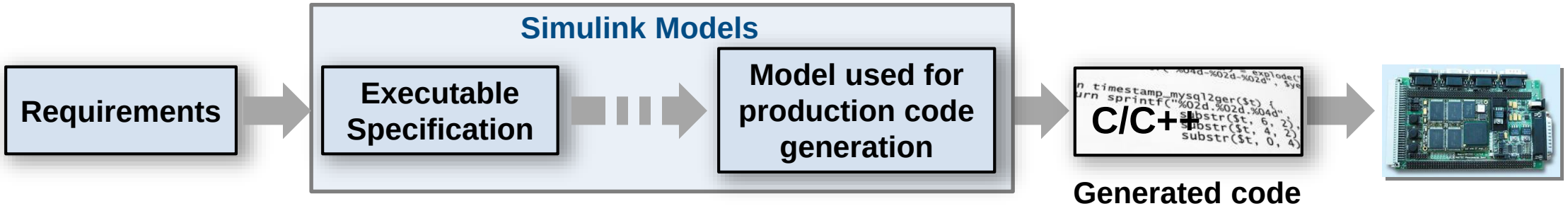
Guidance Provided to Address Issues or Automatically Correct

Recommended Action
Although Embedded Coder supports these blocks, they are not recommended for C/C++ production code deployment. Review the support notes for these blocks and follow the given advice.

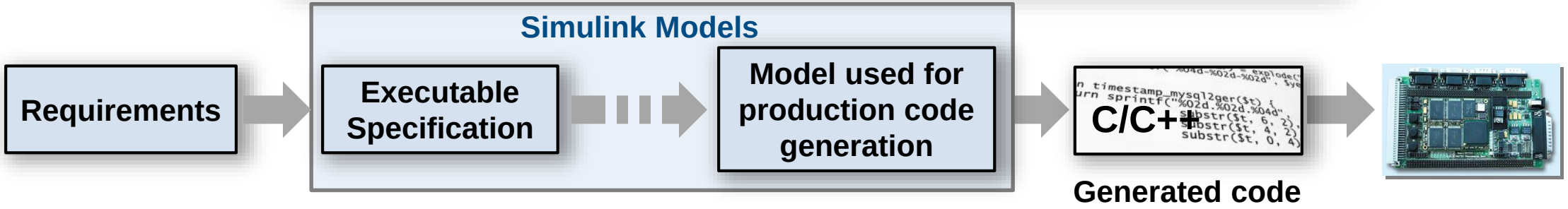


Built in checks for industry standards and guidelines

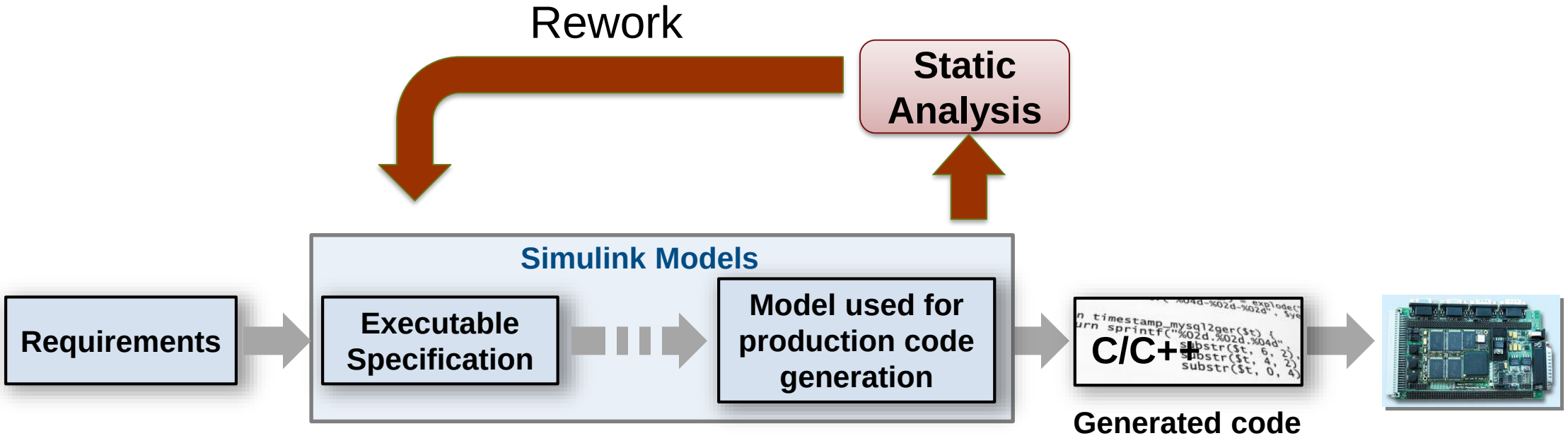
- DO-178/DO-331
- ISO 26262
- IEC 61508
- IEC 62304
- EN 50128
- MISRA C:2012
- CERT C, CWE, ISO/IEC TS 17961
- MAAB (MathWorks Automotive Advisory Board)
- JMAAB (Japan MATLAB Automotive Advisory Board)



