MATLAB EXPO

Master Class: Developing Safe and Secure Embedded Software from Desktop to Cloud Using Model-Based Design

Gaurav Ahuja, MathWorks

Rajat Arora, MathWorks

MathWorks

Tooling and approaches must address today's challenges and trends

Aerospace and Defense

Automotive

Communications

Complex multi-domain systems, software-defined and autonomous, model-based and data-driven

Comms infrastructure, plus all types of connected systems across industries

Railway Systems

Energy Production

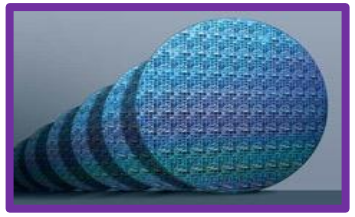
Electronics

Modernization, often on legacy platforms, becoming data-centric for optimization and maintenance

Wide range of compute platforms, many kinds of HW/SW integration

Process Industries

Industrial Machinery



Semiconductors

Software and Internet

Big Data, Agile, DevOps, integration with IT systems

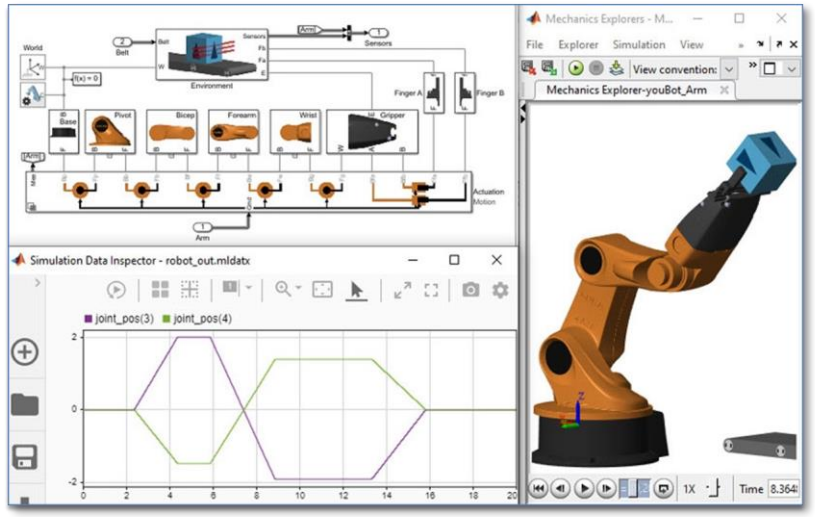
Financial Services

There are **three key pieces** to Model-Based Design

Modeling & Simulation

Test & Verification

Code Generation



Conditions analyzed

Description	True	False
Condition 1, "alt>10000"	4 U1.1	185 U1.1
Condition 2, "anomaly"	0	4 U1.1

MC/DC analysis (combinations in parentheses did not occur)

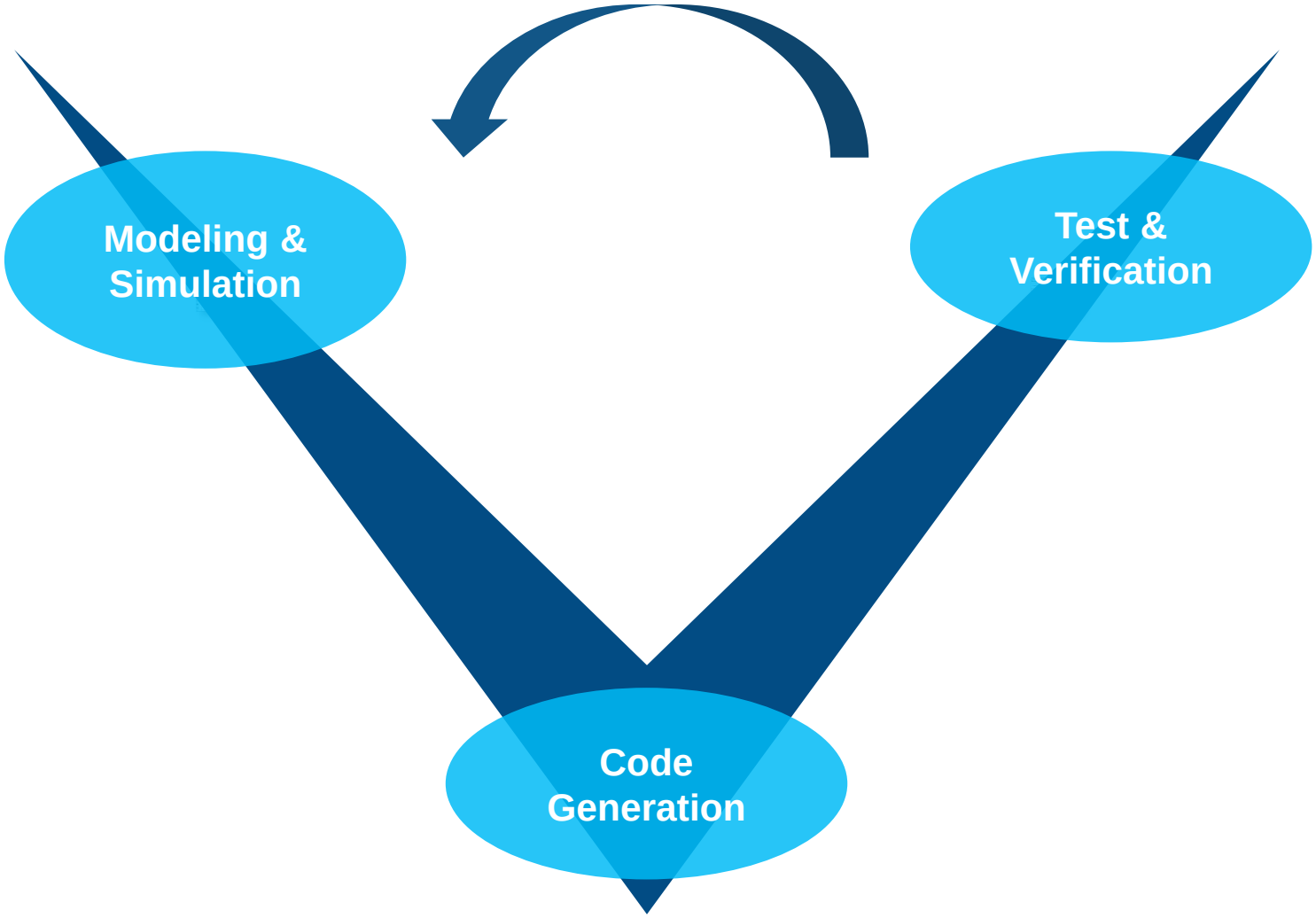
Decision/Condition	True Out	False Out
Transition trigger expression		
Condition 1, "alt>10000"	TF U1.1	Fx U1.1

```
604     /* End of Saturate: '<S210>/Saturation' */
605
606     /* RelationalOperator: '<S196>/NotEqual' */
607     NotEqual_n = (0.0F != Switch_f);
608
609     /* Signum: '<S196>/SignPreSat' */
610     if (Switch_f ≤ 0.0F) {
611       Switch_f = -1.0F;
612     } else {
613       if (Switch_f ≥ 0.0F) {
614         Switch_f = 1.0F;
615       }
616     }
```

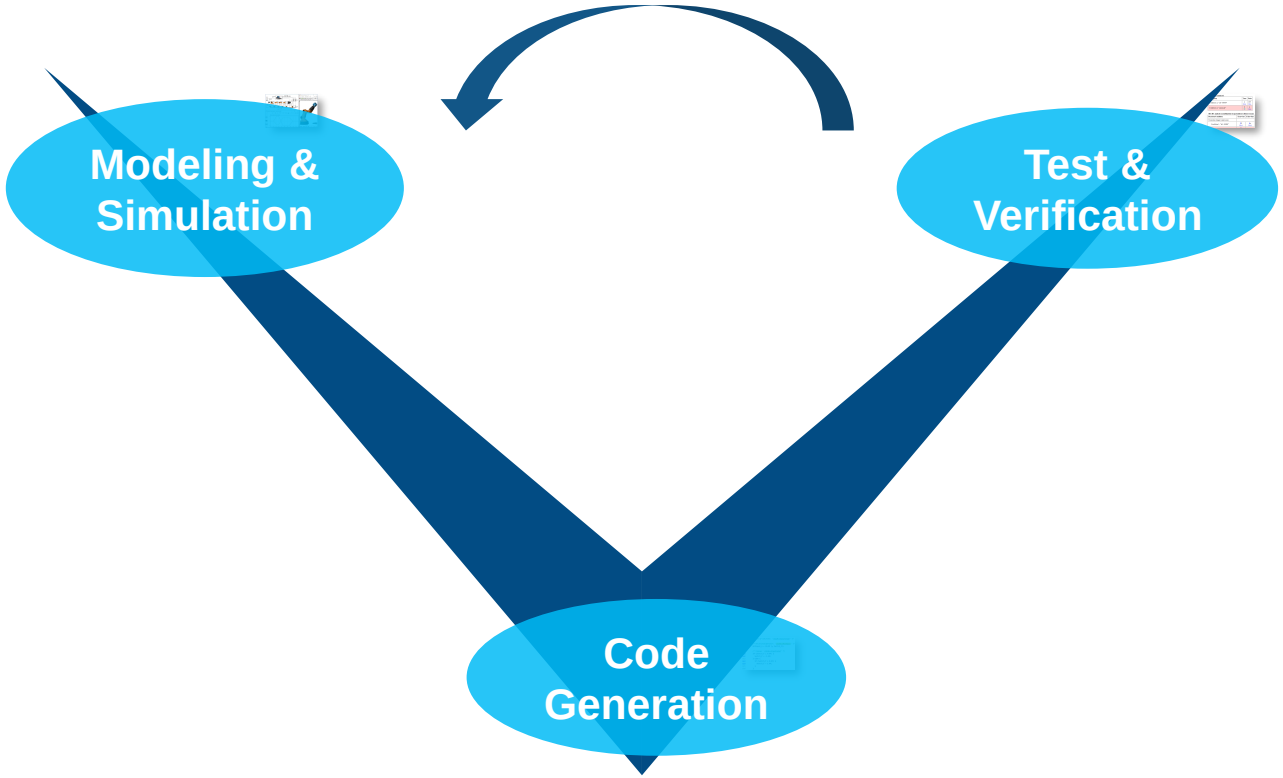
SIMULINK®

Simulation and Model-Based Design

In Model-Based Design, a **system model** is at the **center** of the **workflow**



Traditional Model Based Design Development



Desktop

Design | Analysis

=

Embedded Software

Scaling Development To Address Complexity

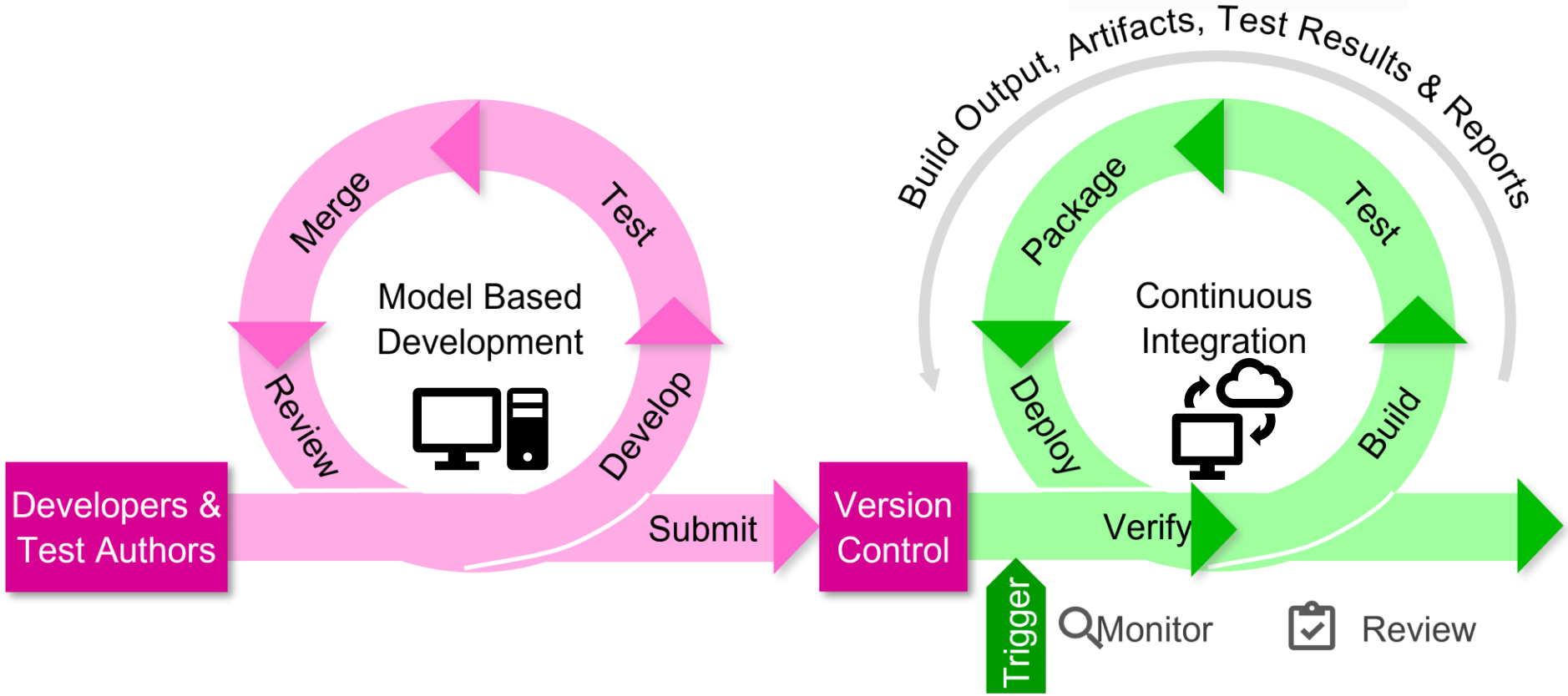
Desktop
Design | Analysis

+

Cloud-DevOps
Build | Verify & Deploy

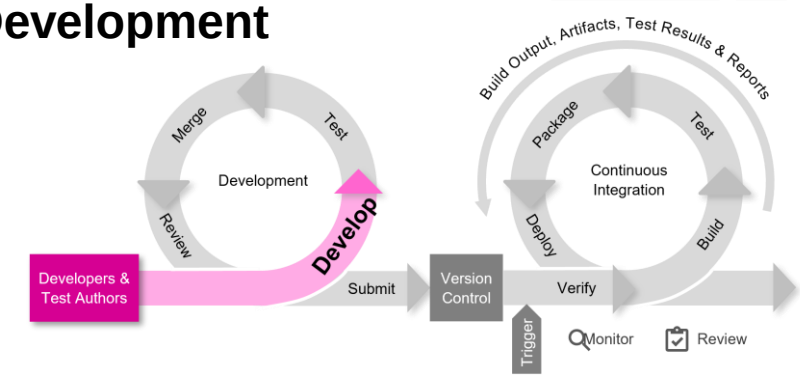
=

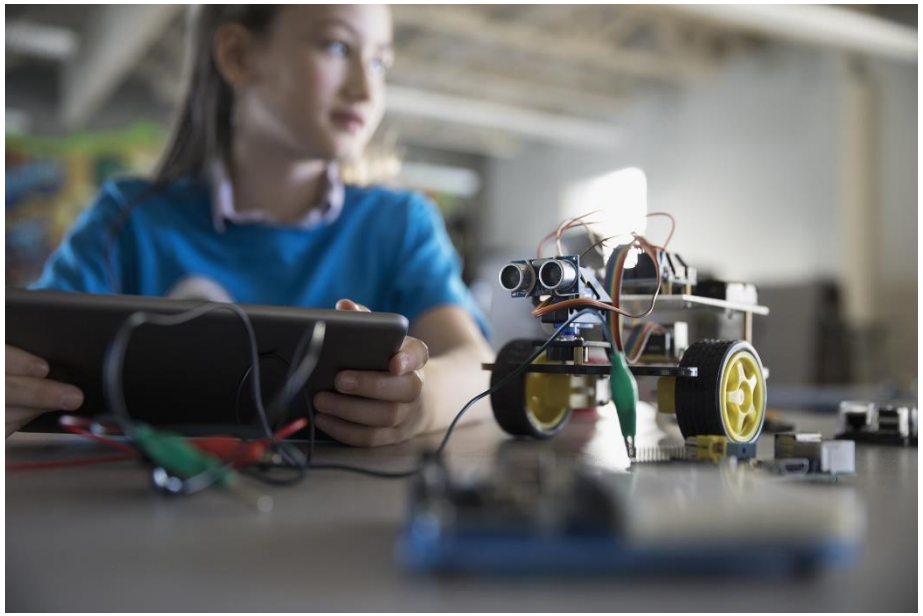
Embedded Software



Make Modeling and Simulation Easier

1 Development

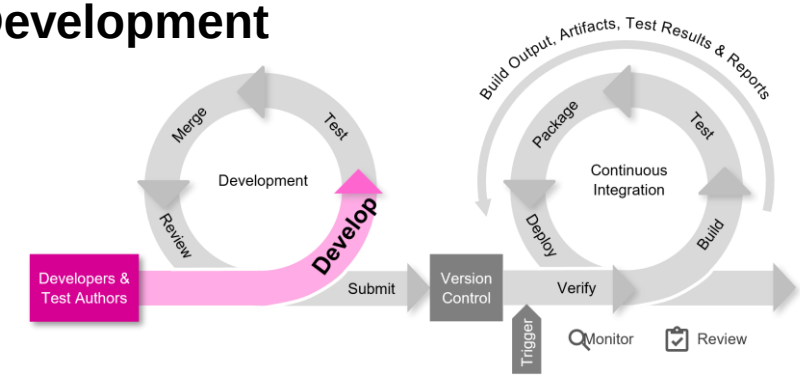


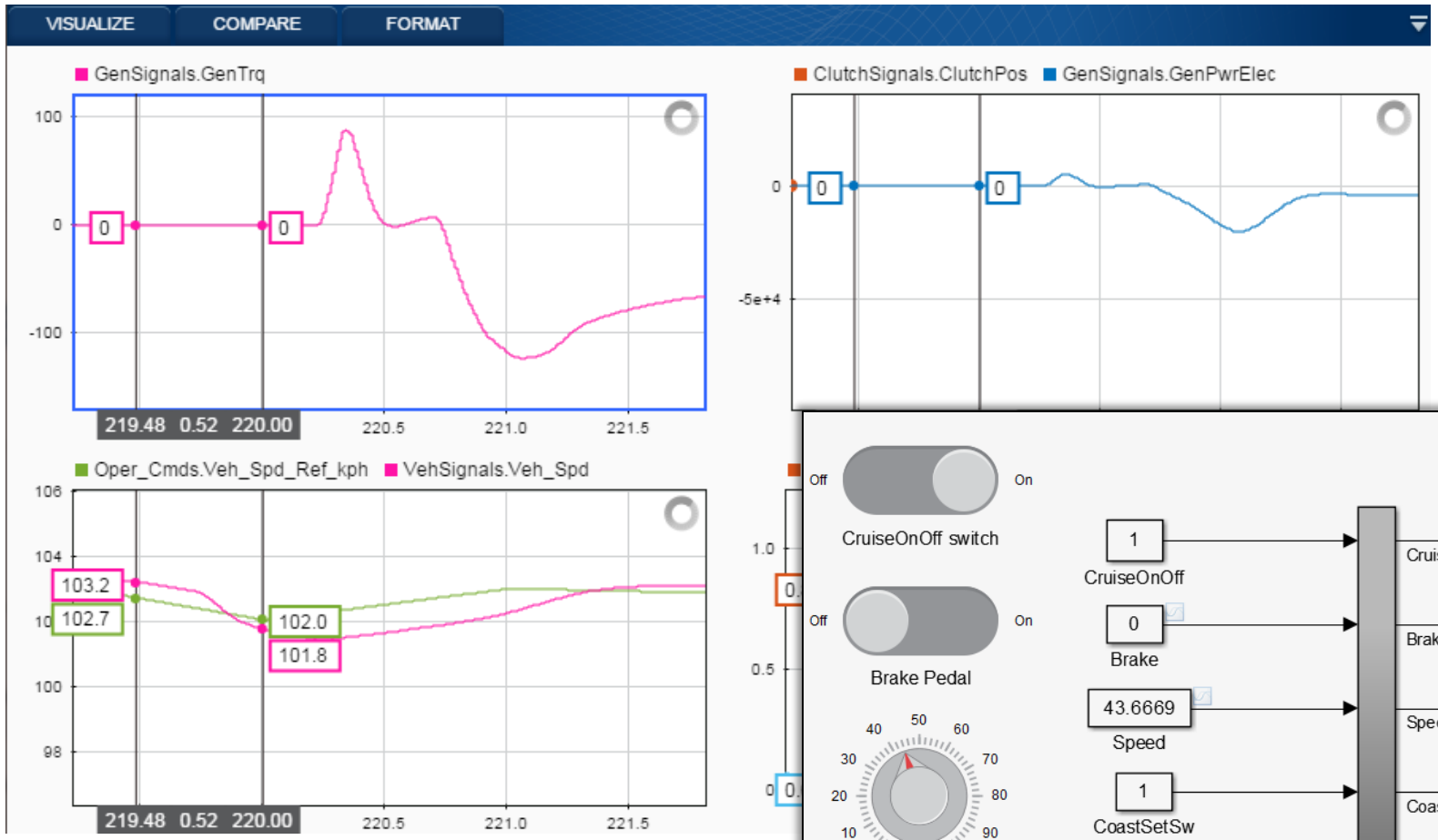


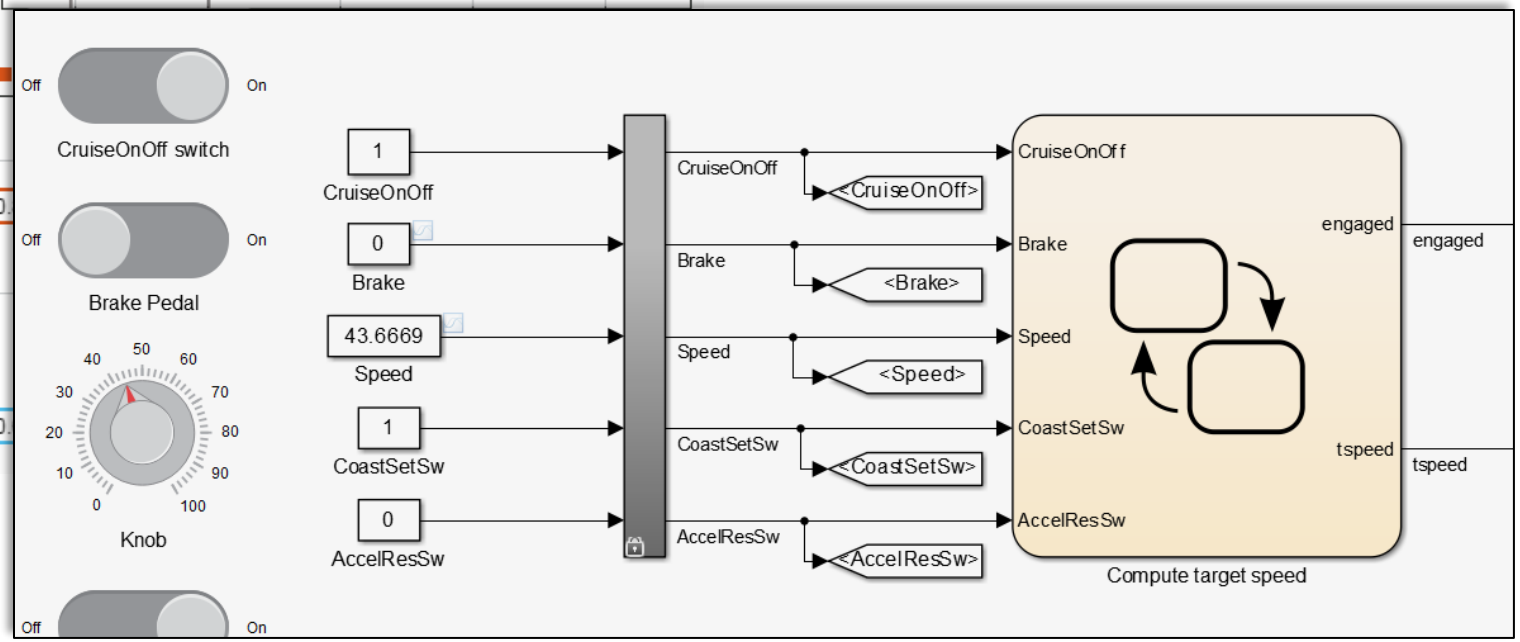
Enable Engineers at Any Level to Model and Simulate Any System

Powerful Interfaces to Explore Behavior

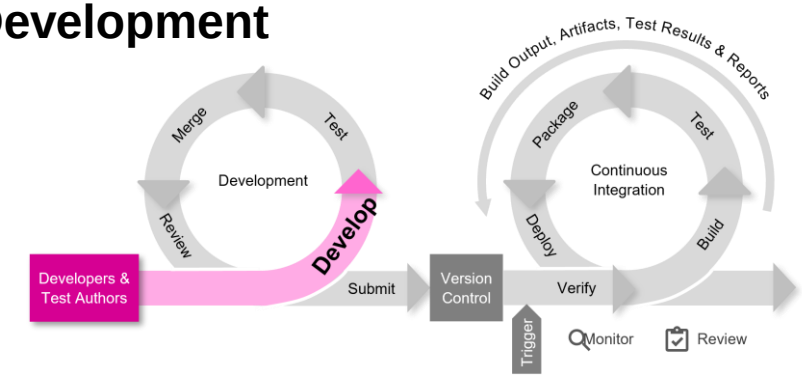
1 Development

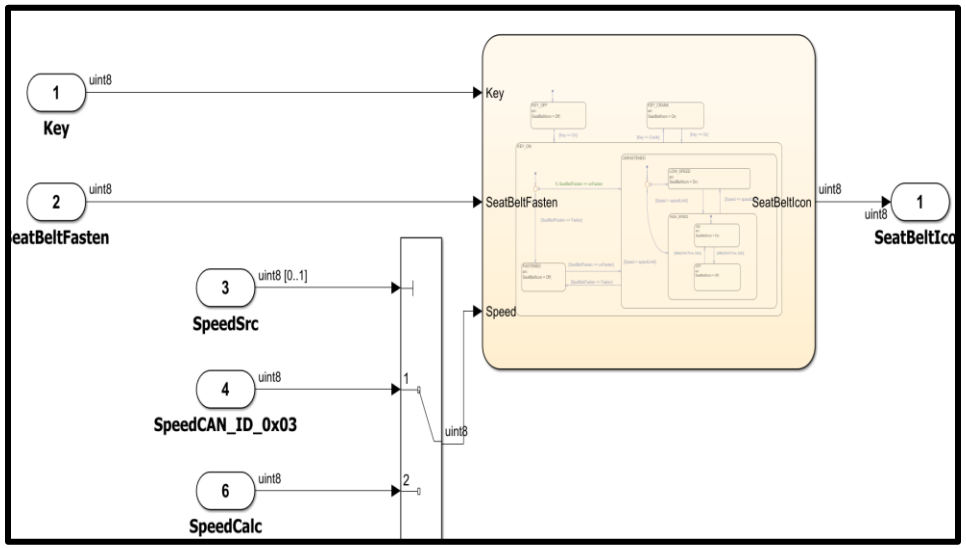






1 Development

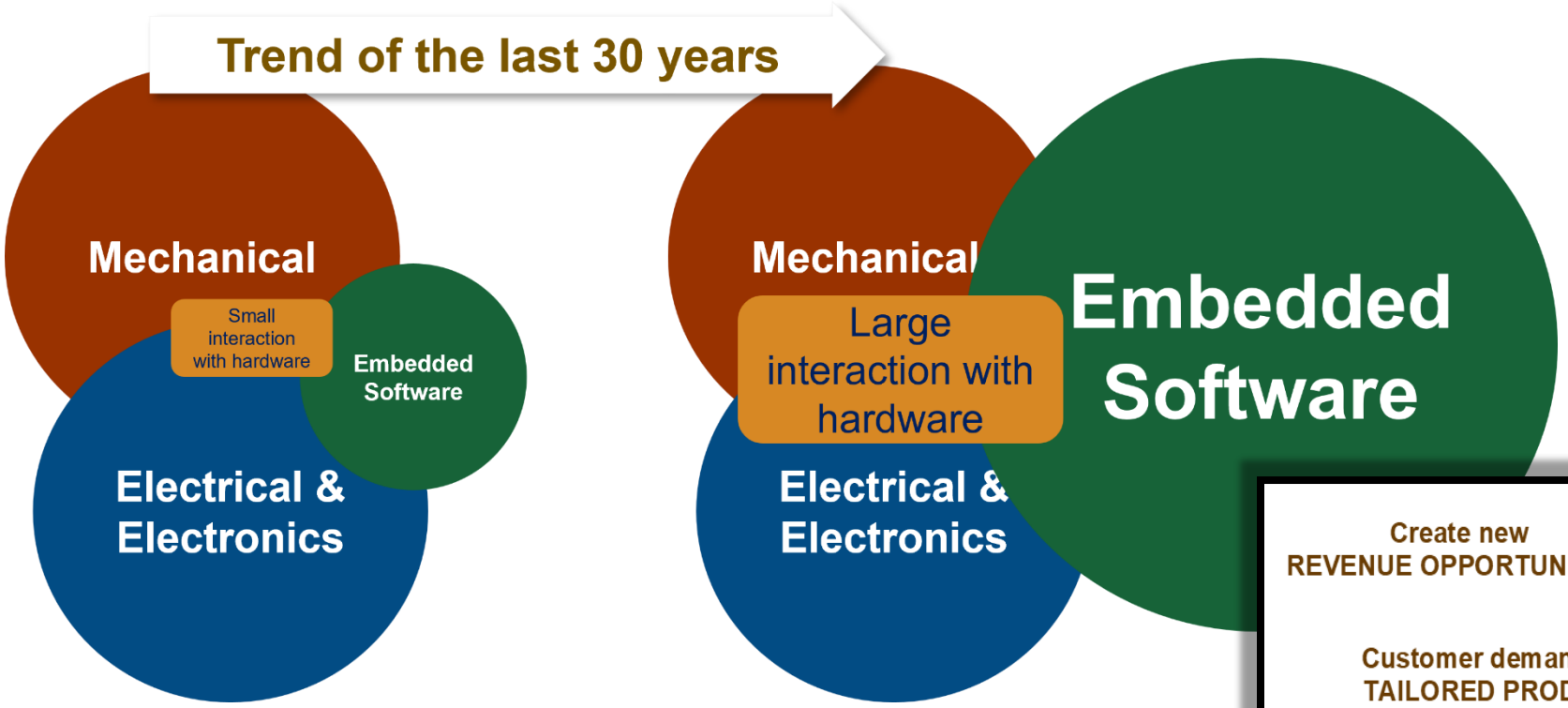




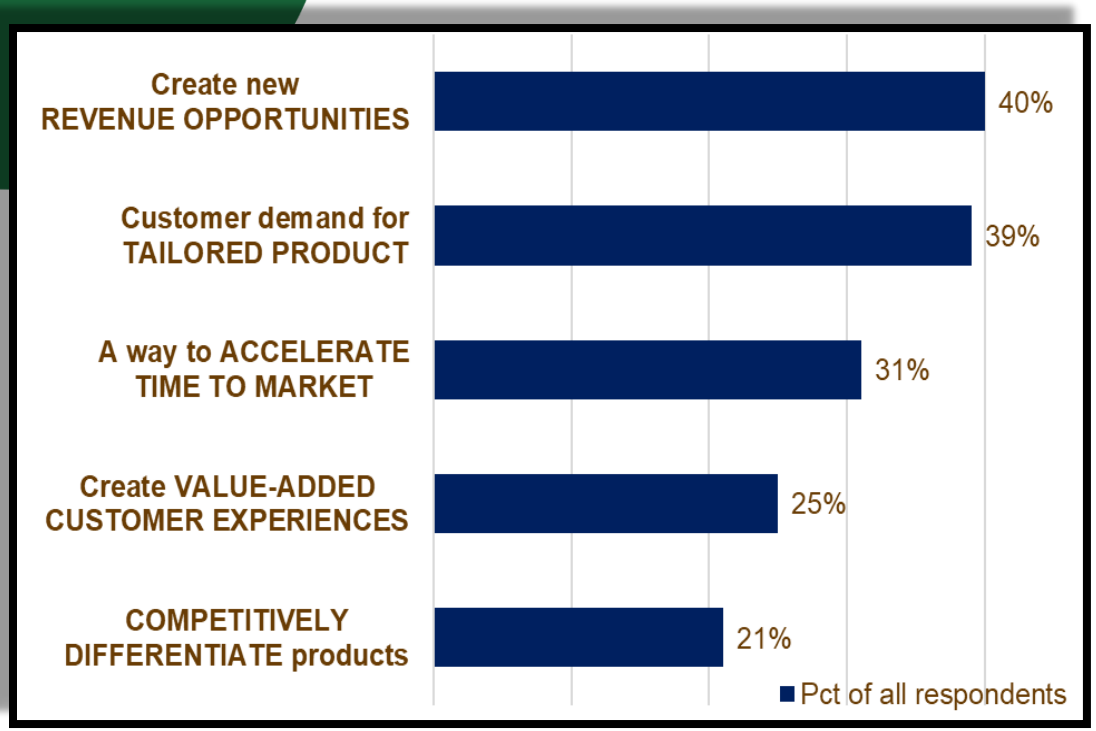
Simulink Model
Application Logic

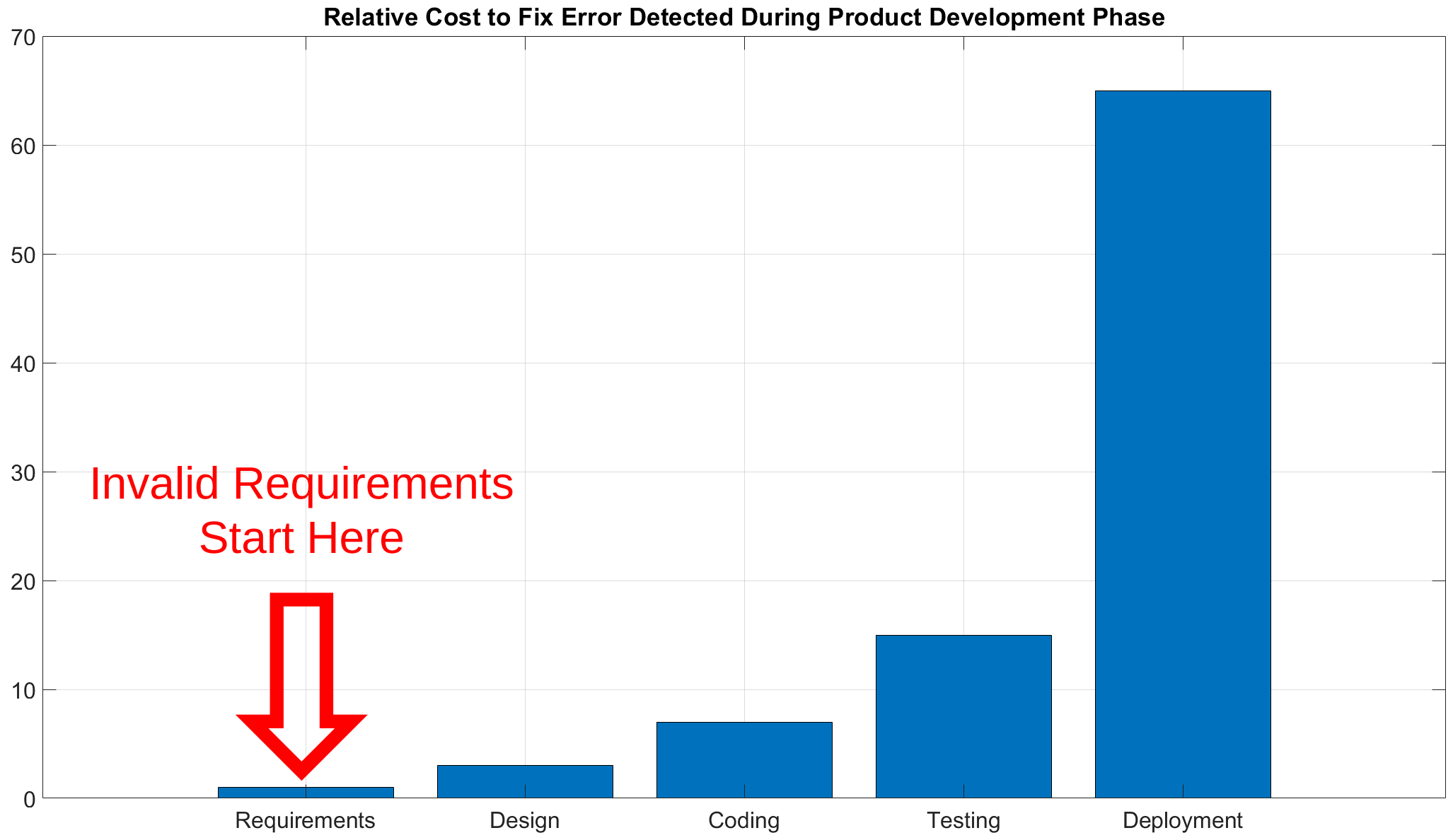
Efficient
C/C++

Why Use Model-Based Design for Embedded System Development?



MathWorks customers tell us that developing embedded software has become ***a major portion of the product design effort***





Data gathered by Hewlett Packard referred by XB in 2017
https://xbsoftware.com/blog/why-should-testing-start-early-software-project-development/

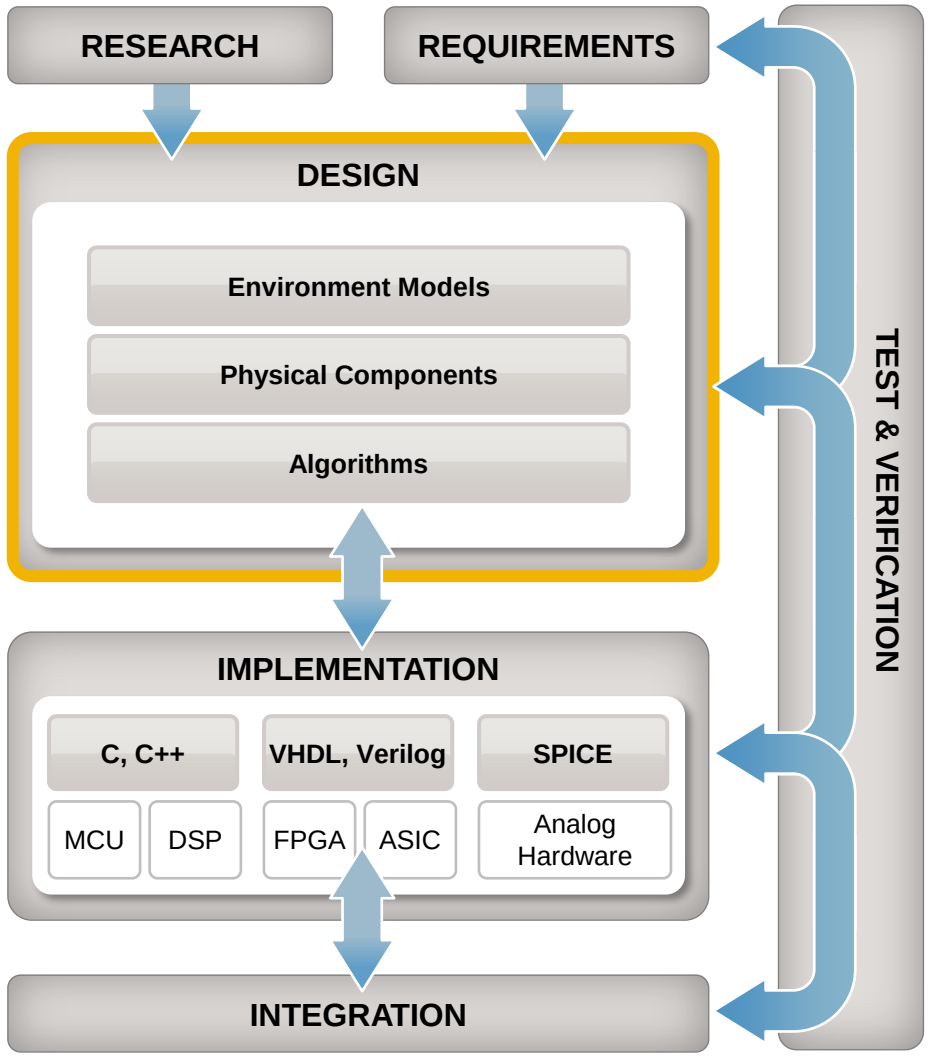
Quantifiable benefits of Model-Based Design

Model-Based Design enabled Continental to verify our design in-vehicle earlier, **eliminating six months of hardware development** and one prototype build. **Verification time was cut by up to 50 percent. 90 percent of application automatically coded.**

Thomas Ehl, Continental

"Front-loaded development with Model-Based Design enables us to **shorten development cycles and minimize rework**, which allows us to **deliver products earlier than our competitors**."

Dr. Hisahiro Ito, Asst. GM.



System models reused across 54 products worldwide. "Once we had moved to Model-Based Design, we were able to use the same core system in many different vehicles by simply calibrating parameters such as the vehicle dimensions **and then re-generating production code.**"

Johan Hägnander, GM Engineering Europe

"We use our system design model in Simulink for ARP4754 to establish stable, objective requirements. **We save time by using the model as the basis for our software design model for DO-178—from which we generate flight code**—and reusing validation tests for software verification."