Configure and customize analysis

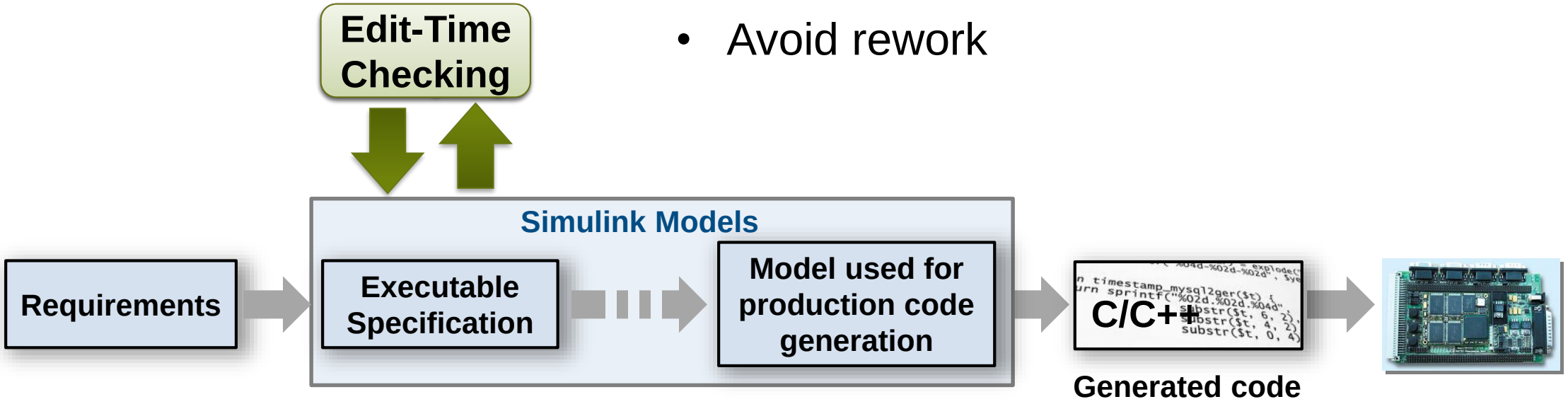


Checks for standards and guidelines are often performed late

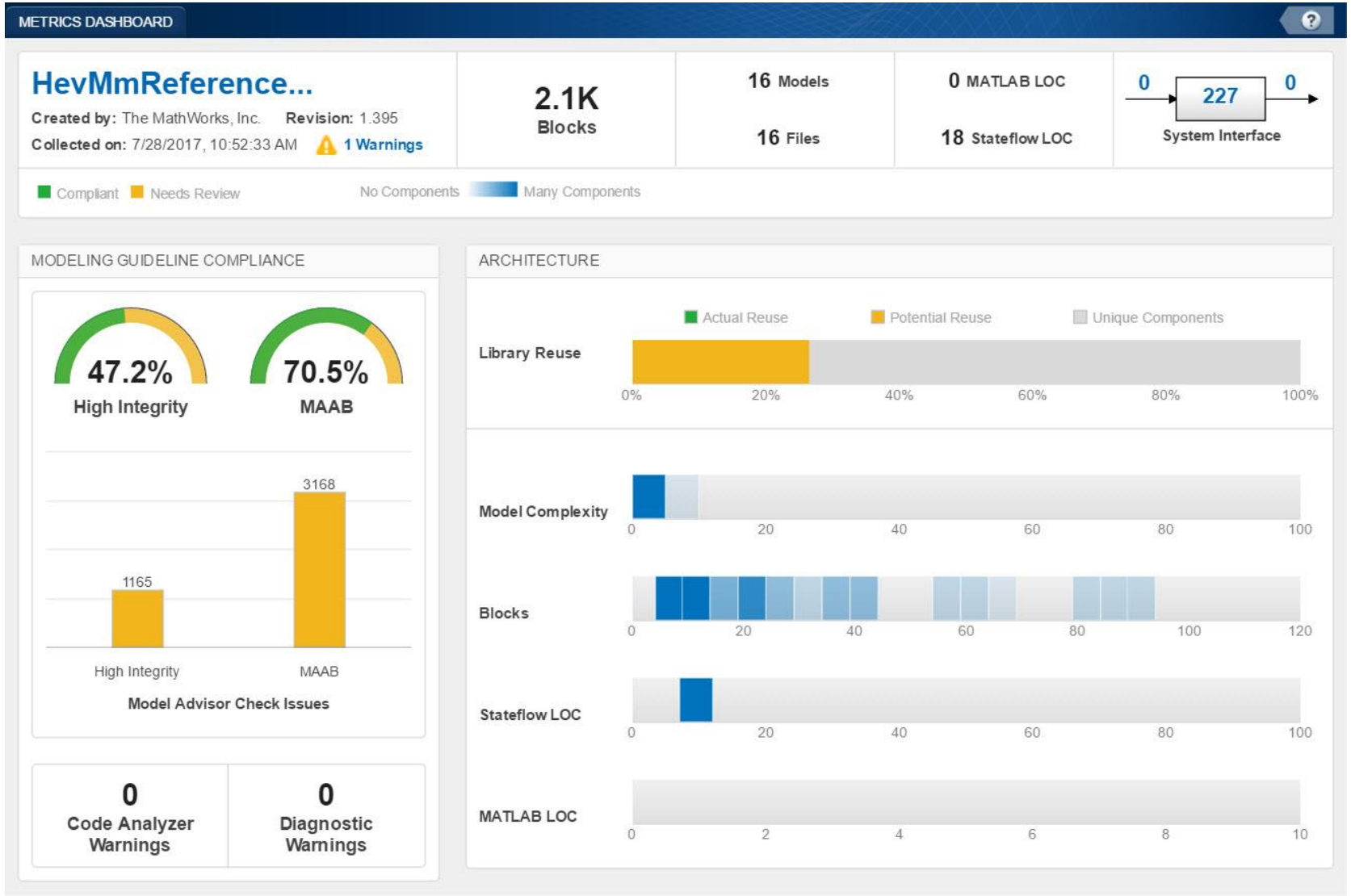


Shift Verification Earlier With Edit-Time Checking

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



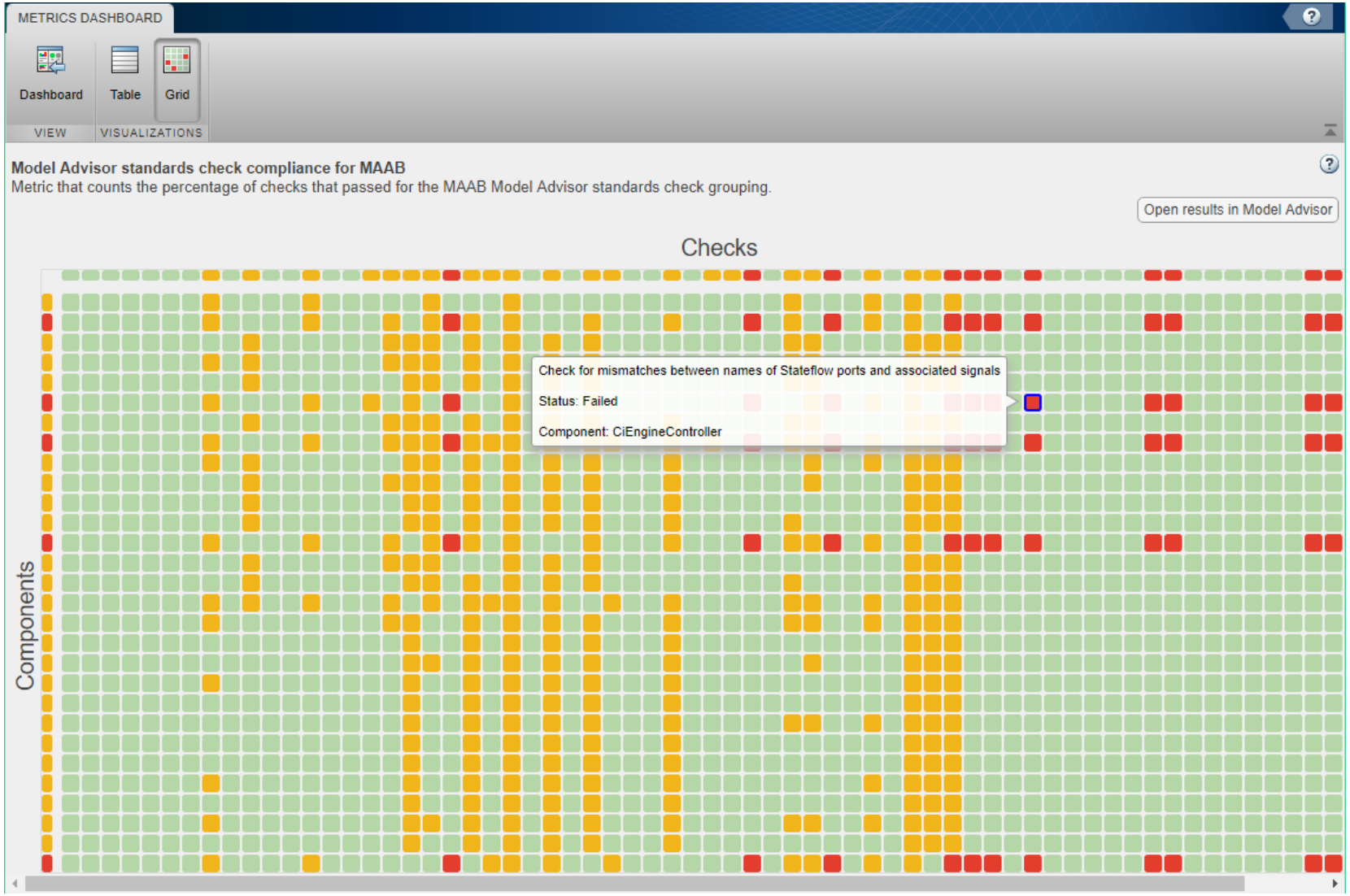