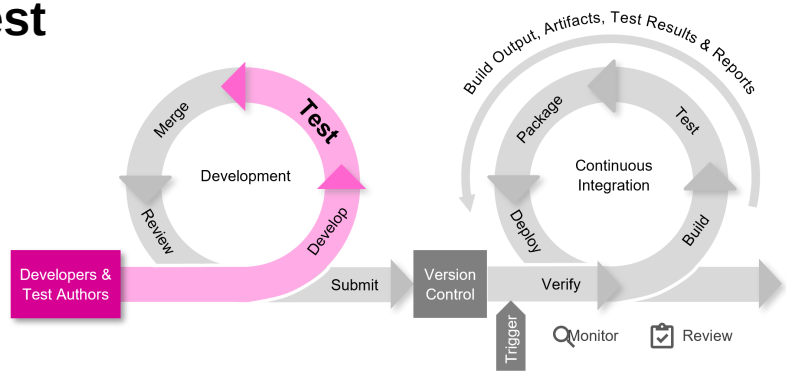
Ronald Blanrue, Airbus Helicopters

Development Processes for High-Integrity Applications

- High integrity applications development follows standards and guidelines
- Standards and Guidelines have objectives for development process activities
  - Impose additional constraints on development
  - Require creation of additional artifacts
  - Require more thorough verification, validation and testing activities
- Standards and Guidelines require evidence that the objectives were met to certify: compliance demonstration

Systems Analysis

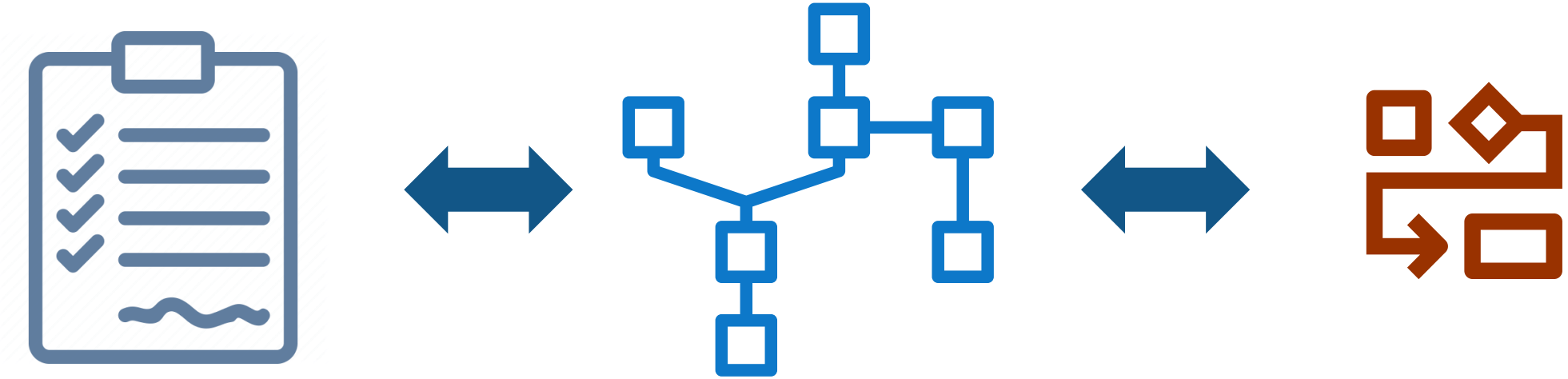
2 Test



Requirements

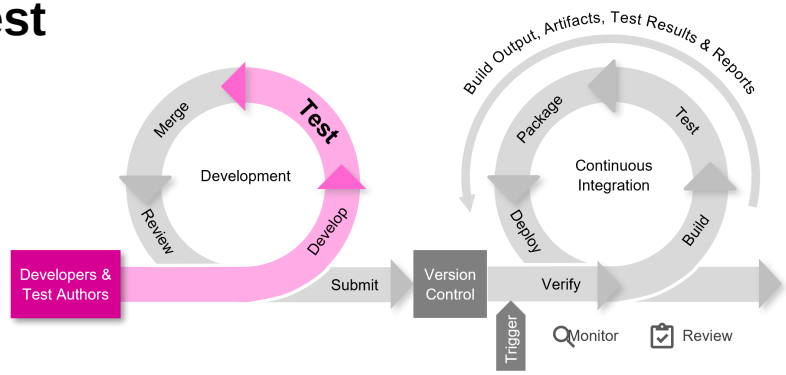
Systems

Components

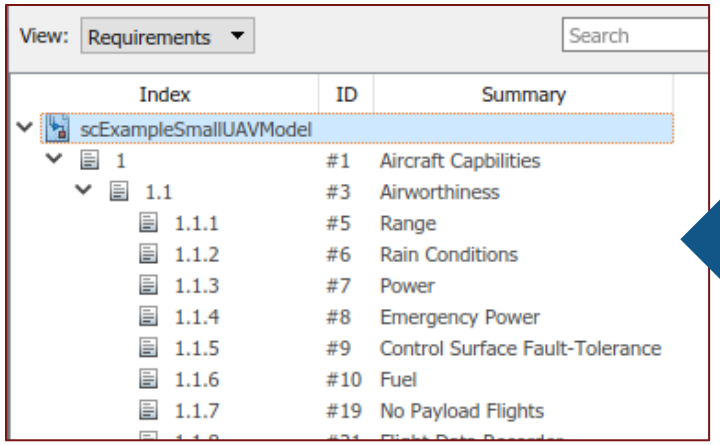


Systems Analysis

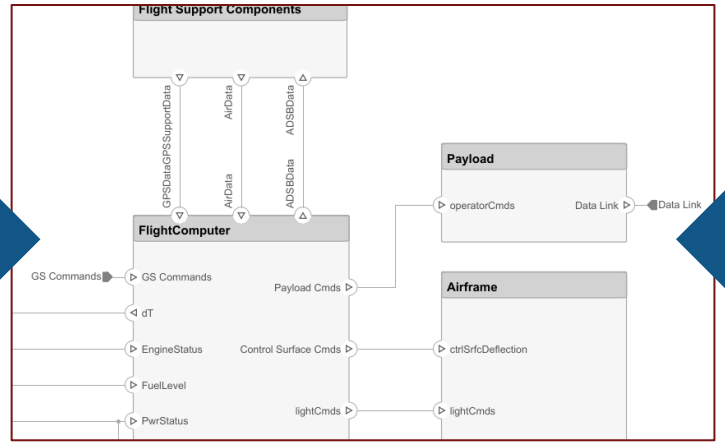
2 Test



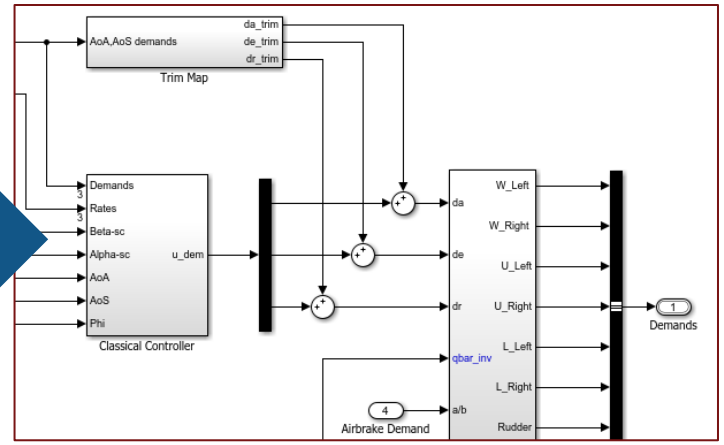
Requirements Toolbox



System Composer

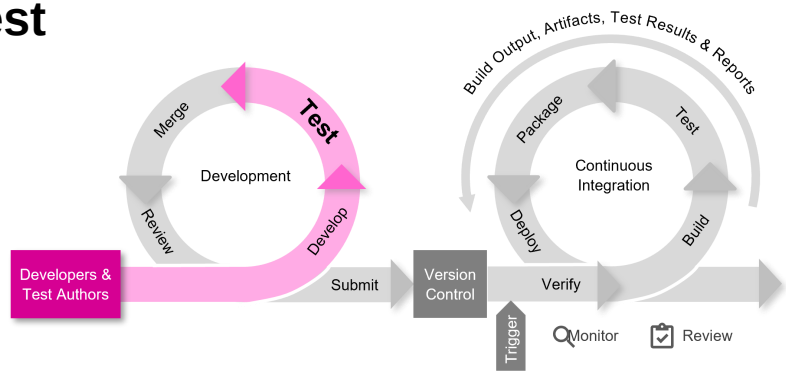


Simulink



High Integrity Verification Workflow

2 Test



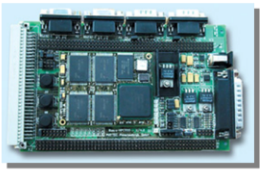
Component and system testing

Review and static analysis

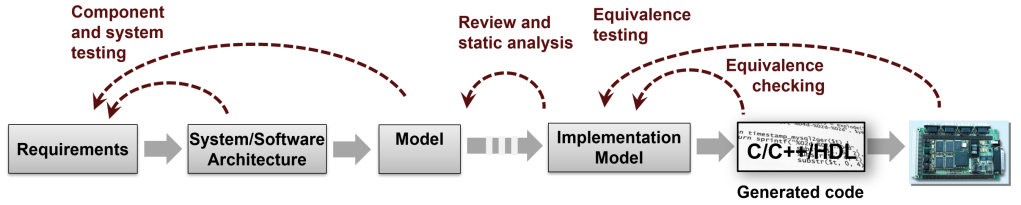
Equivalence testing

Equivalence checking

Requirements → System/Software Architecture → Model → Implementation Model → C/C++/HDL → Generated code

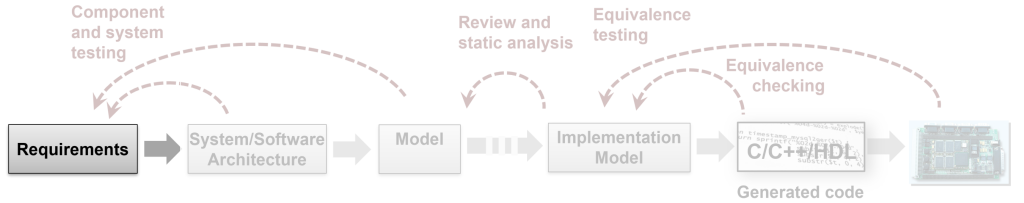


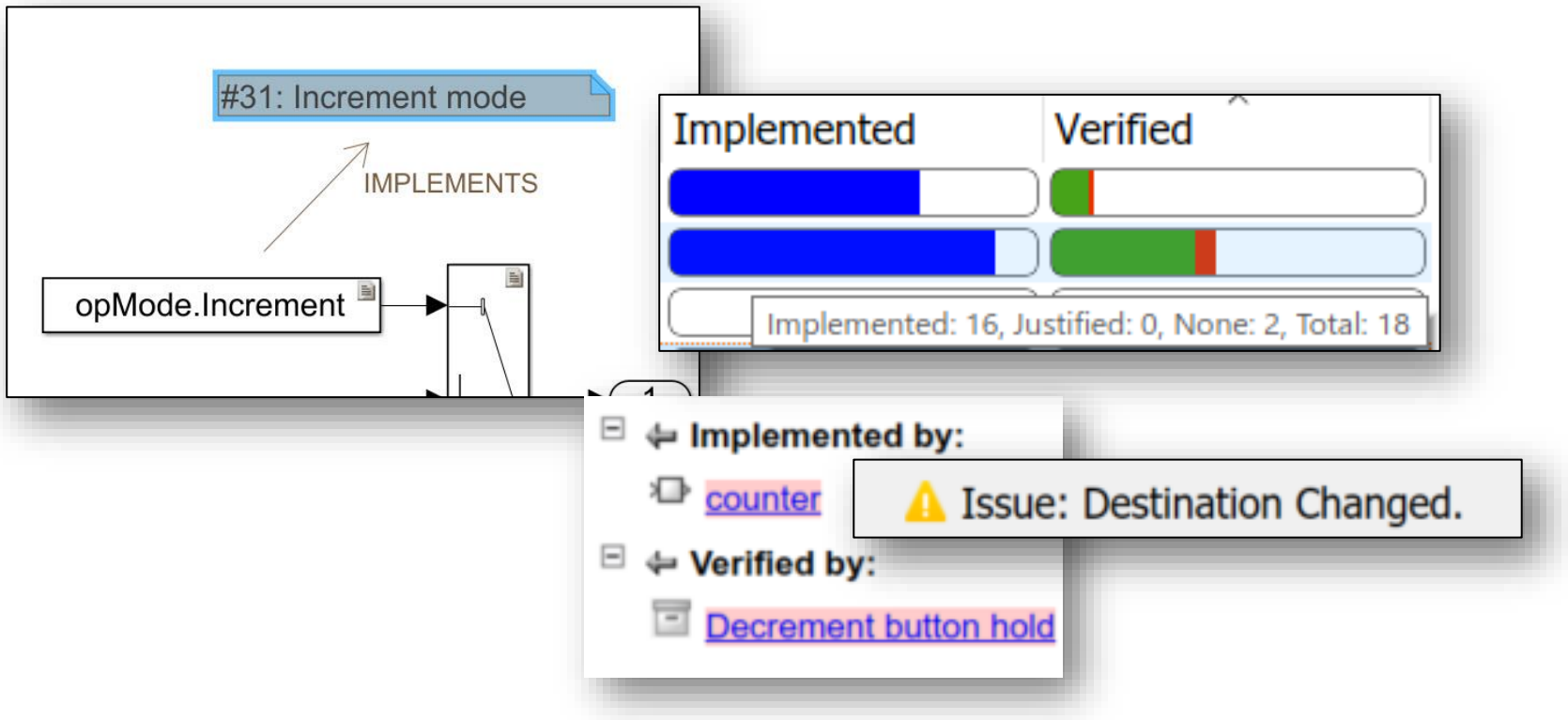
High Integrity Verification Workflow



2 Test

Requirements Traceability and Analysis





- Where are requirements implemented?
- Is design and requirements consistent?
- How are they tested?

Why traceability matters…

Ensure application is complete, fully tested, and meets customer requirements

Understand the impact of requirement changes to implementation and test (i.e. “Digital Thread”)

Required to meet certification standards such as:

- ISO 26262, ASPICE for Auto
- DO-178C for Aerospace
- IEC 62304 in Medical
- Many others….

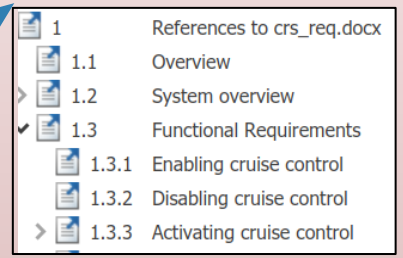
Author, link, and validate requirements for designs and tests

External Requirements

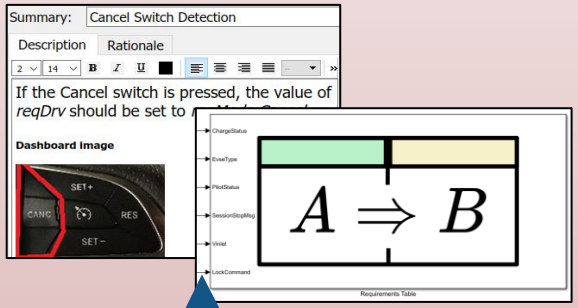
Requirements Management Tools

Requirements Toolbox

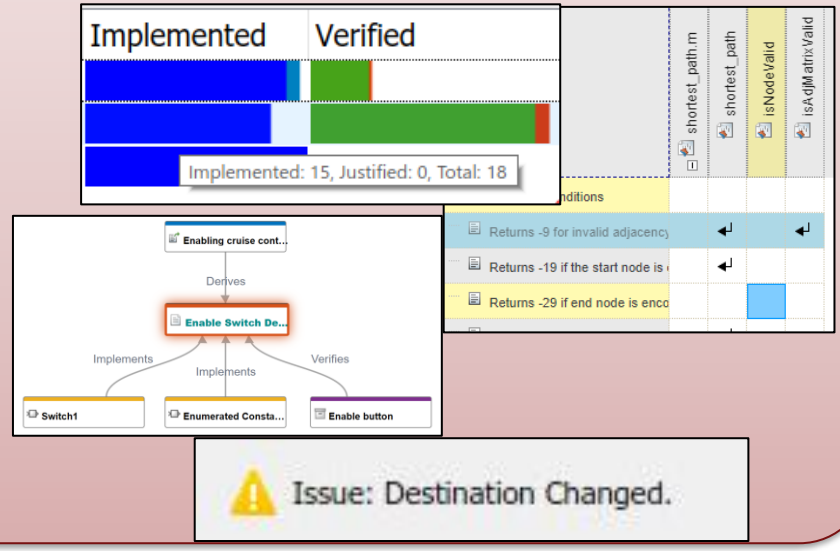
Import / Export



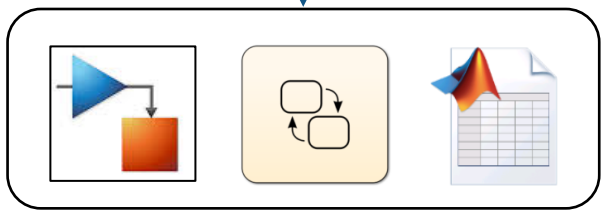
Author / Model



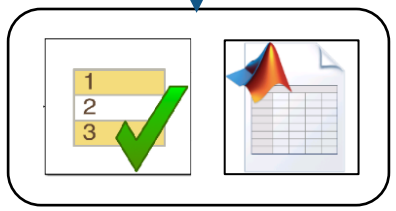
Analyze



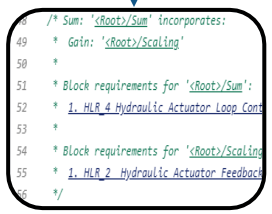
Trace



Simulink, System Composer, Stateflow, MATLAB Code

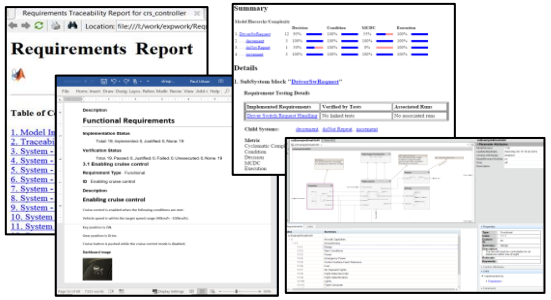


Simulink Test MATLAB Unit Test

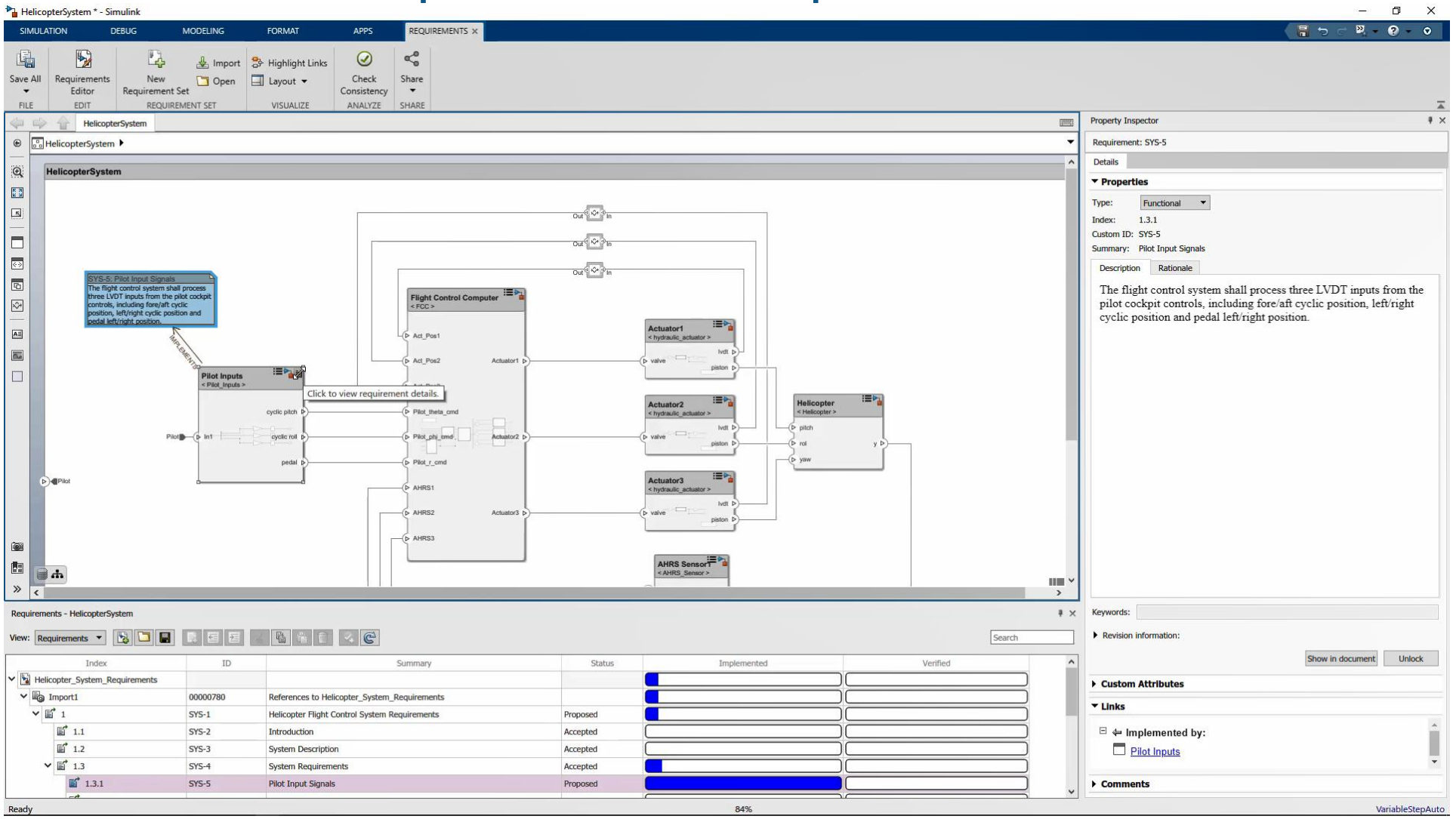


Generated Code

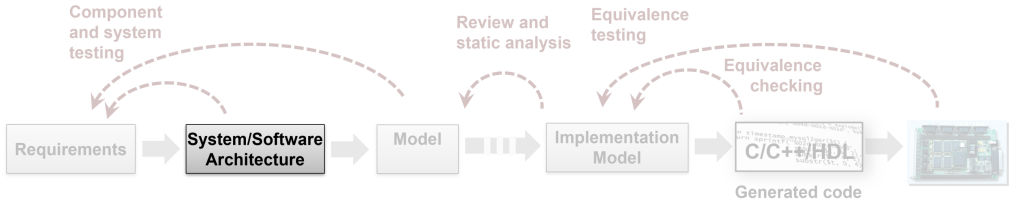
Report

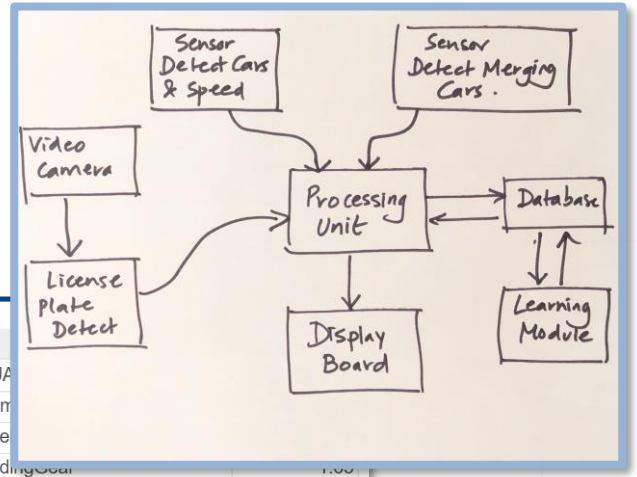


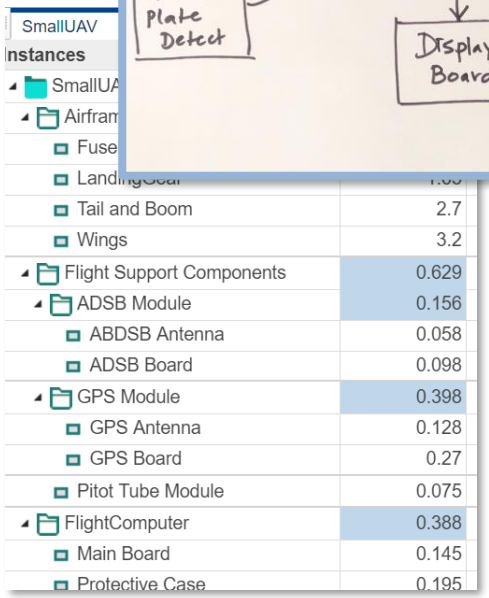
Work with DOORS requirements within System Composer, Simulink or Stateflow with Requirements Perspective