Assess Quality with Metrics Dashboard



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

Grid Visualization for Metrics

R2018a

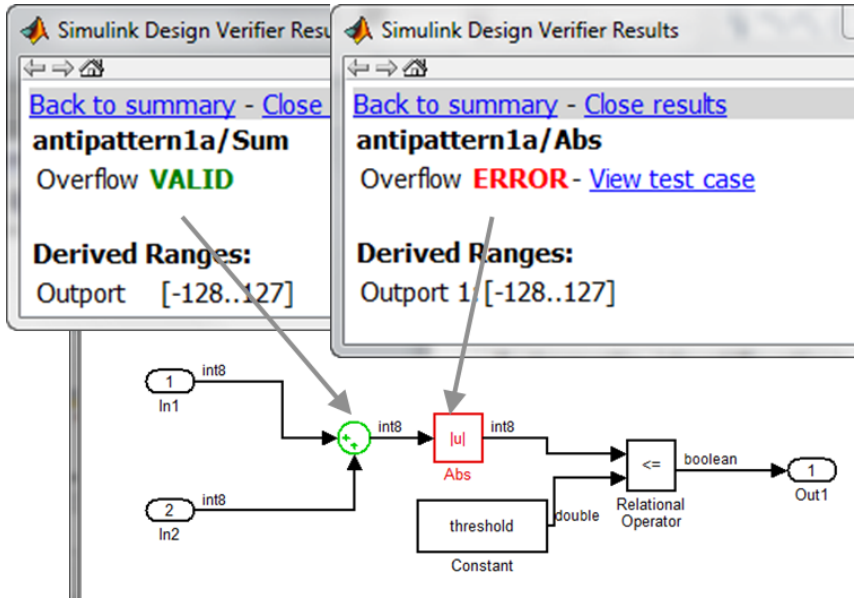


- Visualize Standards Check Compliance
  - Find Issues
  - Identify patterns
  - See hot spots

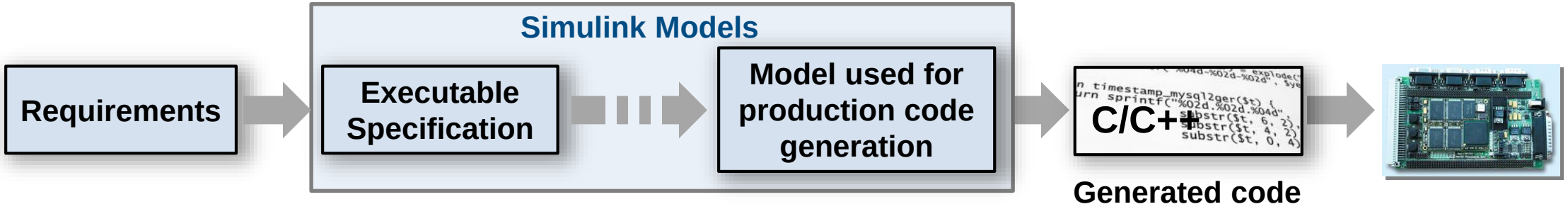
Legend:

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

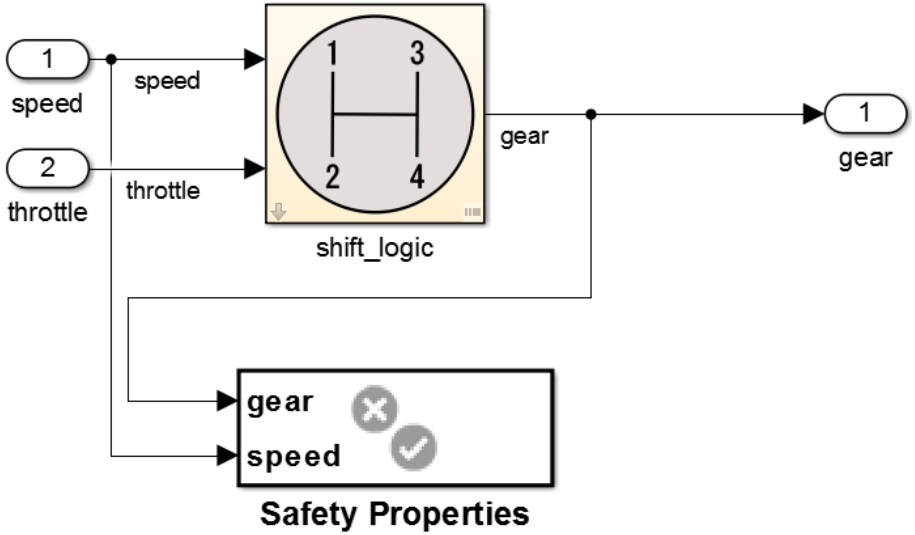
Detect Design Errors with Formal Methods



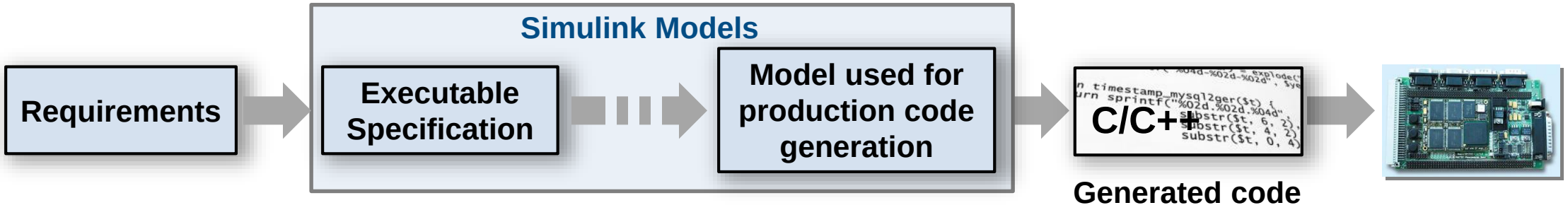
- Find run-time design errors:
  - Integer overflow
  - Dead Logic
  - Division by zero
  - Array out-of-bounds
  - Range violations
- Generate counter example to reproduce error



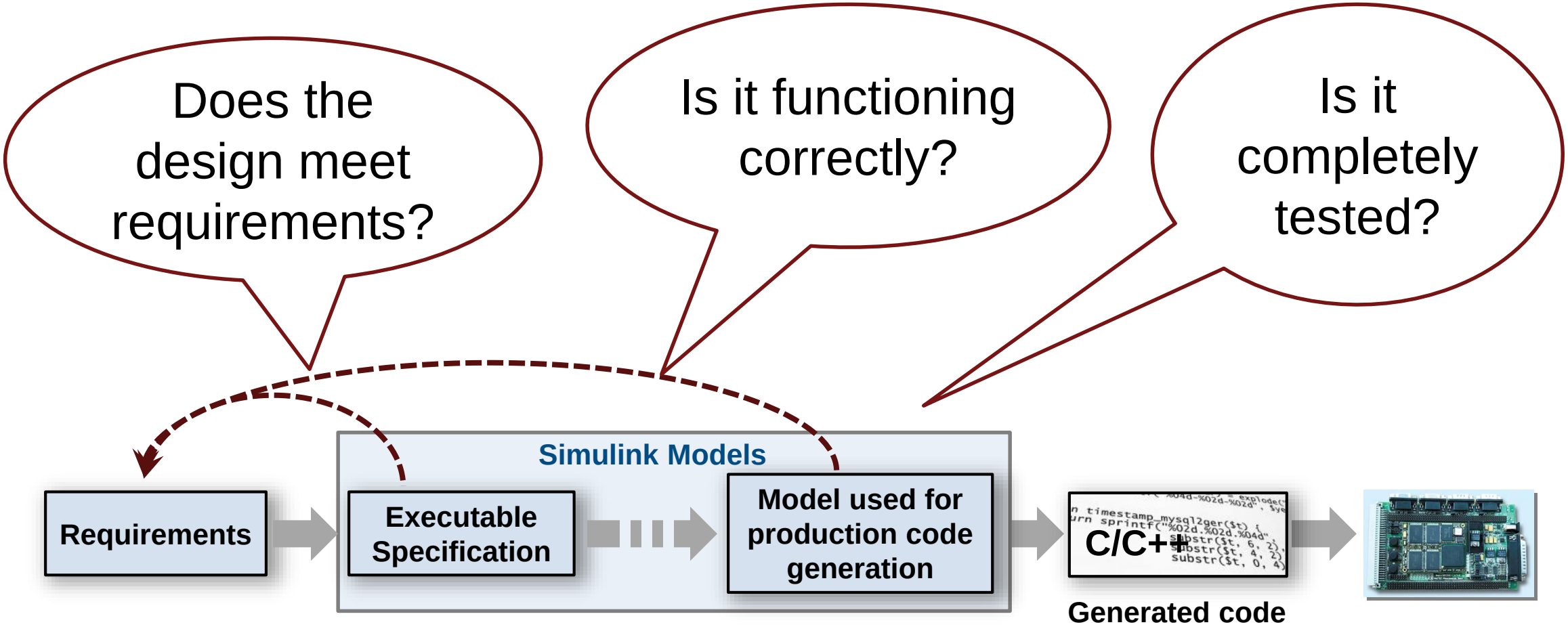
Prove That Design Meets Requirements



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



Functional Testing



Systematic Functional Testing

Test Case

Inputs

MAT file (input)