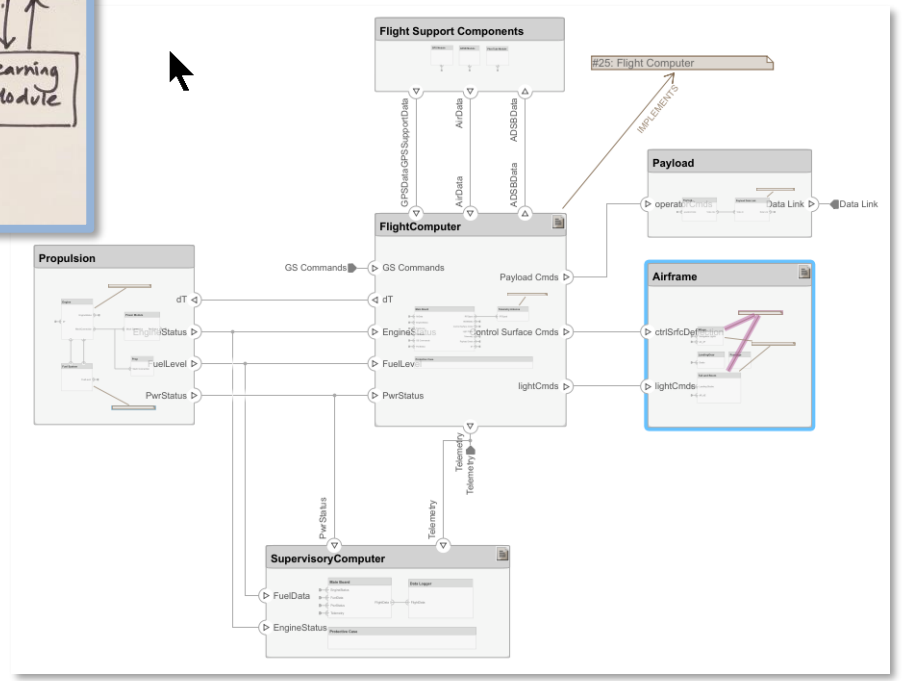
Model-Based Systems Engineering







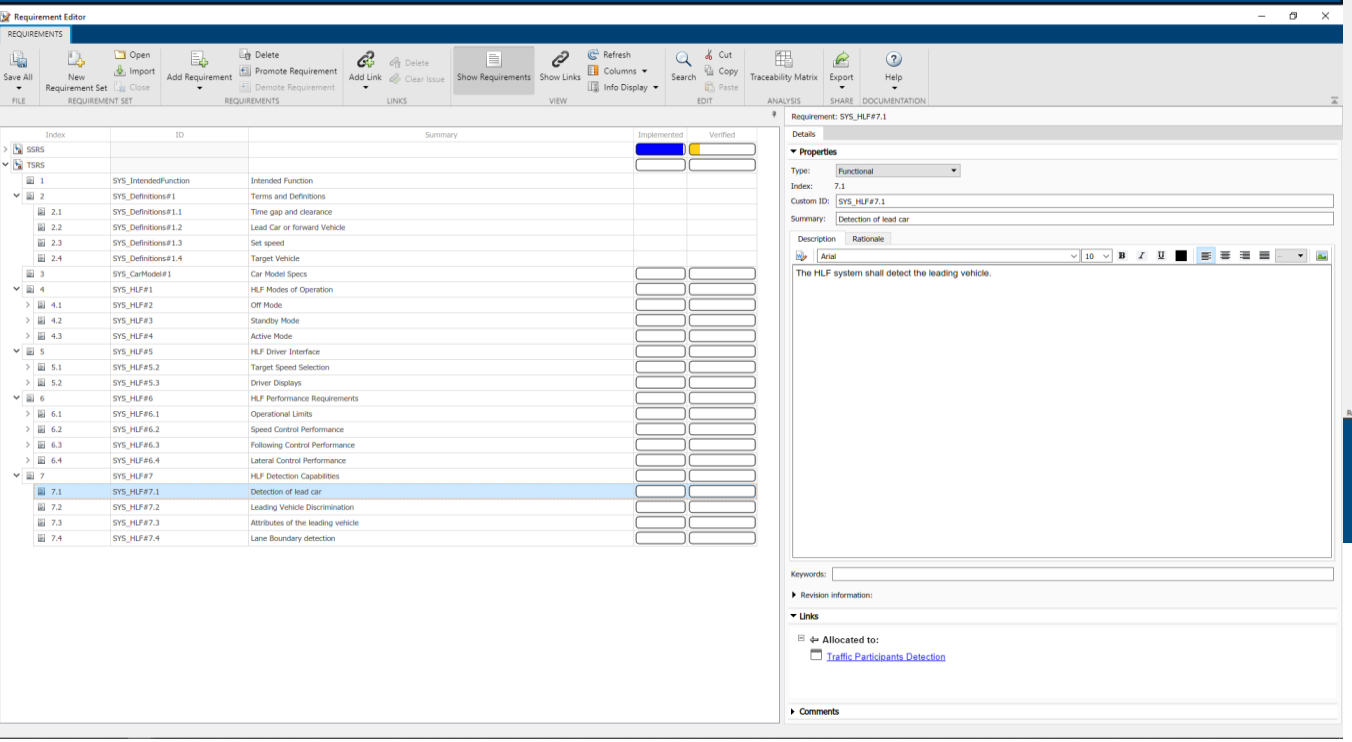
System Composer

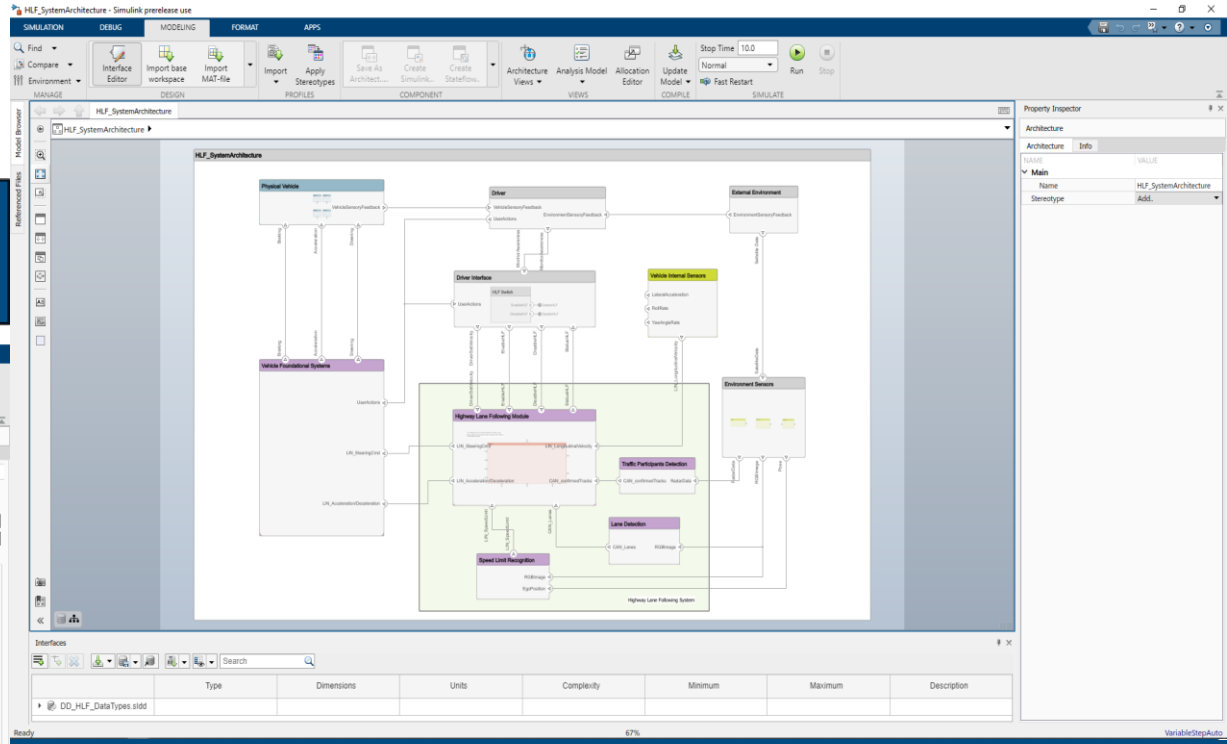


How to Model and Analyse System and Software Architecture

Model-Based Systems Engineering

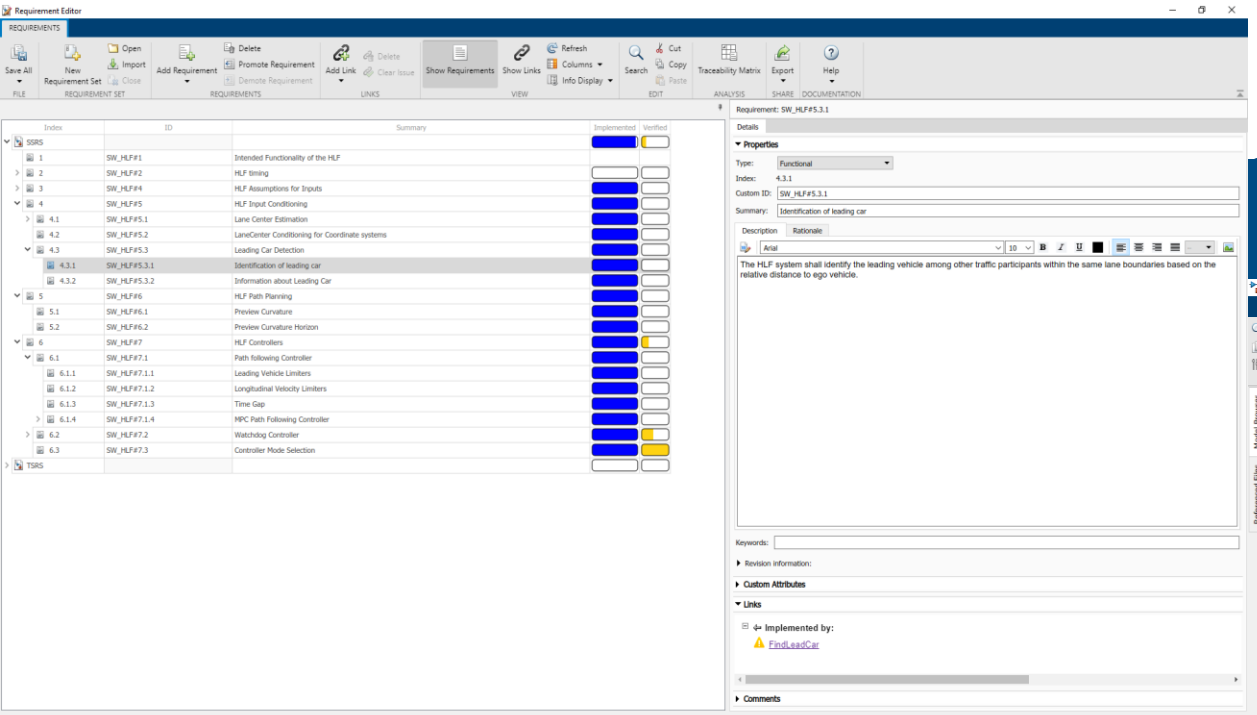
System requirements
(Simulink Requirements)





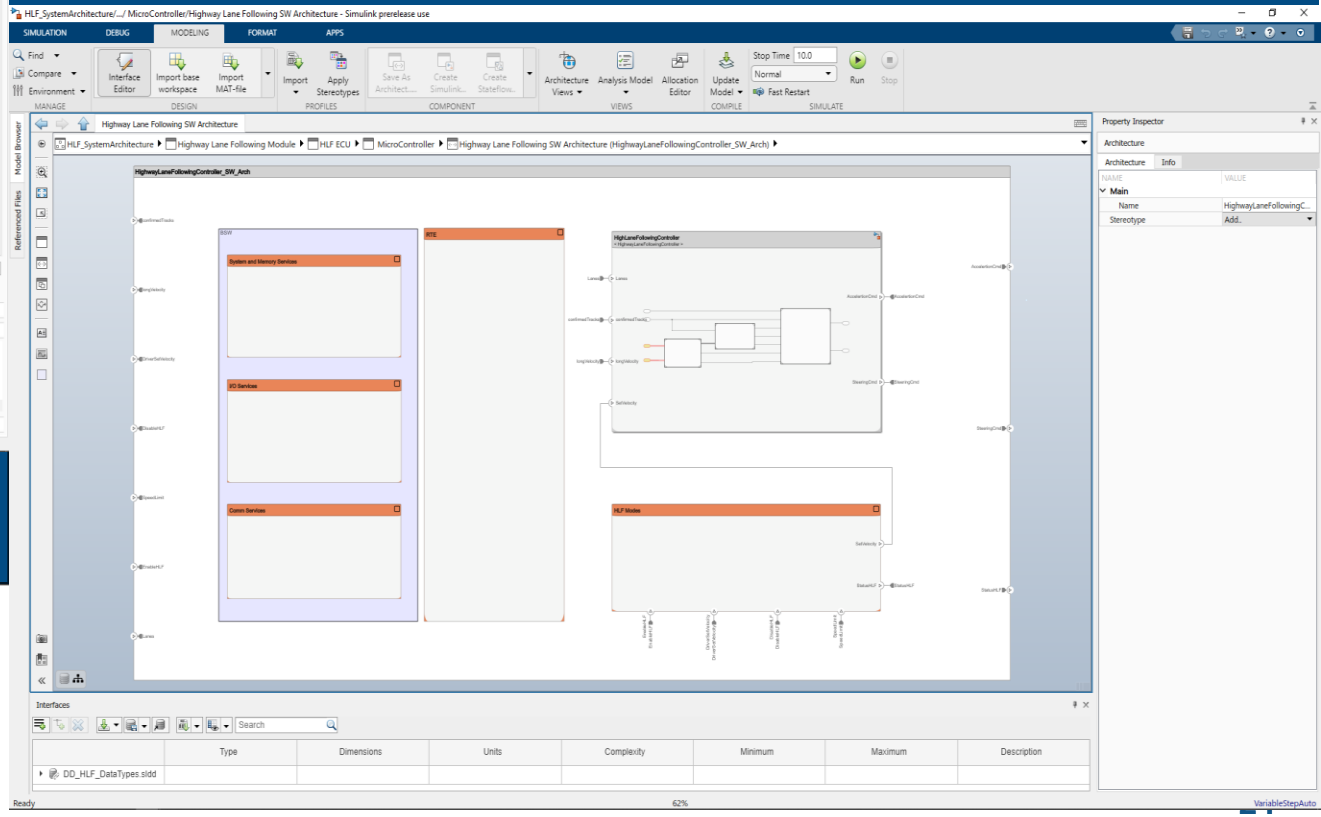
System Architecture Models
(System Composer)

Model-Based Systems Engineering

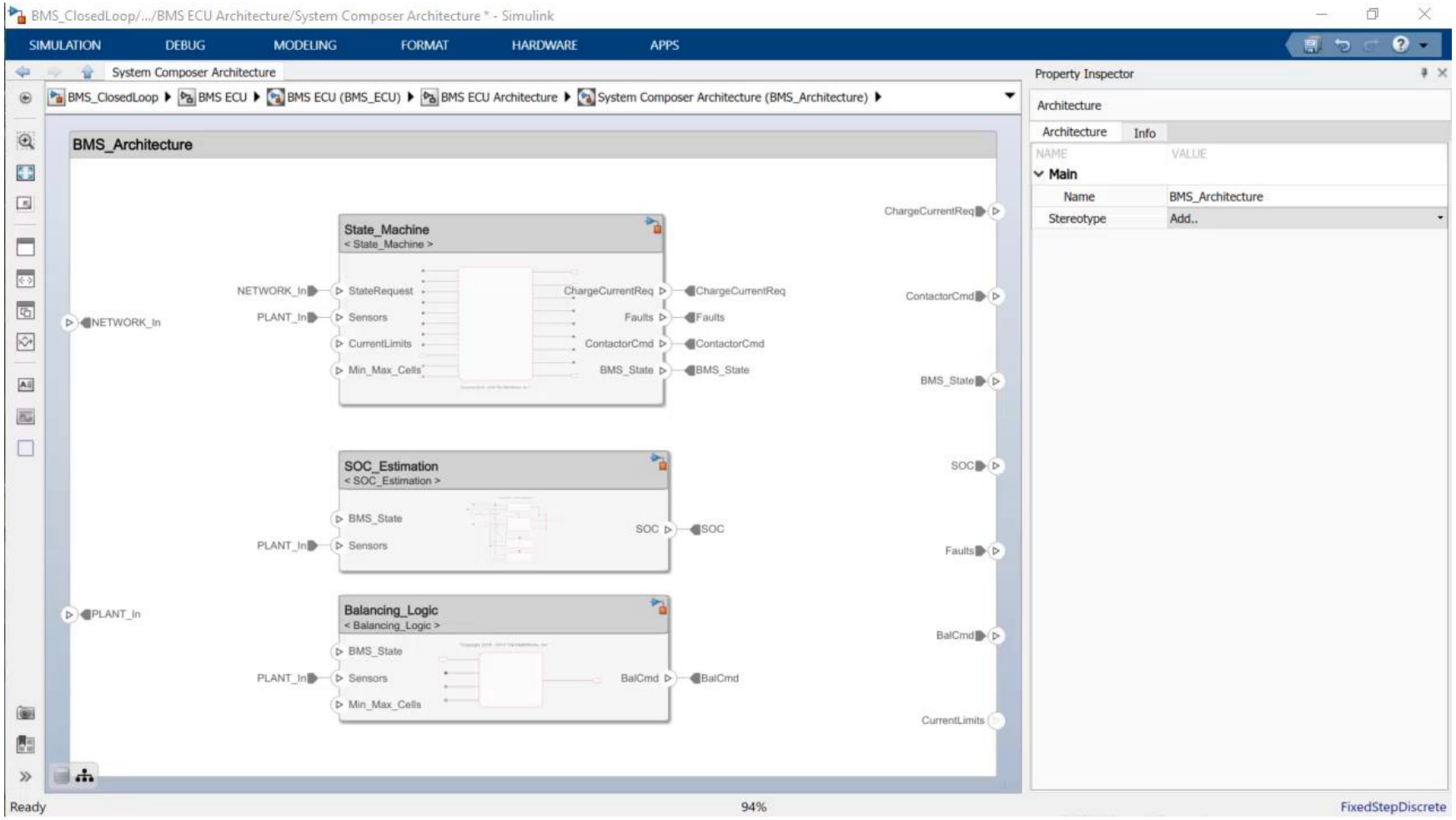


Software Requirements
(Simulink Requirements)

Software Architecture
(System Composer and Simulink)



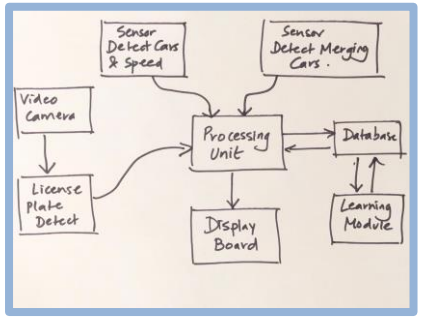
Model-Based Systems Engineering

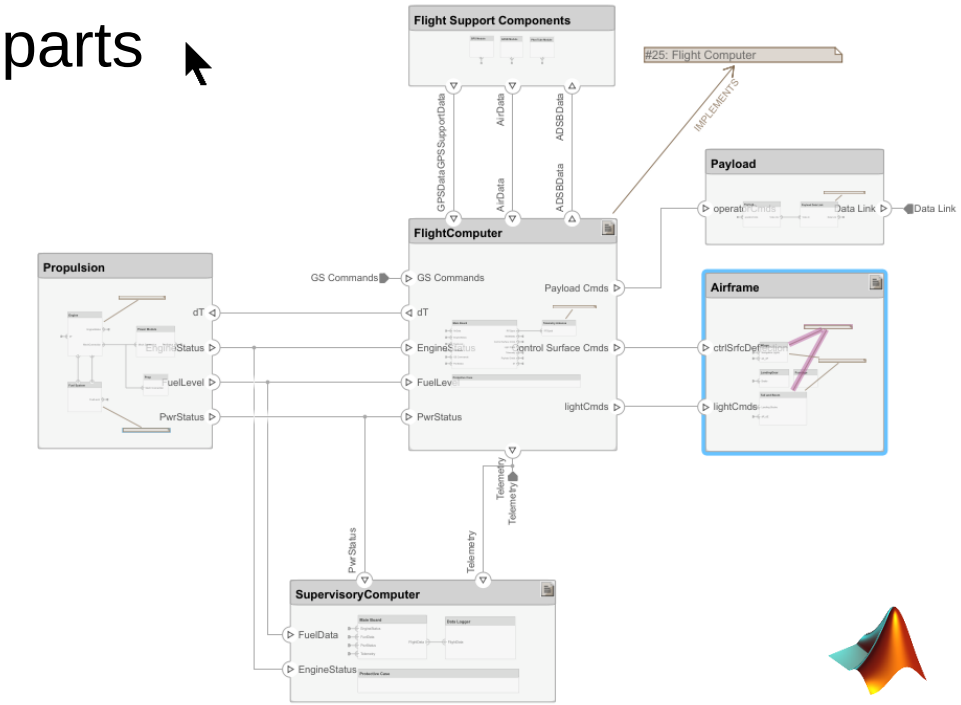


Model-Based Systems Engineering

- Architecture Models
- Profiles, stereotypes, properties
- Allocate requirements
- Views to focus on relevant parts
- Perform Analysis