Group 1
Signal 1

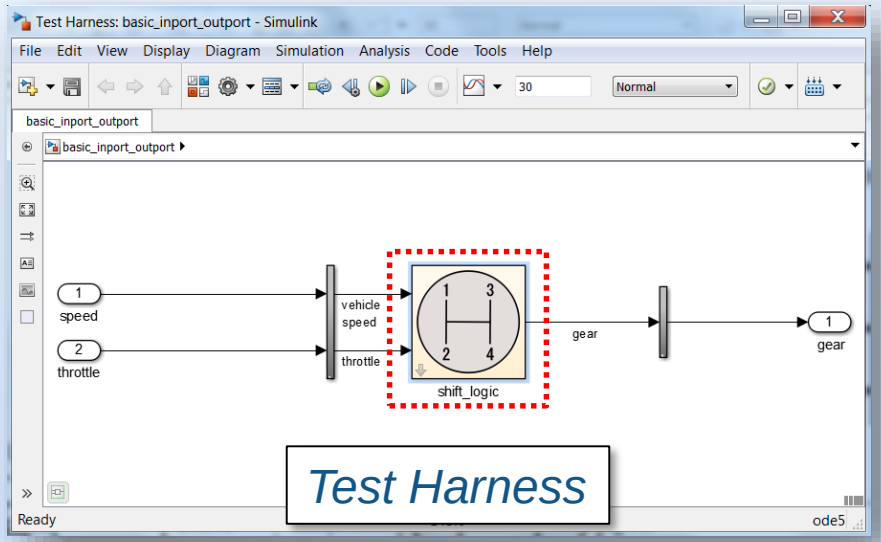
Signal Builder

1
2 Output
3

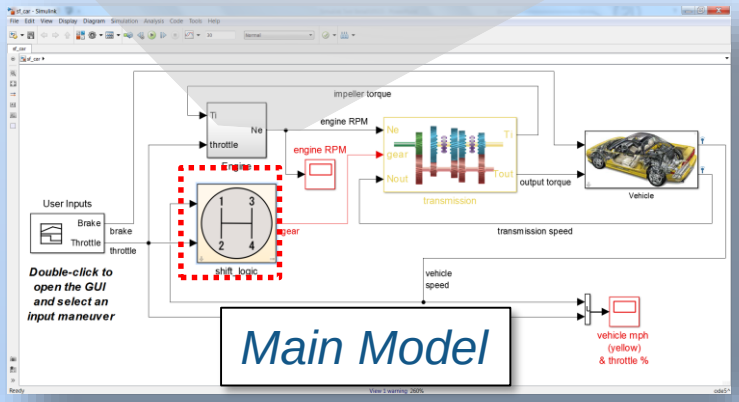
Test Sequence

R2017b

Excel file (input)



Test Harness



Main Model

Assessments

MAT file (baseline)

```
function customCriteria
Perform custom criteria
test.verifyThat(test.sl
```

MATLAB Unit Test

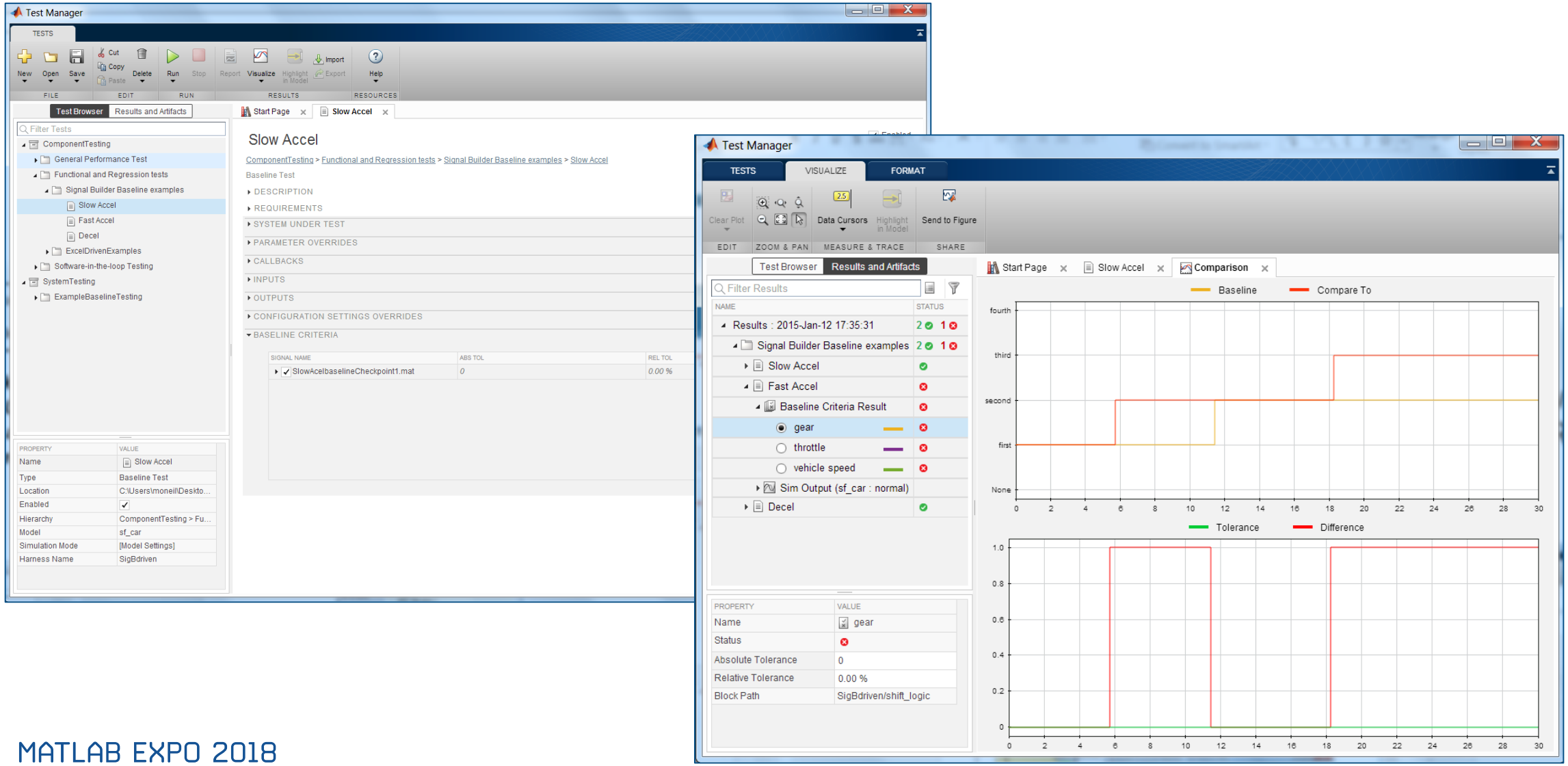
1
Input
3

Test Assessment

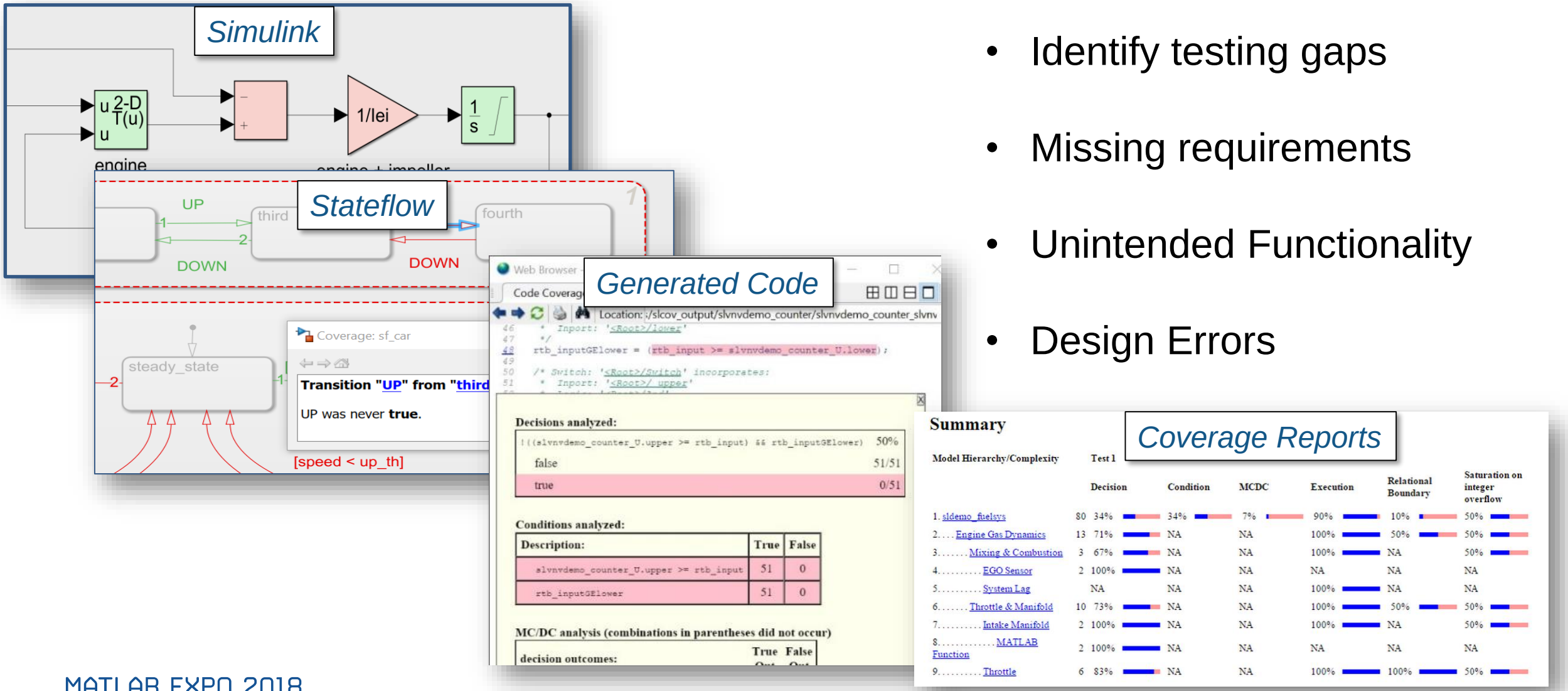
R2017b

Excel file (baseline)