System Composer

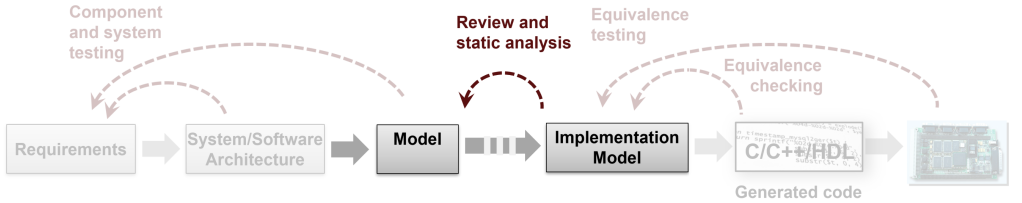


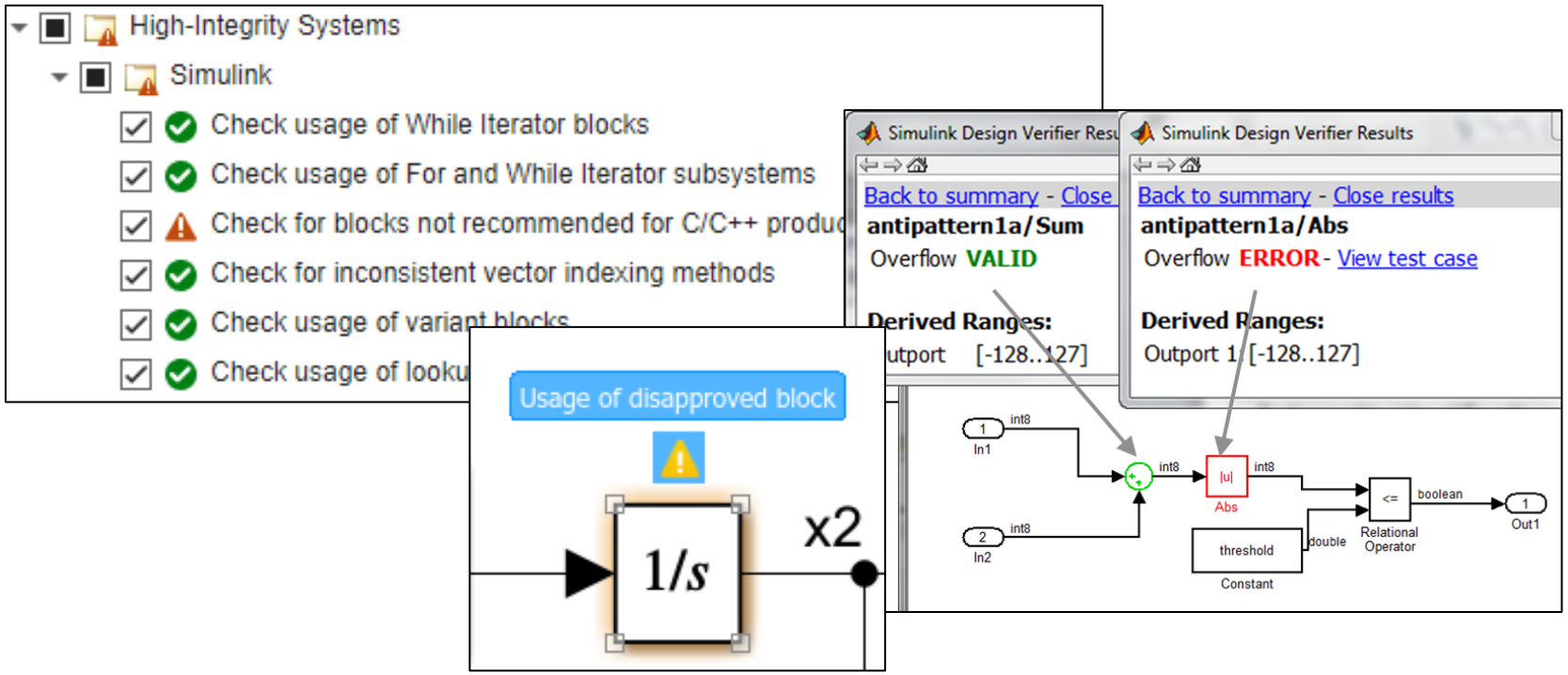


SmallUAV

Instances	Mass(kg)
SmallUAV	15.932
Airframe	9.25
Fuselage	1.7
LandingGear	1.65
Tail and Boom	2.7
Wings	3.2
Flight Support Components	0.629
ADSB Module	0.156
ABDSB Antenna	0.058
ADSB Board	0.098
GPS Module	0.398
GPS Antenna	0.128
GPS Board	0.27
Pitot Tube Module	0.075
FlightComputer	0.388
Main Board	0.145
Protective Case	0.195

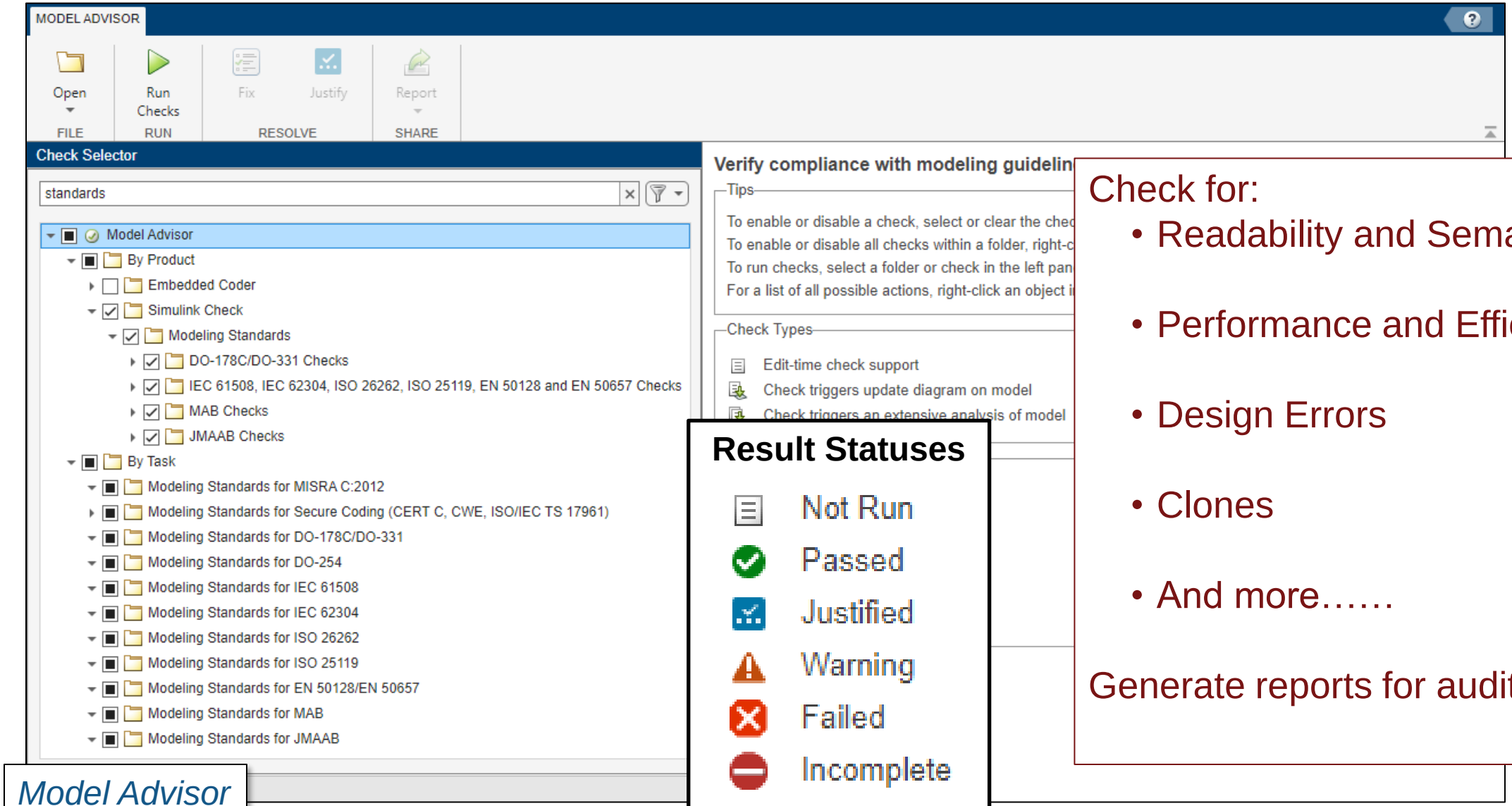
Compliance to Standards and Guidelines





Is the design built right?
Is it too complex?
Is it ready for code generation?

Automate verification with static analysis



Model Advisor

Result Statuses

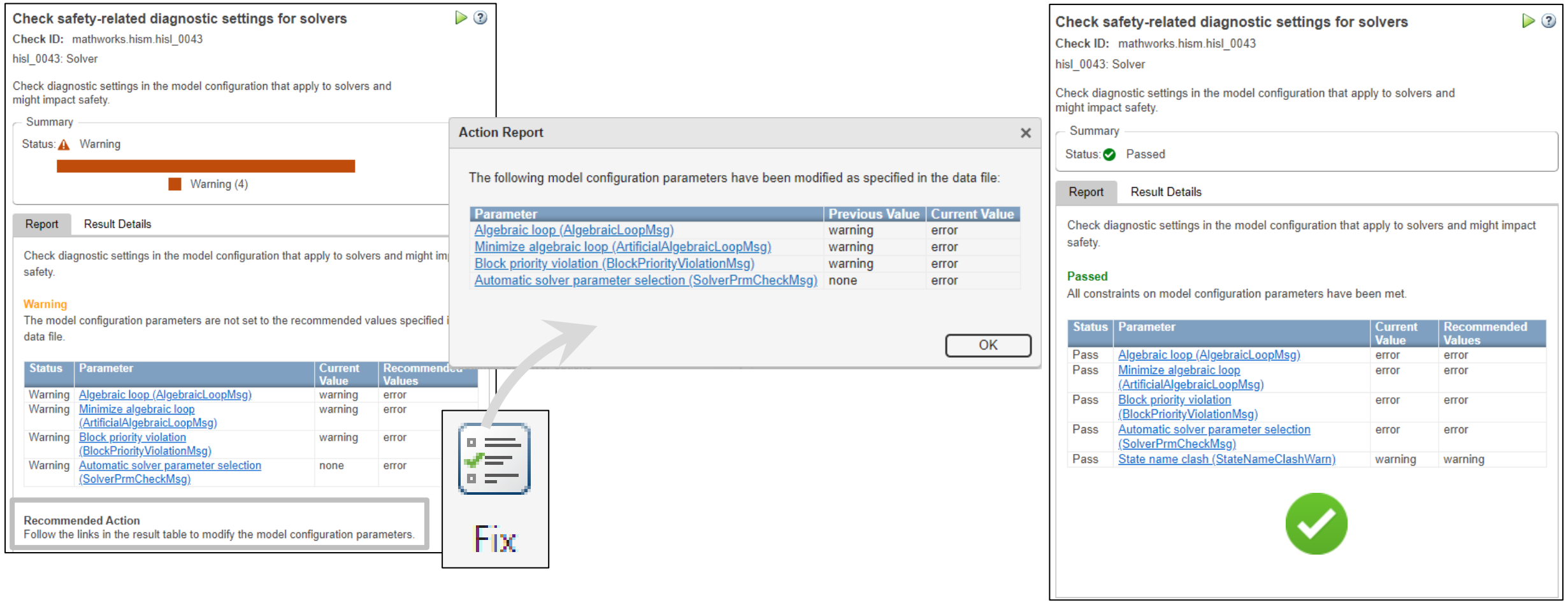
- Not Run
- Passed
- Justified
- Warning
- Failed
- Incomplete

Check for:

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

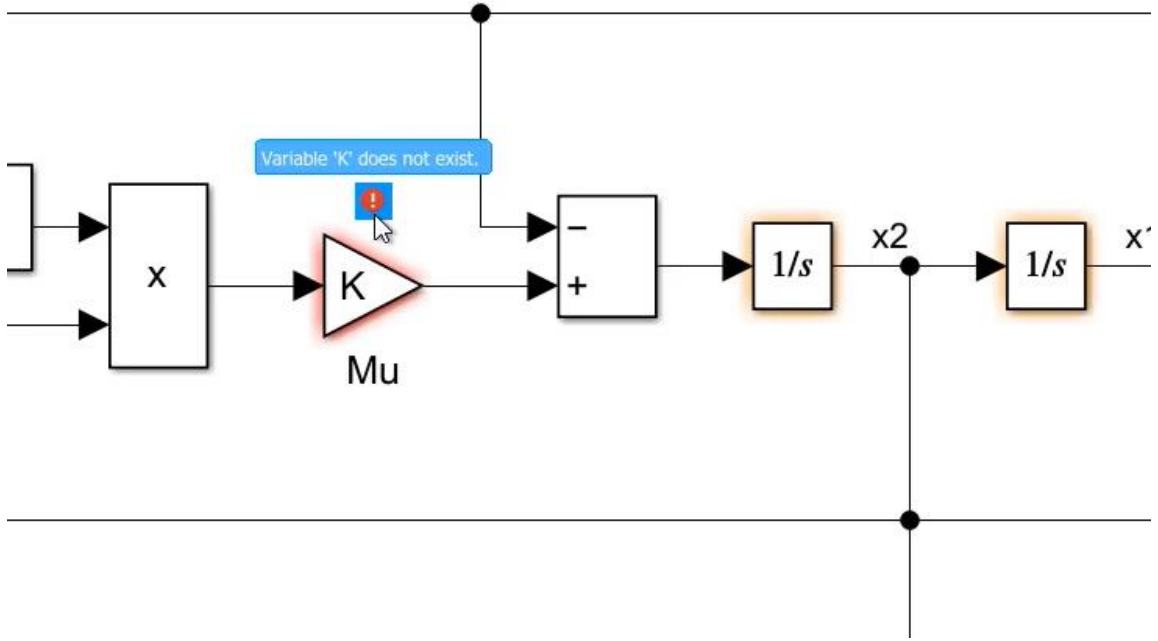
Generate reports for audits

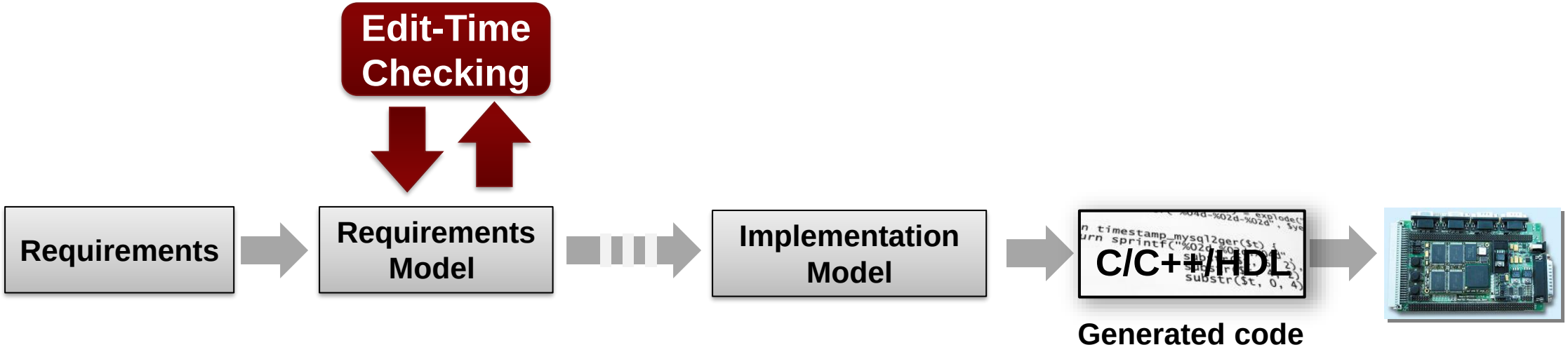
Guidance Provided to Address Issues or Automatically Correct



Shift Verification Earlier With Edit-Time Checking

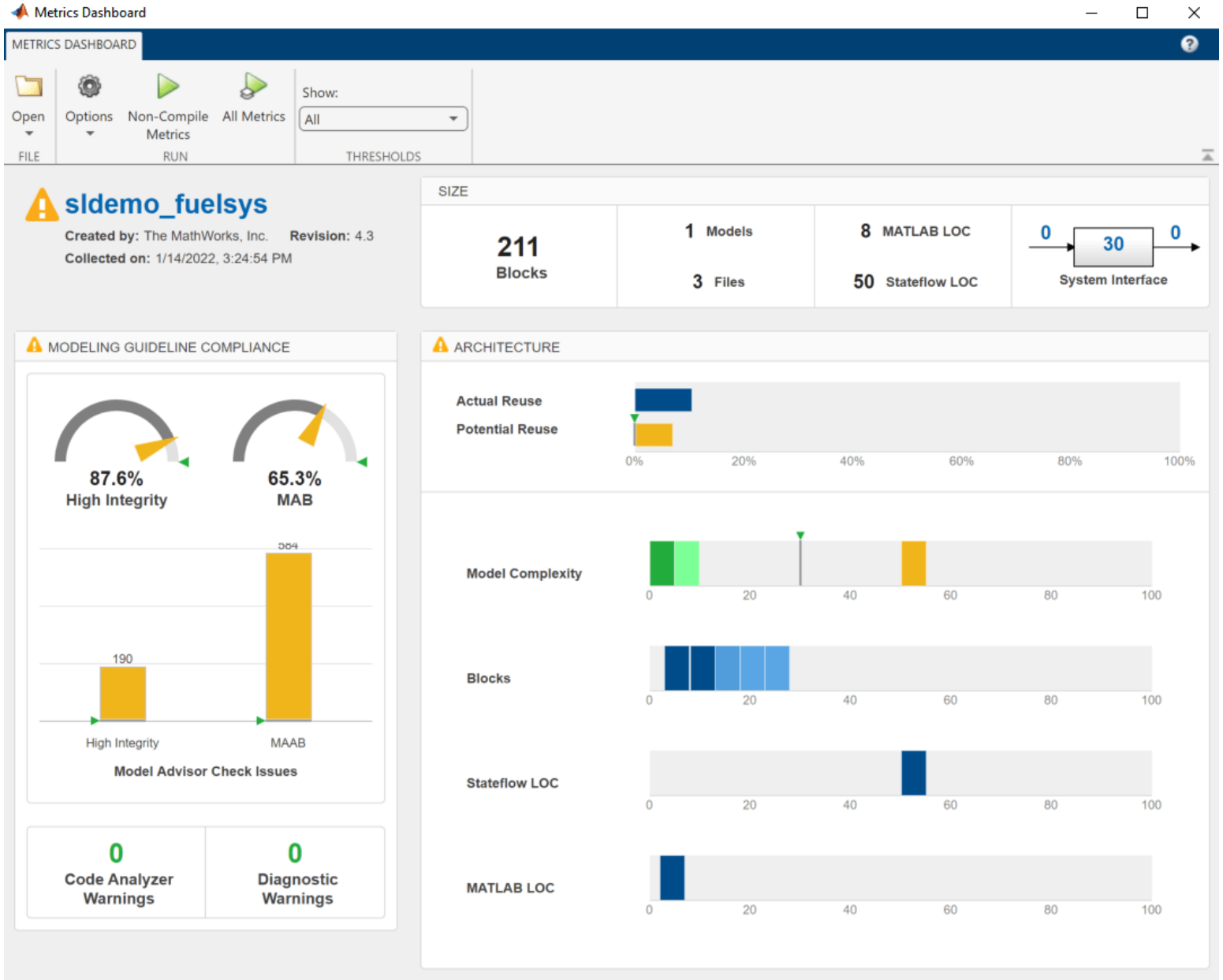
- Highlight violations as you edit
- Fix issues earlier
- Avoid rework
- Author and customize edit-time checks