Manage Testing and Test Results

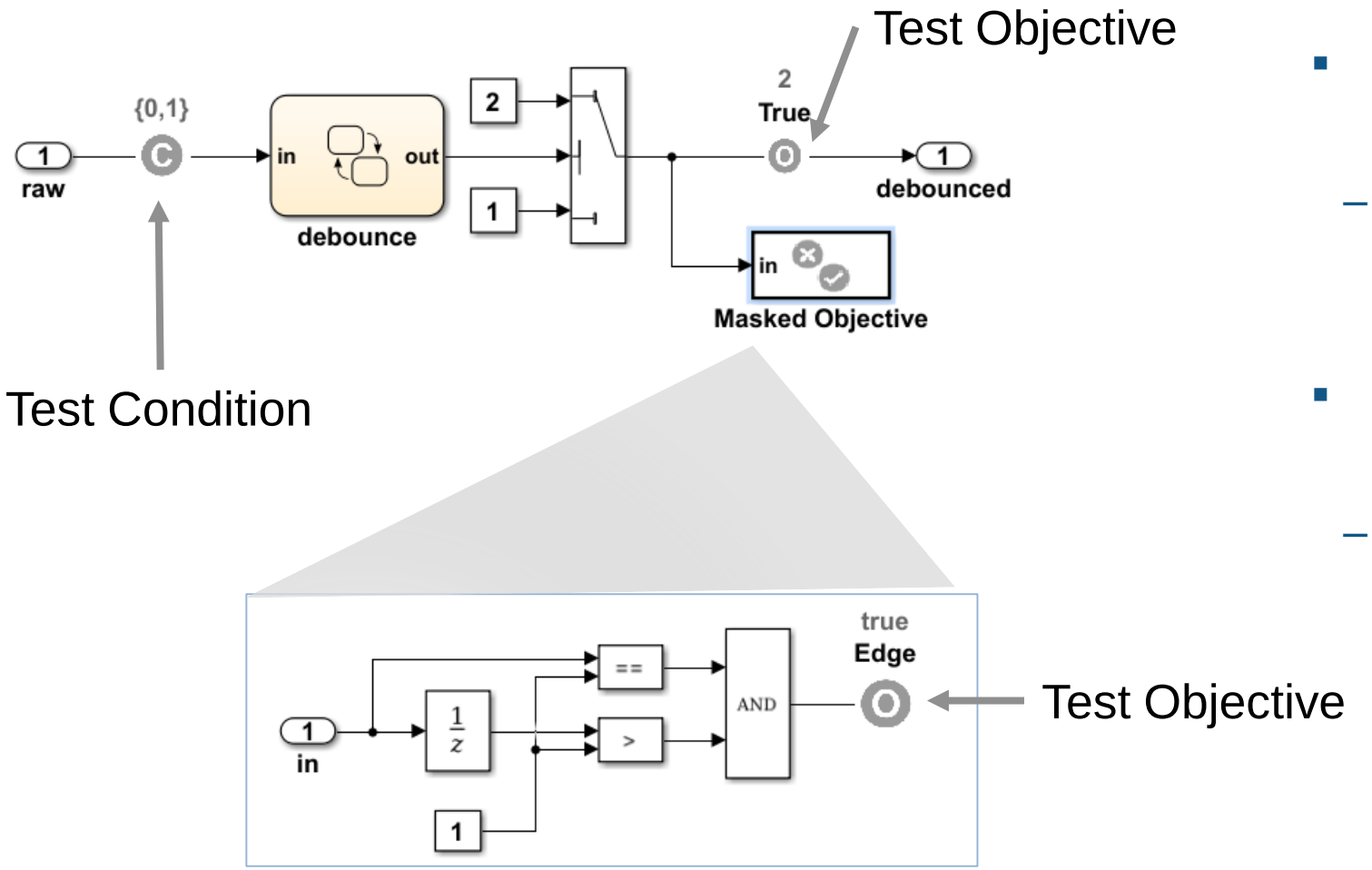


Coverage Analysis to Measure Testing



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

Test Case Generation for Functional Testing



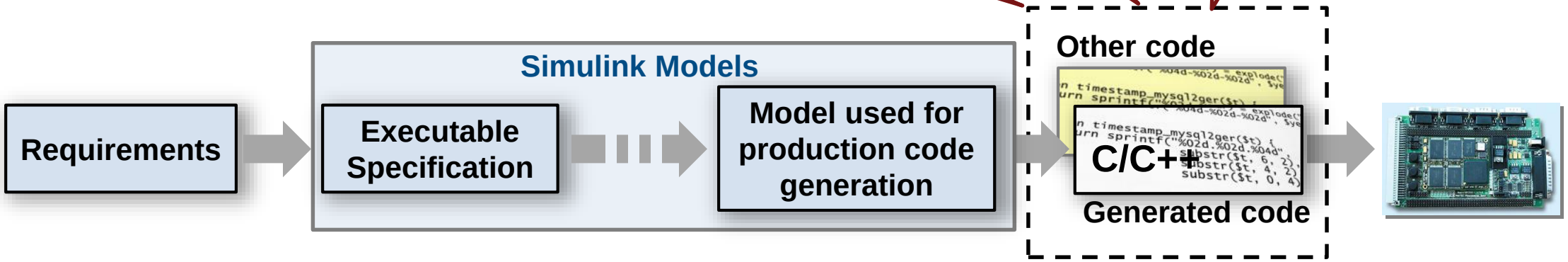
- Specify functional test objectives
  - Define custom objectives that signals must satisfy in test cases
- Specify functional test conditions
  - Define constraints on signal values to constrain test generator

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?



The Generated Code is integrated with Other Code (Handwritten)