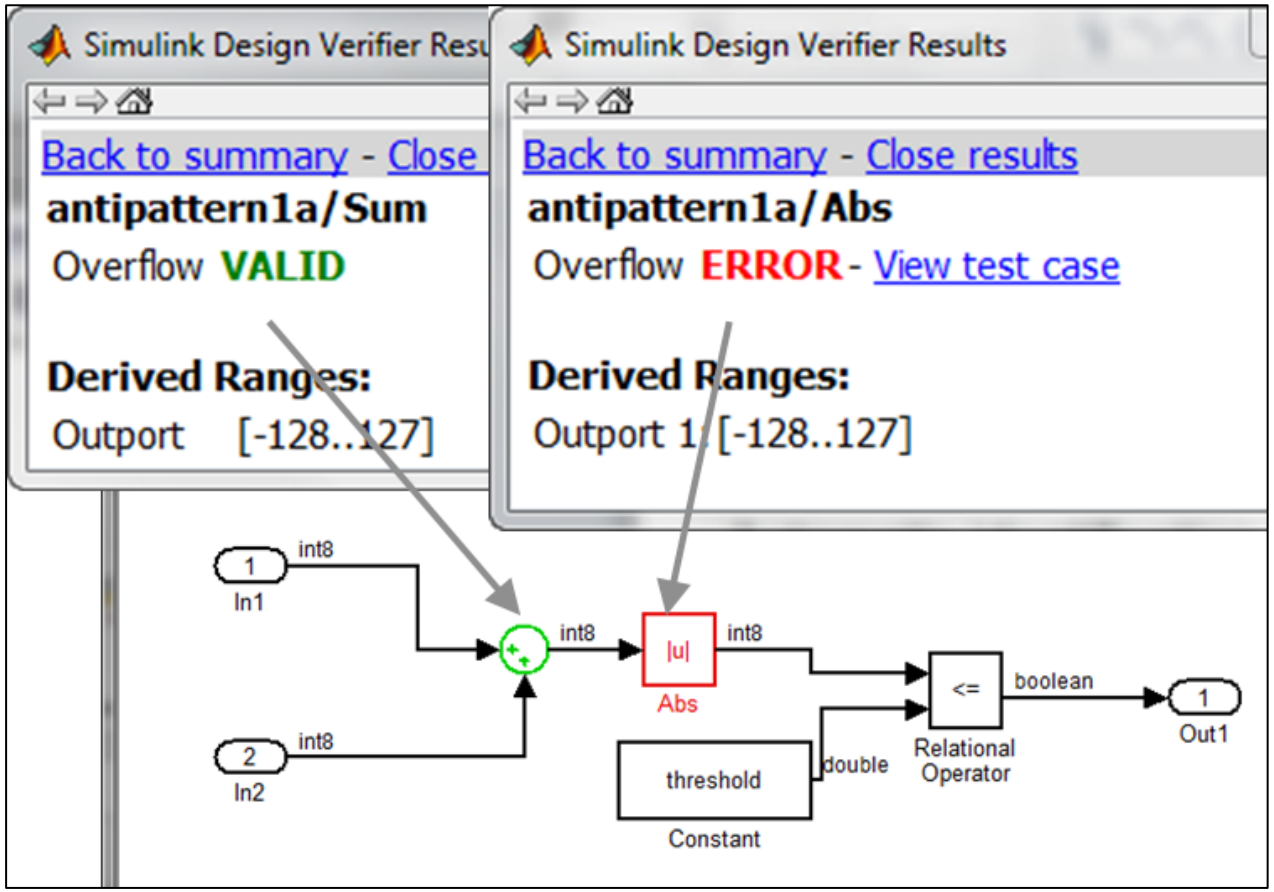


Assess Quality with Metrics Dashboard

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

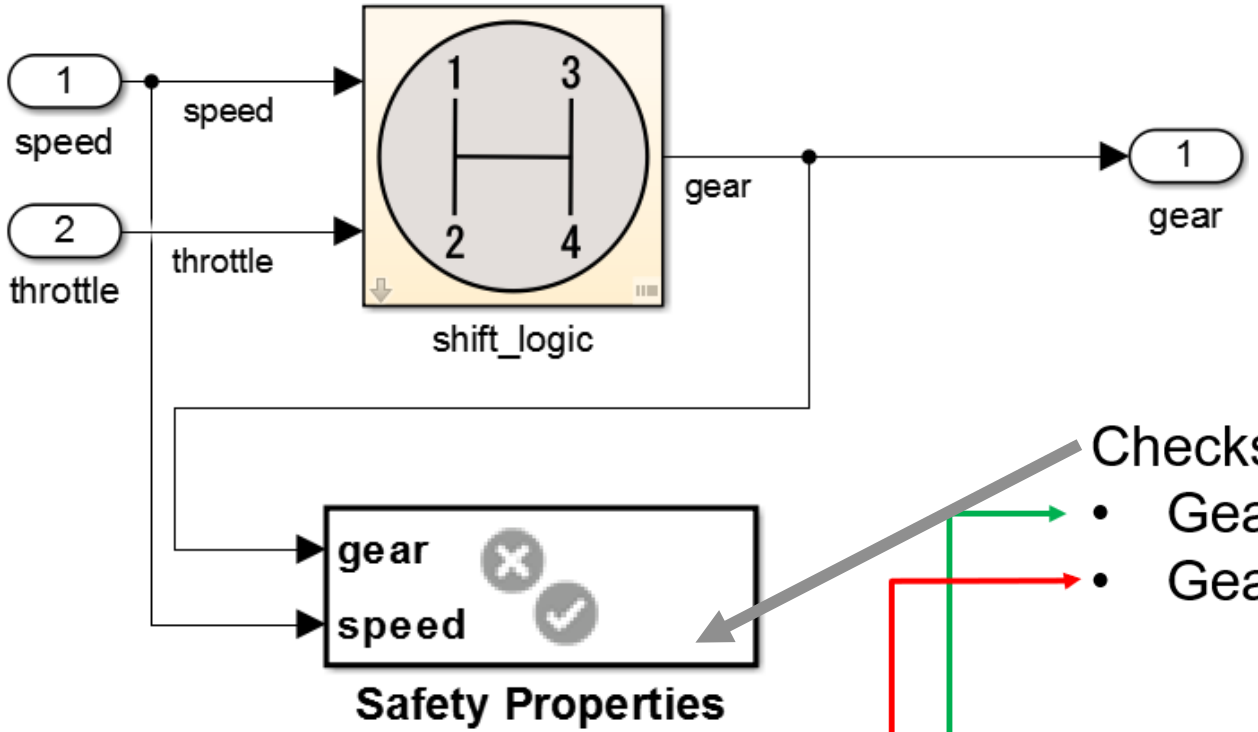


Detect Design Errors with Formal Methods



- 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

Checks that design meets requirements

- Gear 2 *always* engages when speed ≥ 5 and ≤ 25
- Gear 2 *never* engages when speed < 5 or > 25

Expected behavior of design

Behavior that design should not exhibit

Reviews, Static Analysis and Formal Verification at the Model Level

Standards & Guidelines Checks

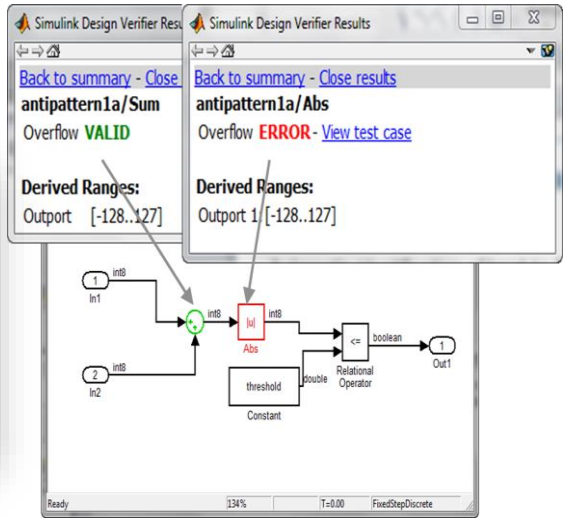
- **Automate** compliance to standards
- **Customize** checks
- **Find and fix** compliance issues while you design with Edit Time Checking

Model Metrics

- **Analyze** complexity, size, reusability
- **Assess** design quality

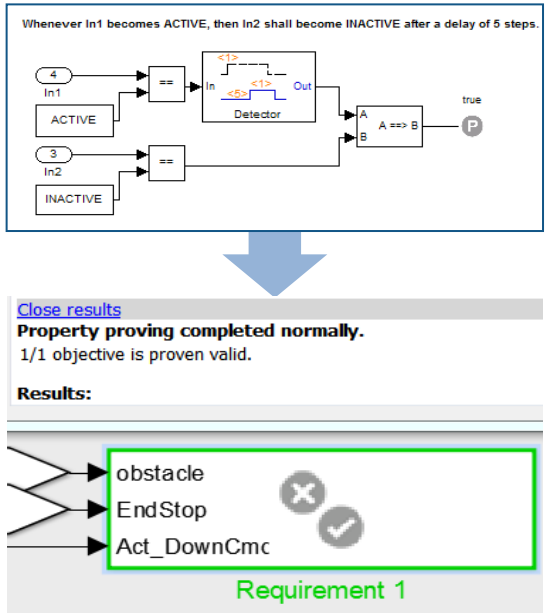
Design Error Detection

Uncover hard to find dead logic and design flaws

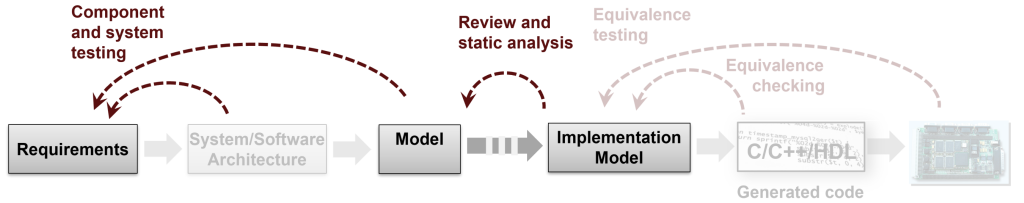


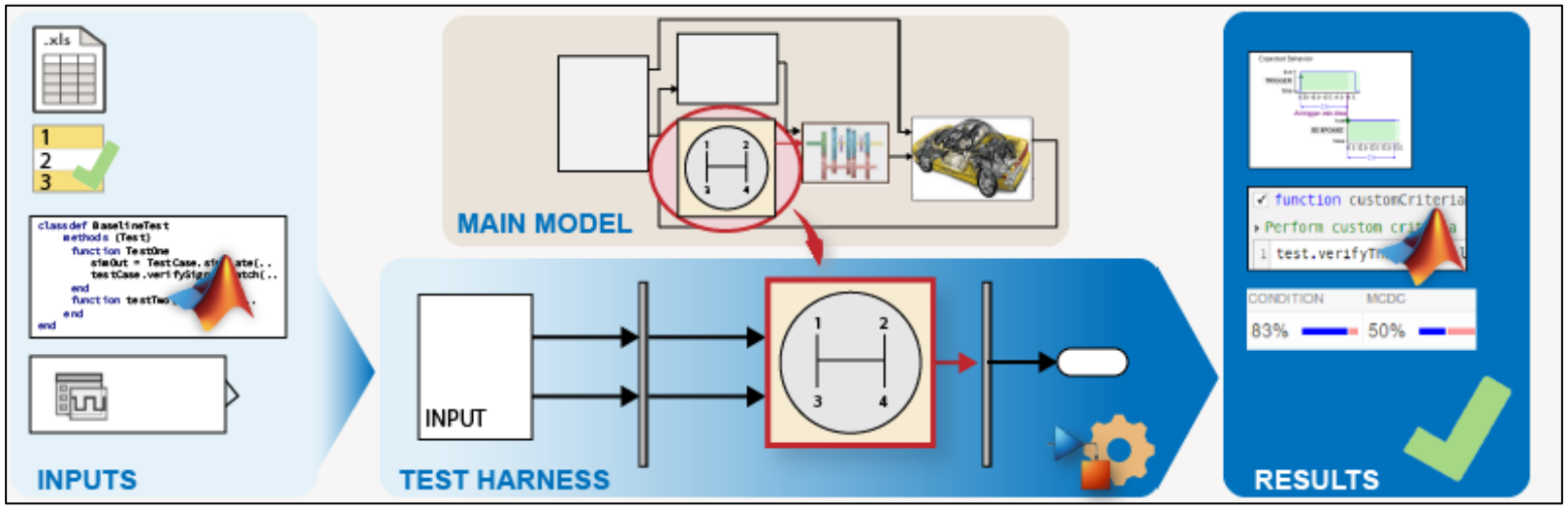
Property Proving

Prove design meets requirements



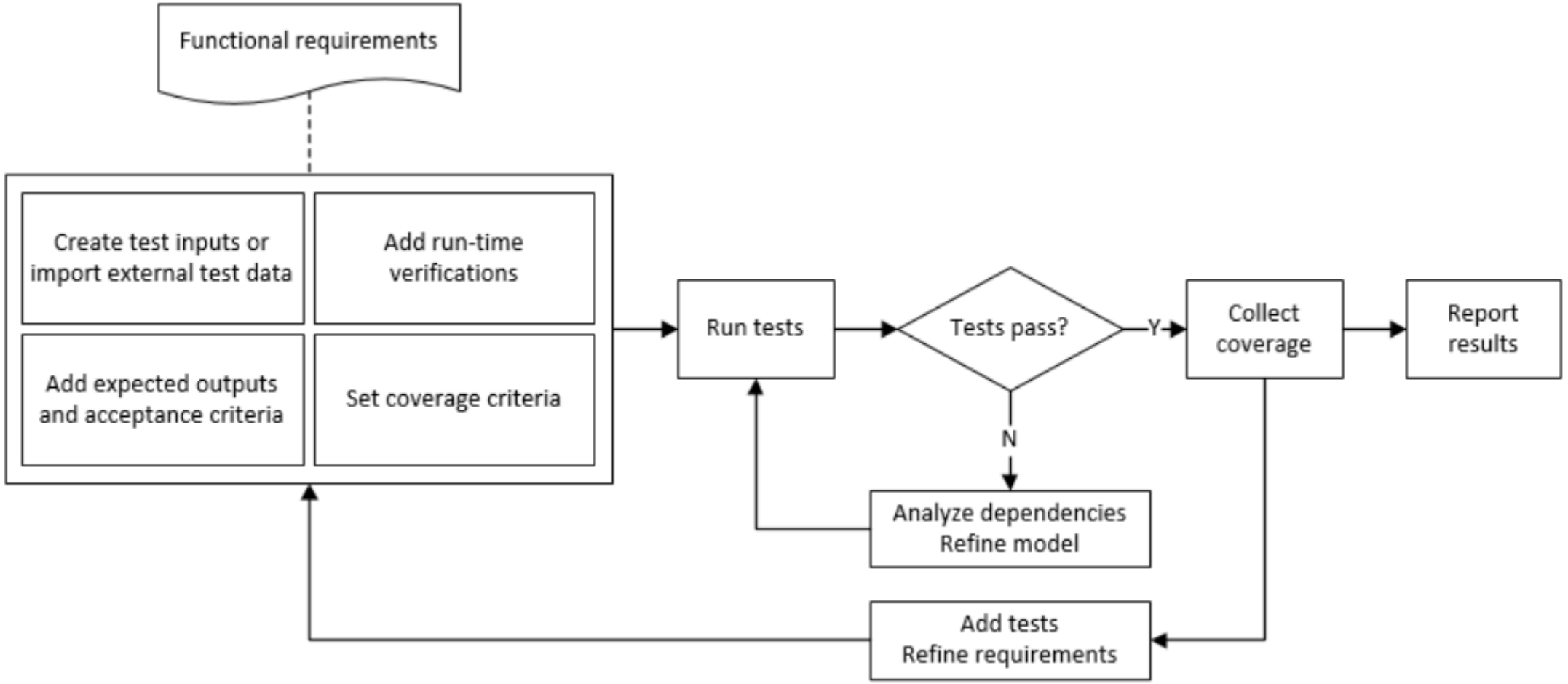
Systematic Functional Testing





Does the design meet requirements?
Is it functioning correctly?
Is it completely tested?

Typical Functional Testing Workflow in Model Based Design

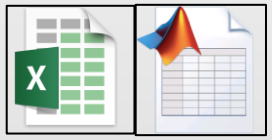


Systematic Functional Testing with Simulink Test

Test Case

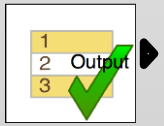
Model Sim through SIL, PIL and HIL
Scale with Parallel Computing Toolbox and Continuous Integration

Inputs

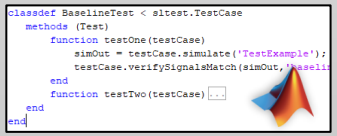


Data file (input)

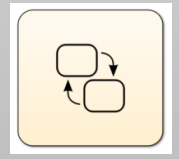
Signal Editor



Test Sequence

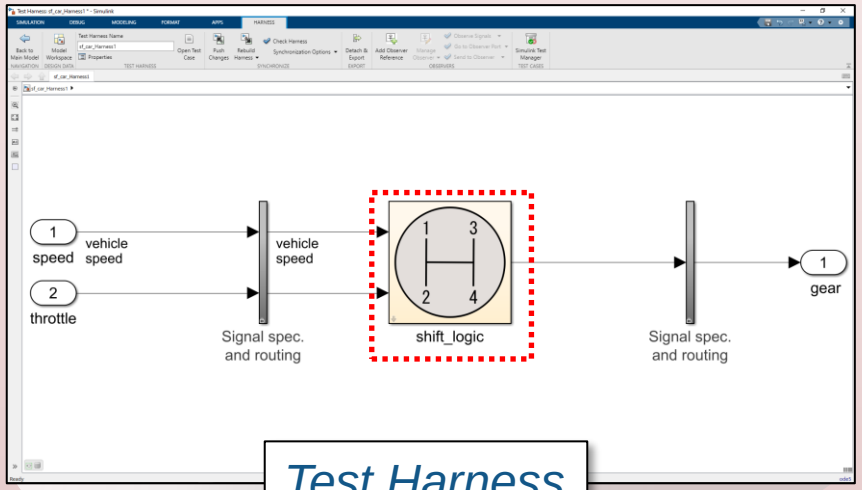


MATLAB Code

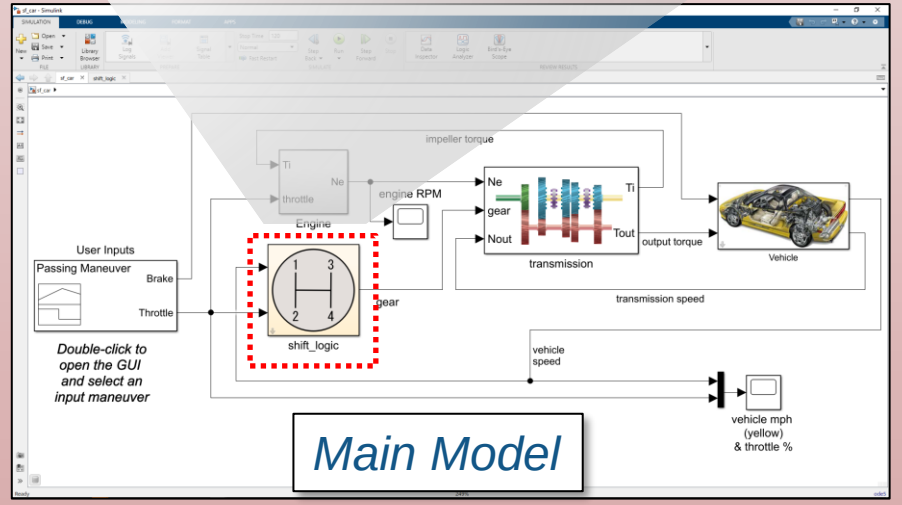


Stateflow

and more!

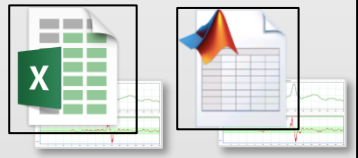


Test Harness

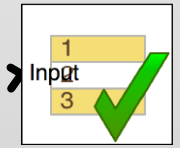


Main Model

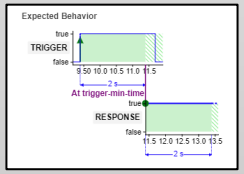
Assessments



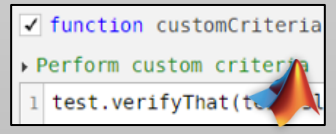
Data file baseline)



Test Assessment



Temporal Assessment



MATLAB Code

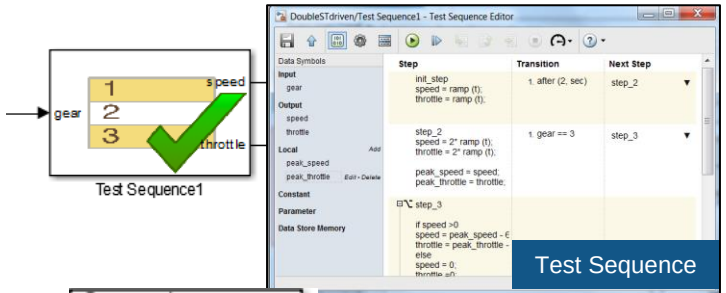
and more!

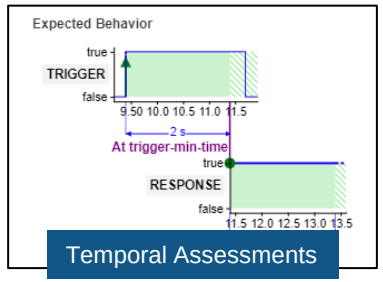
Simulink Test

Develop, manage, and execute simulation-based tests

Test Authoring

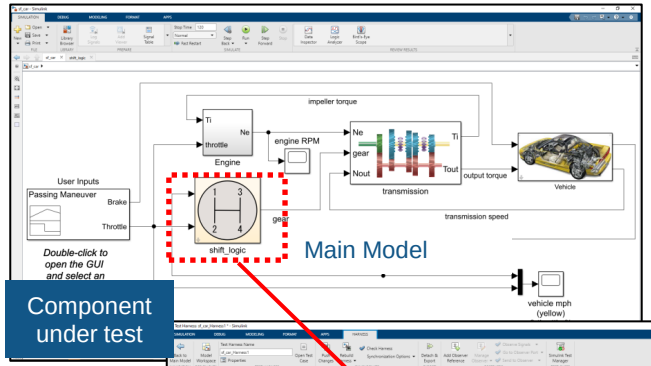
- Specify test inputs, expected outputs, and tolerances
- Construct complex test sequences and assessments

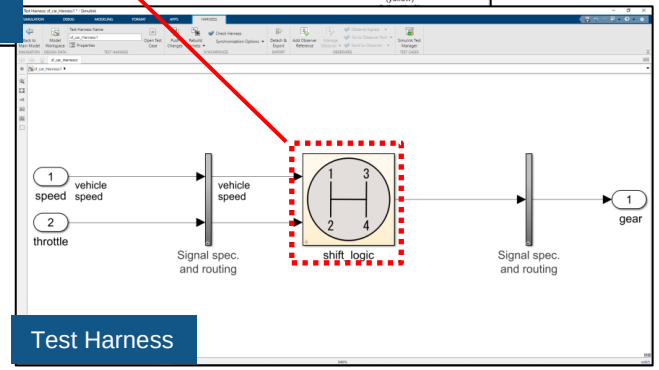




Test Harnesses

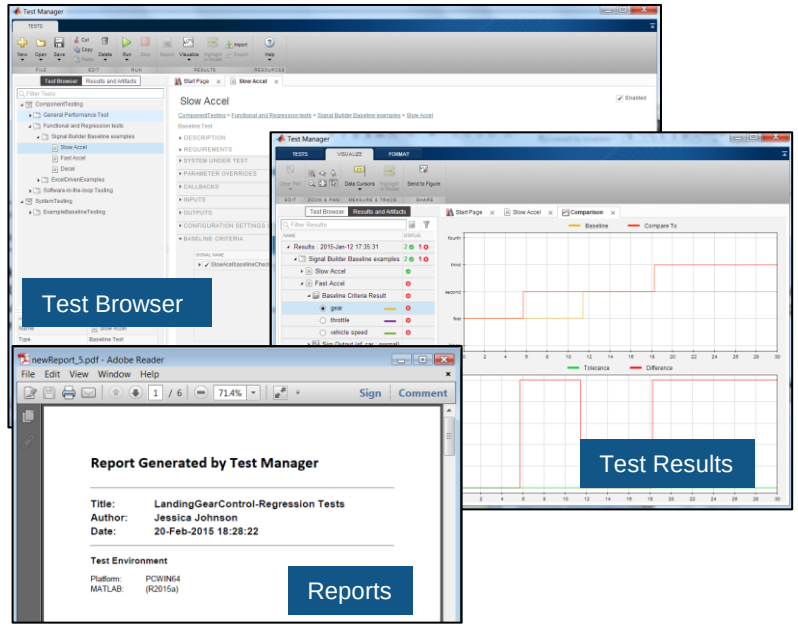
- Isolate Component Under Test
- Synchronized, simulation test environment



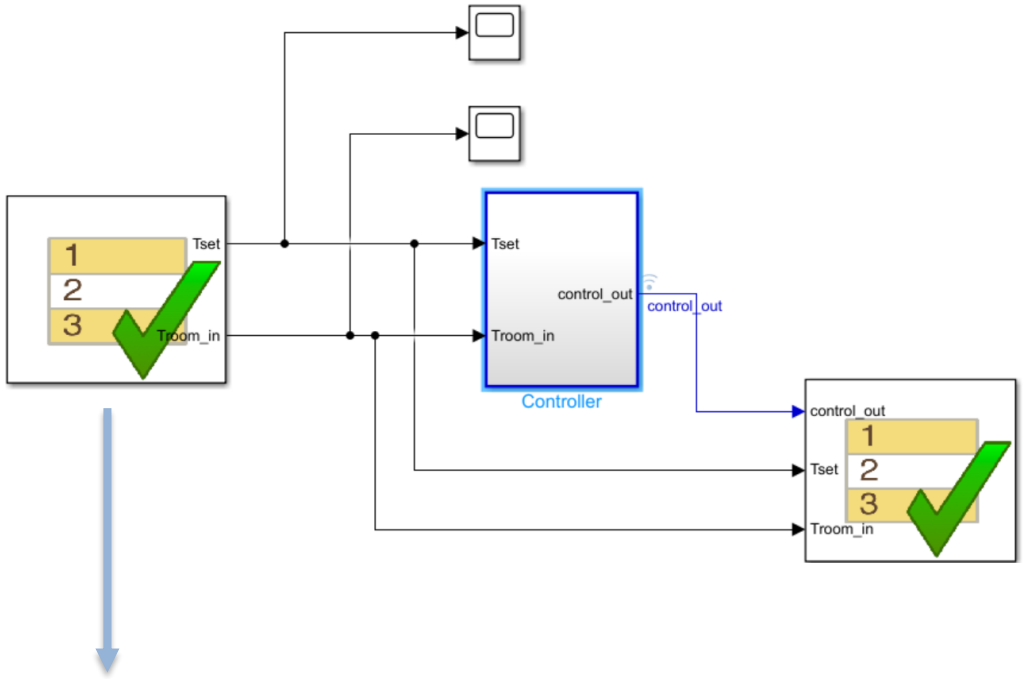


Test Manager

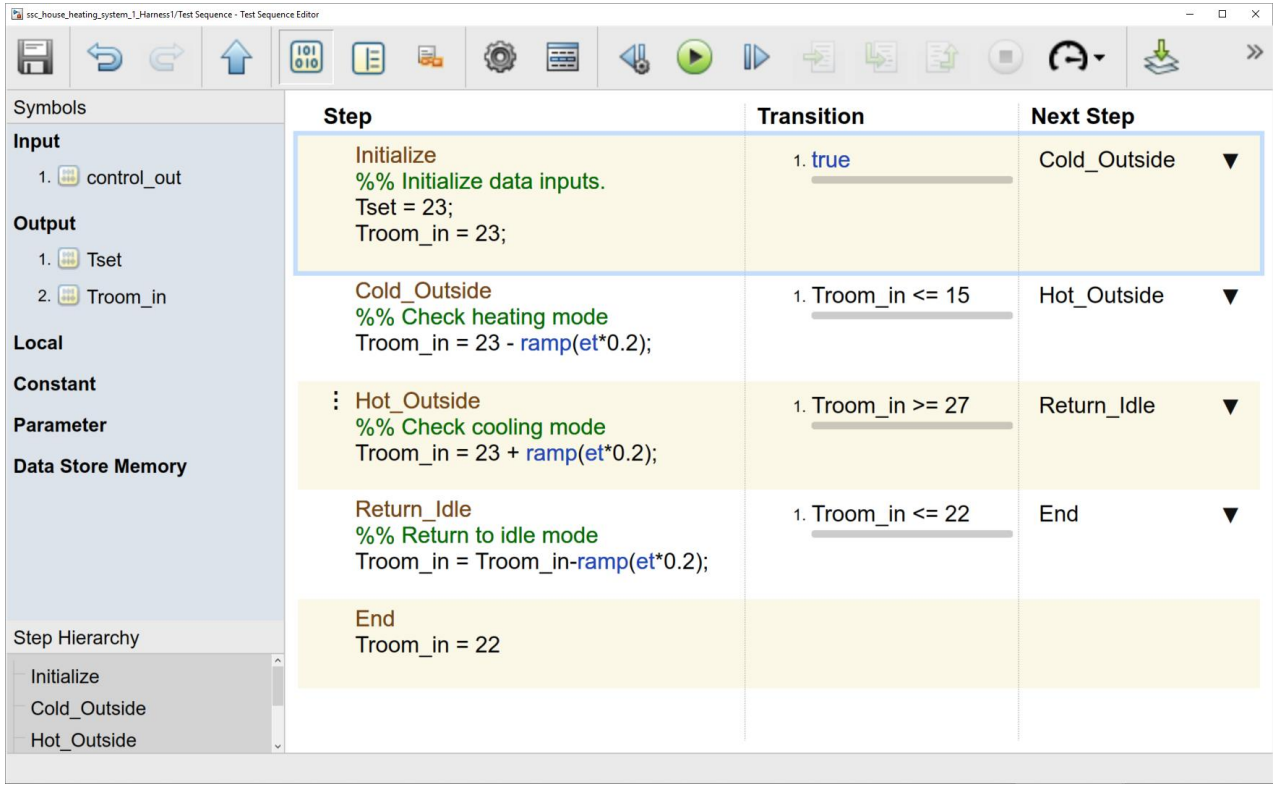
- Author, manage, organize tests
- Execute simulation, equivalence and baseline tests
- Review, export, report

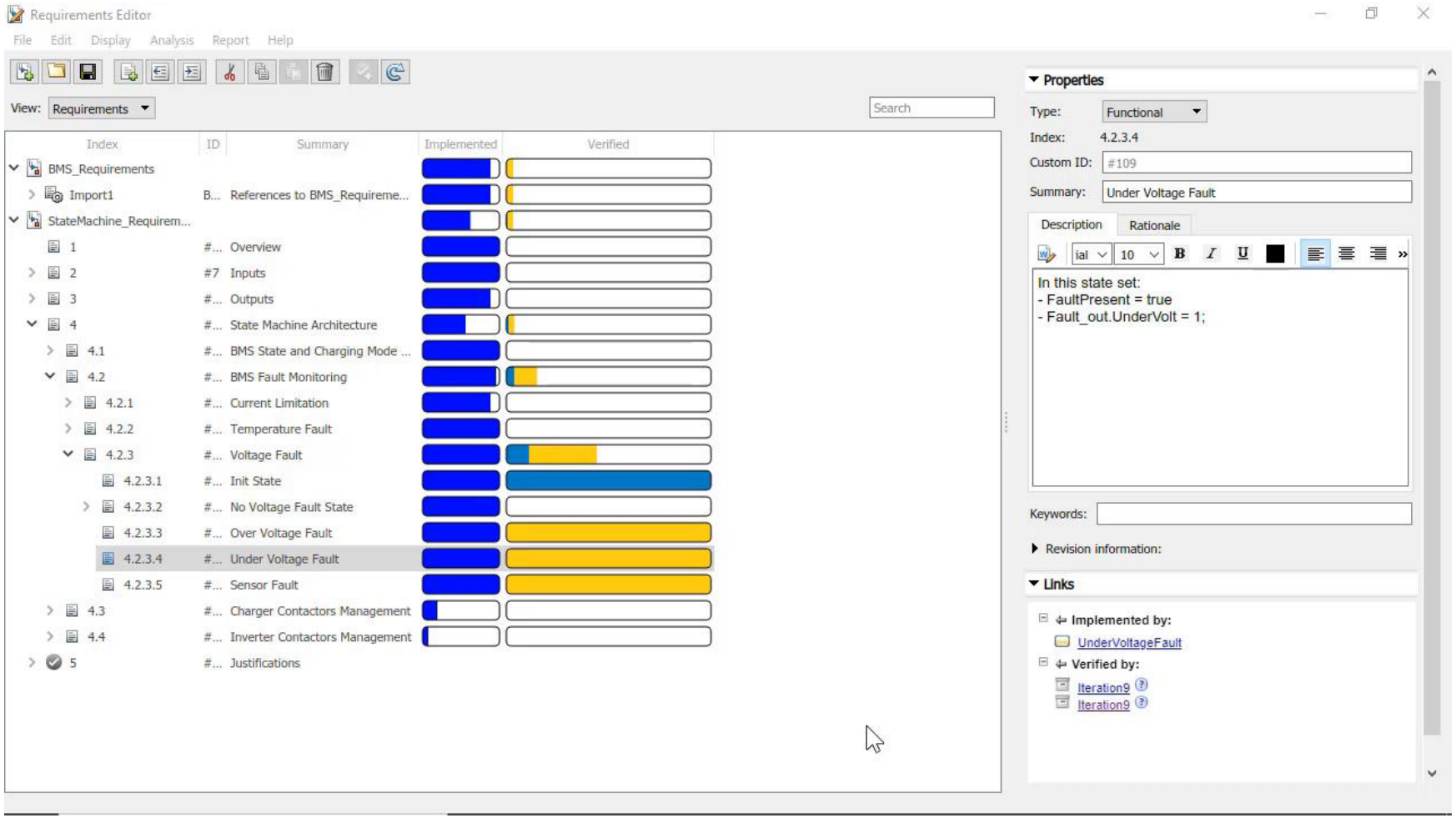


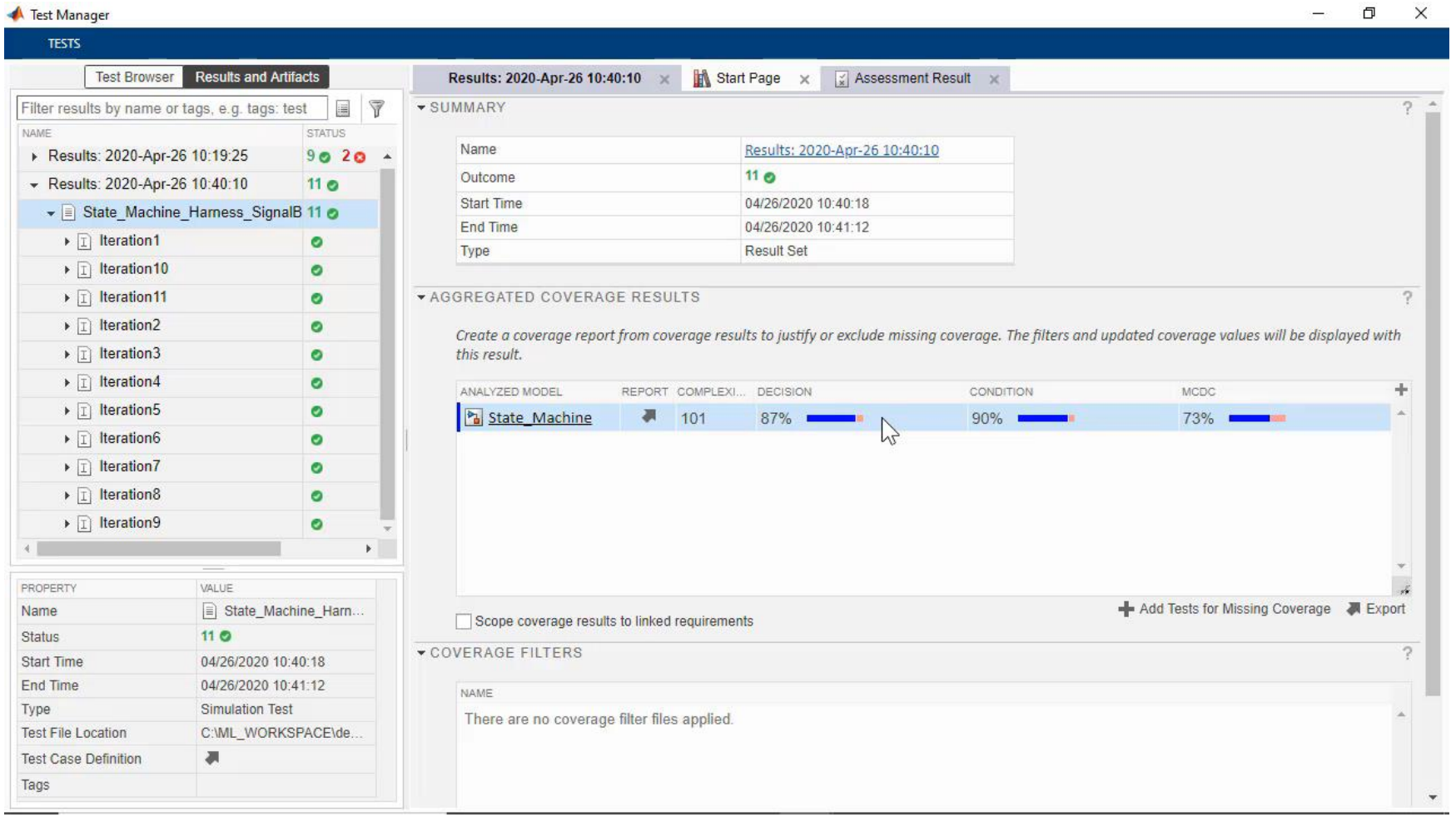
Test Sequence Block: Step-based and temporal test sequences



Test Sequence Block




Requirements Editor
File Edit Display Analysis Report Help
View: Requirements
Search
Index ID Summary Implemented Verified
BMS_Requirements
Import1 B... References to BMS_Requireme...
StateMachine_Requirem...
1 #... Overview
2 #7 Inputs
3 #... Outputs
4 #... State Machine Architecture
4.1 #... BMS State and Charging Mode ...
4.2 #... BMS Fault Monitoring
4.2.1 #... Current Limitation
4.2.2 #... Temperature Fault
4.2.3 #... Voltage Fault
4.2.3.1 #... Init State
4.2.3.2 #... No Voltage Fault State
4.2.3.3 #... Over Voltage Fault
4.2.3.4 #... Under Voltage Fault
4.2.3.5 #... Sensor Fault
4.3 #... Charger Contactors Management
4.4 #... Inverter Contactors Management
5 #... Justifications
Properties
Type: Functional
Index: 4.2.3.4
Custom ID: #109
Summary: Under Voltage Fault
Description Rationale
In this state set:
- FaultPresent = true
- Fault_out.UnderVolt = 1;
Keywords:
Revision information:
Links
Implemented by:
UnderVoltageFault
Verified by:
Iteration9
Iteration9

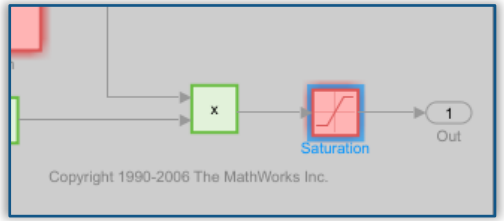

Test Manager
TESTS
Test Browser
Results and Artifacts
Filter results by name or tags, e.g. tags: test
NAME
STATUS
Results: 2020-Apr-26 10:19:25
9 2
Results: 2020-Apr-26 10:40:10
11
State_Machine_Harness_SignalB 11
Iteration1
Iteration10
Iteration11
Iteration2
Iteration3
Iteration4
Iteration5
Iteration6
Iteration7
Iteration8
Iteration9
PROPERTY
VALUE
Name
State_Machine_Harn...
Status
11
Start Time
04/26/2020 10:40:18
End Time
04/26/2020 10:41:12
Type
Simulation Test
Test File Location
C:\ML_WORKSPACE\de...
Test Case Definition
Tags
Results: 2020-Apr-26 10:40:10
Start Page
Assessment Result
SUMMARY
Name
Results: 2020-Apr-26 10:40:10
Outcome
11
Start Time
04/26/2020 10:40:18
End Time
04/26/2020 10:41:12
Type
Result Set
AGGREGATED COVERAGE RESULTS
Create a coverage report from coverage results to justify or exclude missing coverage. The filters and updated coverage values will be displayed with this result.
ANALYZED MODEL
REPORT
COMPLEXI...
DECISION
CONDITION
MCDC
State_Machine
101
87%
90%
73%
Add Tests for Missing Coverage
Export
Scope coverage results to linked requirements
COVERAGE FILTERS
NAME
There are no coverage filter files applied.

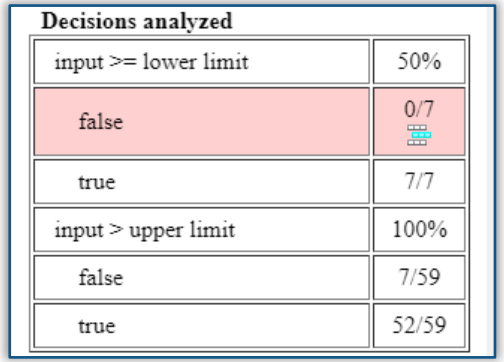
Simulink Coverage

Measure test coverage in models and generated code

Model Coverage

- Measure test completeness
- Identify missing tests or unintended functionality

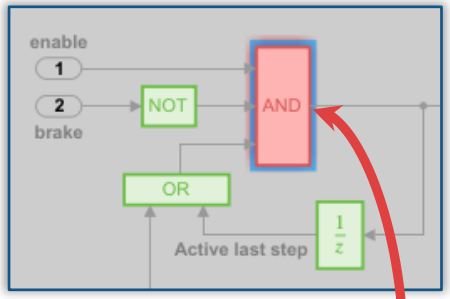




Decisions analyzed	
input >= lower limit	50%
false	0/7
true	7/7
input > upper limit	100%
false	7/59
true	52/59

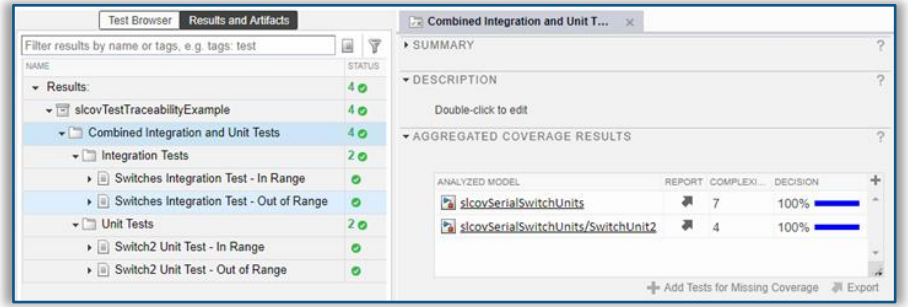
Generated Code Coverage

- Find untested generated code
- Map results from code to model object

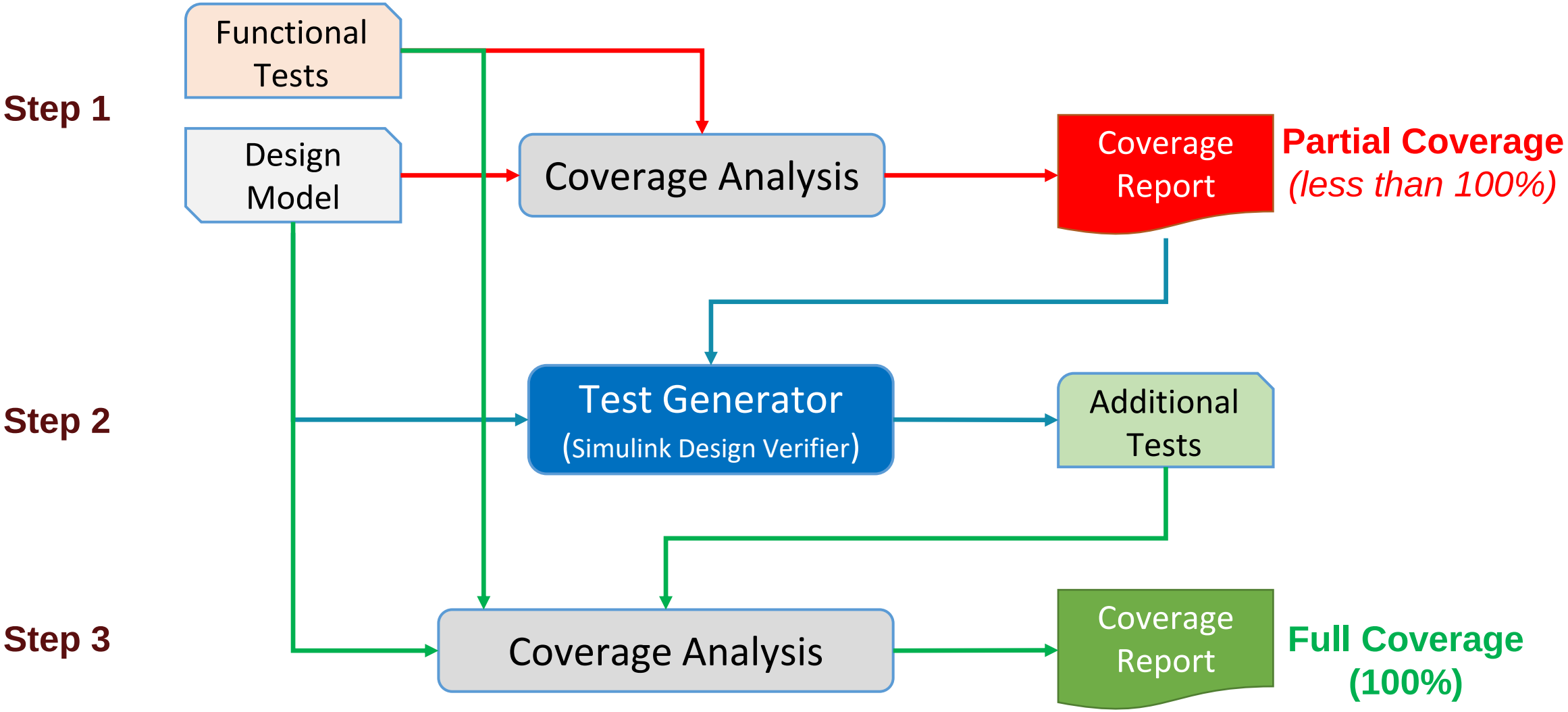


Highlighting and Reporting

- View coverage results on diagrams
- Manage coverage results in **Simulink Test Manager**