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

Green: reliable
safe pointer access

Red: faulty
out of bounds error

Gray: dead
unreachable code

Orange: unproven
may be unsafe for some conditions

Purple: violation
MISRA-C/C++ or JSF++ code rules

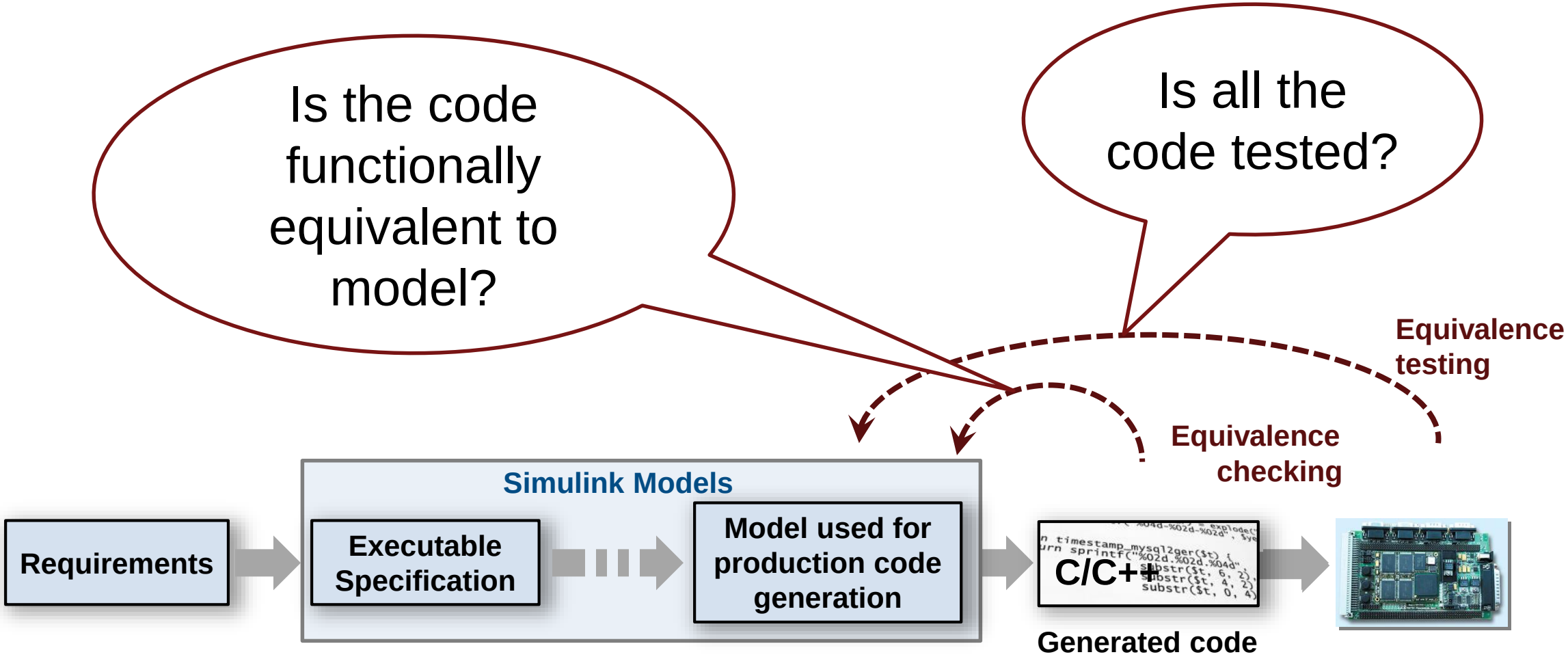
Range data
tool tip

```
static void pointer_arithmetic (void) {
    int array[100];
    int *p = array;
    int i;

    for (i = 0; i < 100; i++) {
      *p = 0;
      p++;
    }

  if (get bus status() > 0) {
    if (get oil pressure() > 0) {
      *p = 5;
    } else {
      i++;
    }
  }

  i = get bus status();

  if (i >= 0) {
    *(p - i) = 10;
  }
}
```

variable 'I' (int32): [0 .. 99]
assignment of 'I' (int32): [1 .. 100]

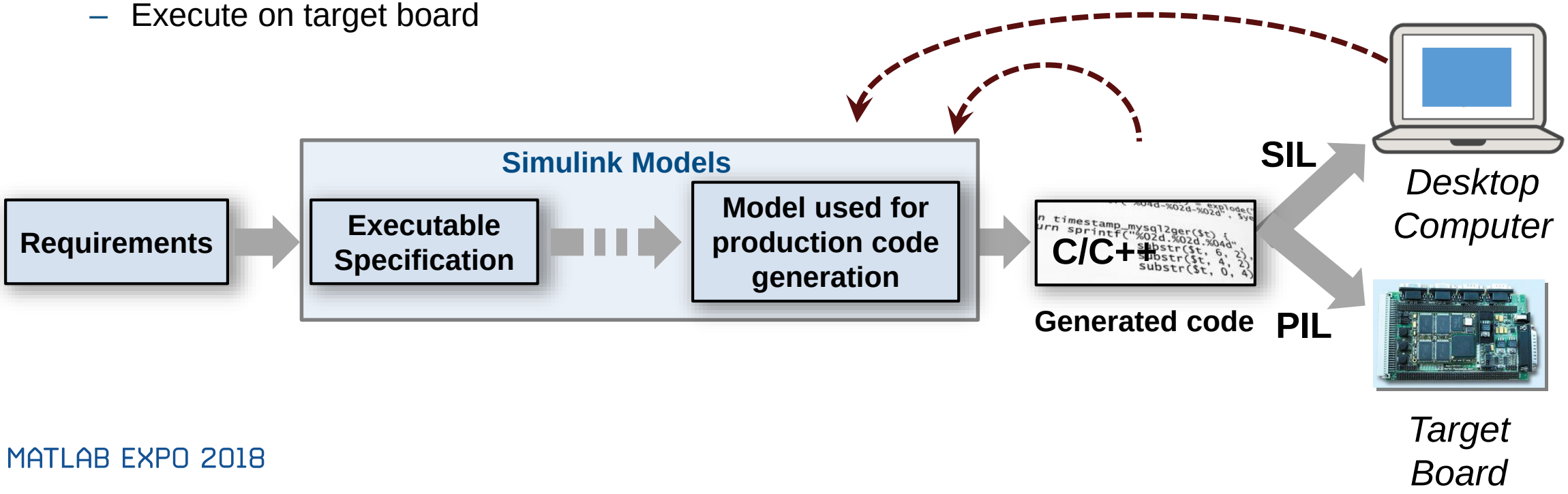
Results from Polyspace Code Prover

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
- Collect code coverage



Qualify tools with IEC Certification Kit and DO Qualification Kit

- Qualify code generation and verification products
- 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

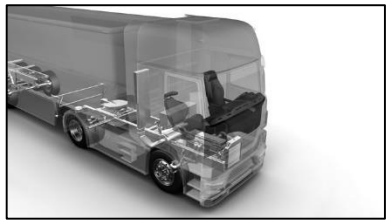
Customer References and Applications

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 manual verification tasks
4. Reference workflow that conforms to safety standards