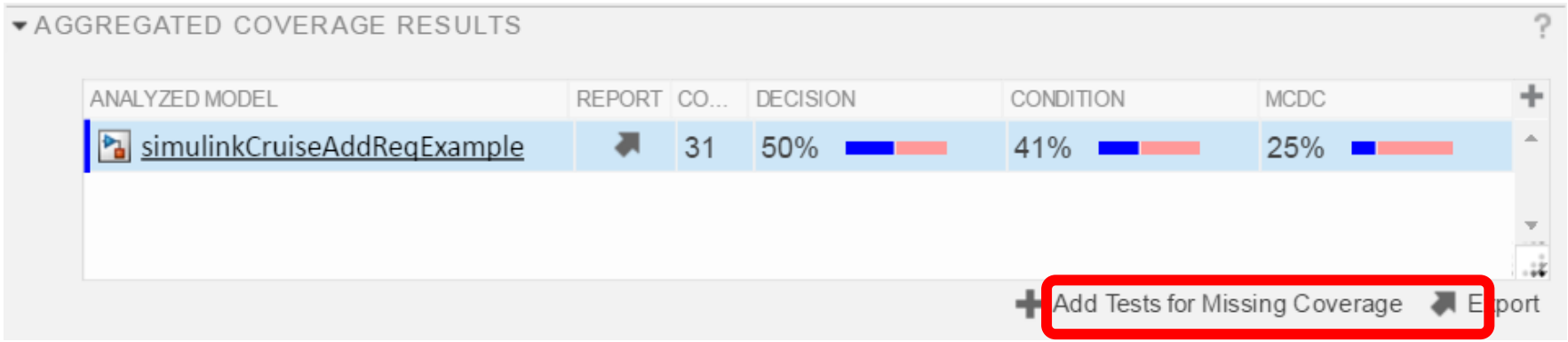
Addressing Missing Coverage



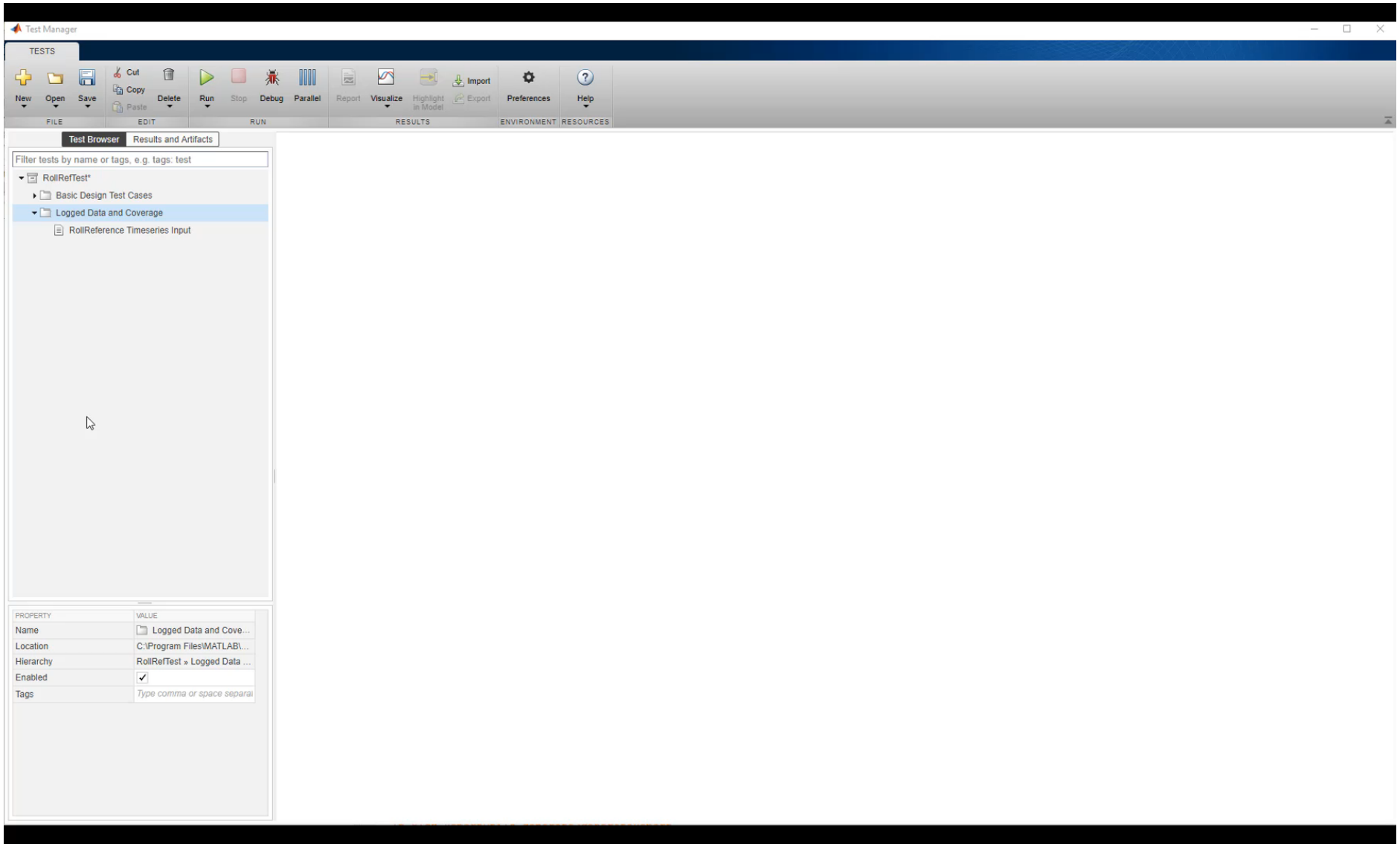
Automatically Address Missing Coverage

Generate additional tests automatically using Simulink Design Verifier from within the Test Manager to increase coverage

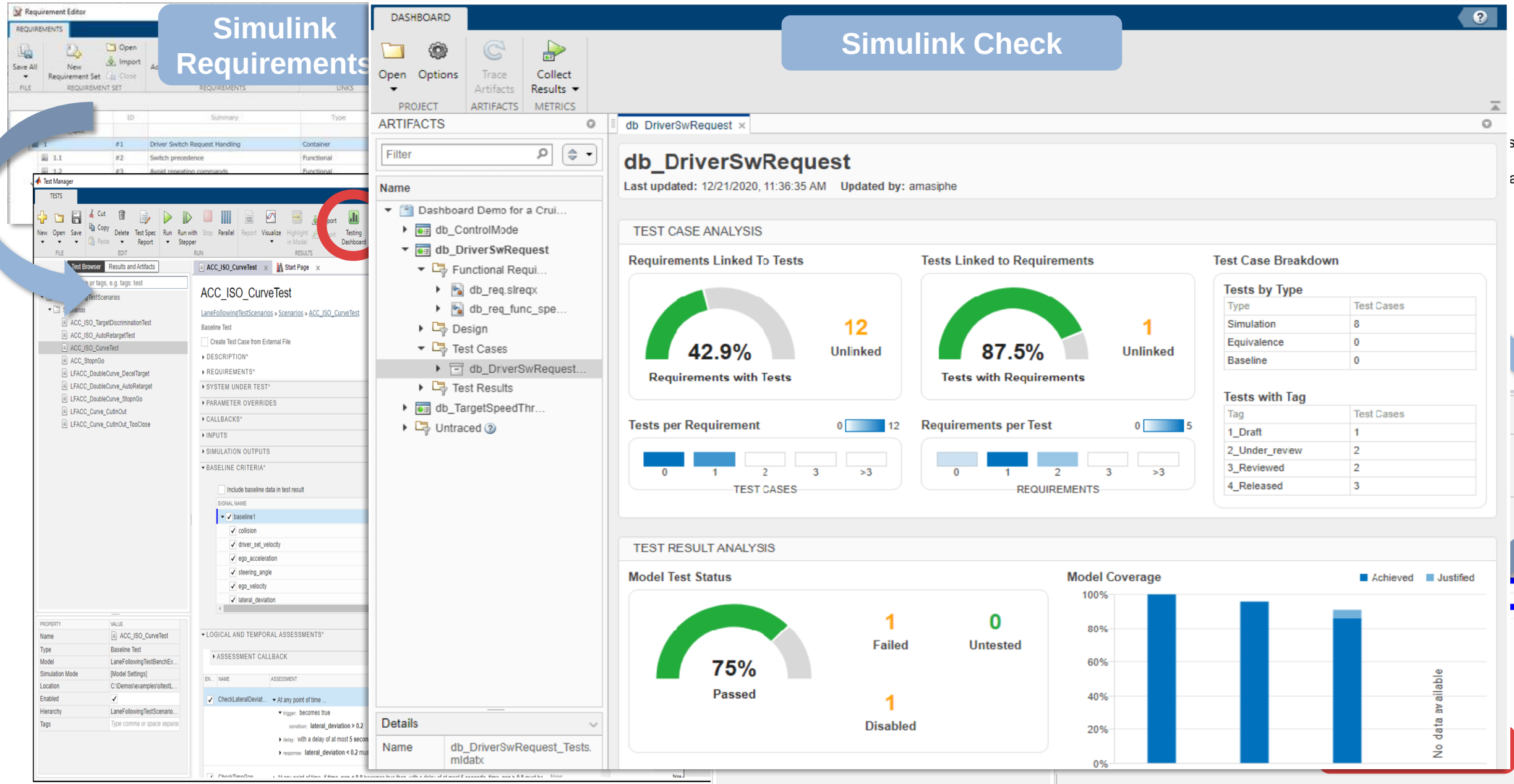
- View coverage results in the Test Manager for existing tests
- Select coverage results and click Add Tests for Missing Coverage



Demo: Generate Tests for Coverage from Test Manager

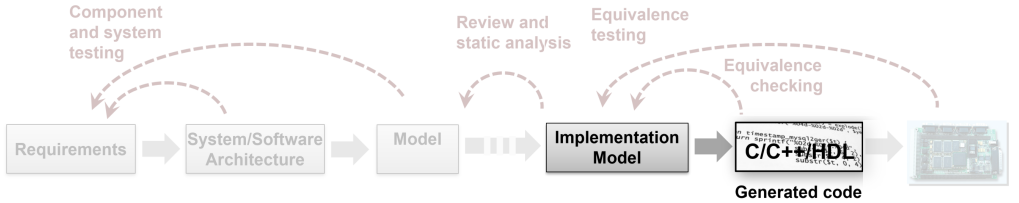


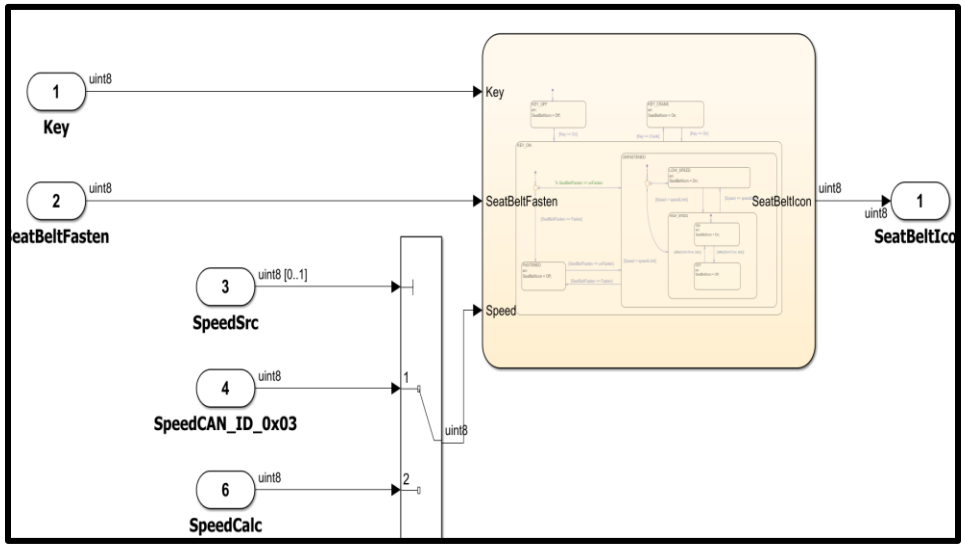
Simulink Test Manager – Integrates MathWorks V&V tools



Automatic Code Generation

Reliable and high performance, with flexible choice of targets



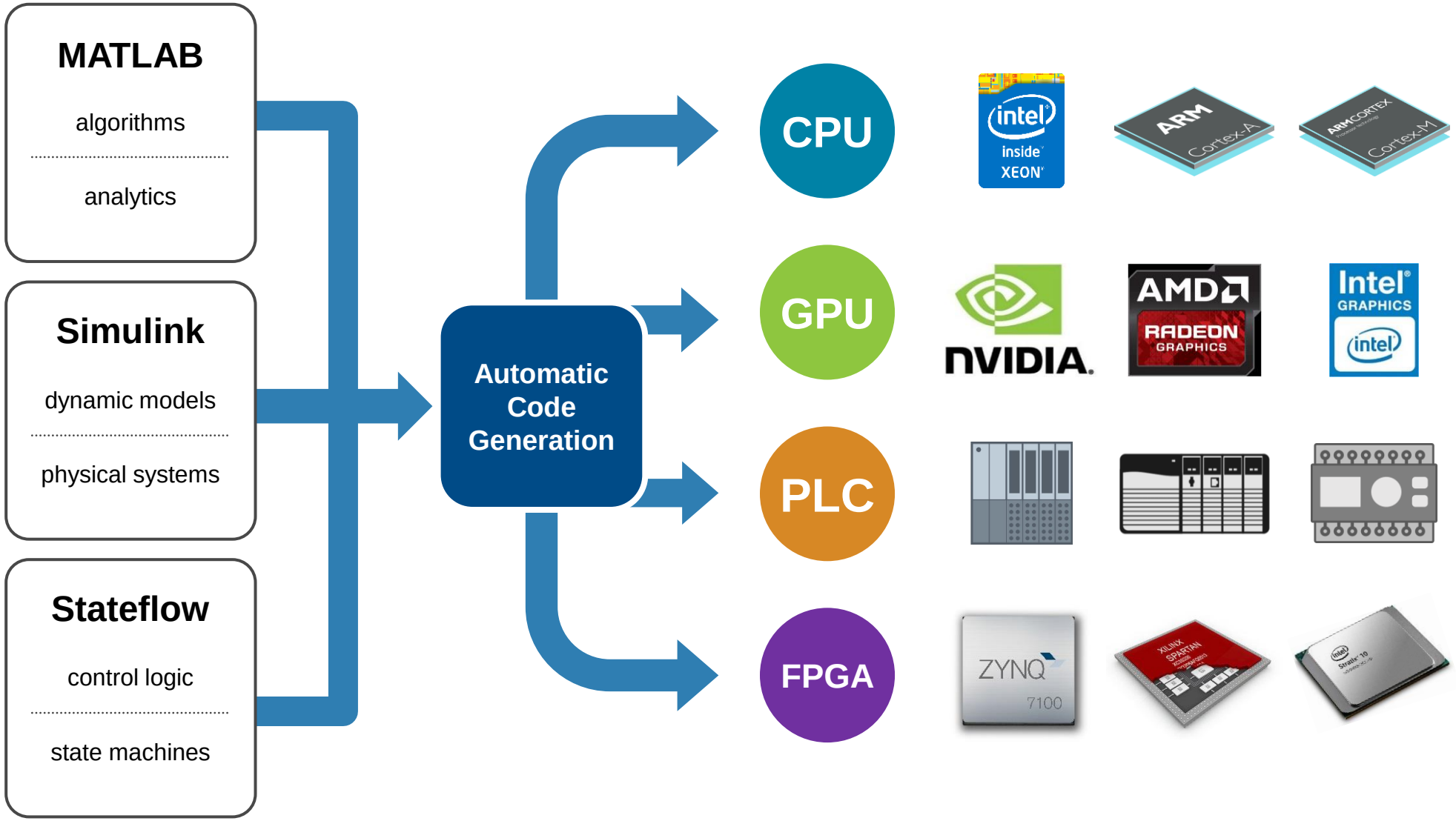


Simulink Model
Application Logic

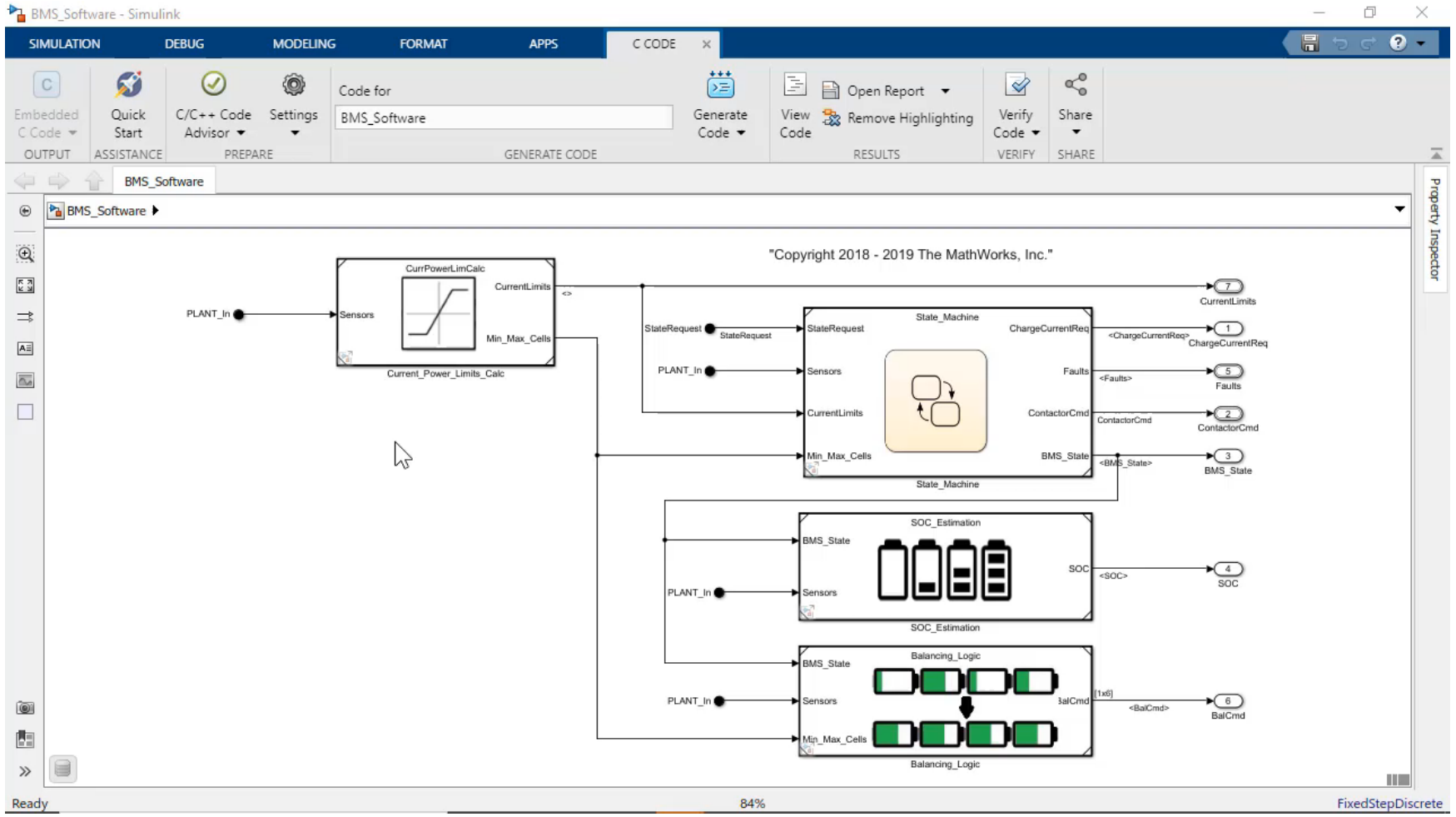
Efficient
C/C++

Automatic Code Generation

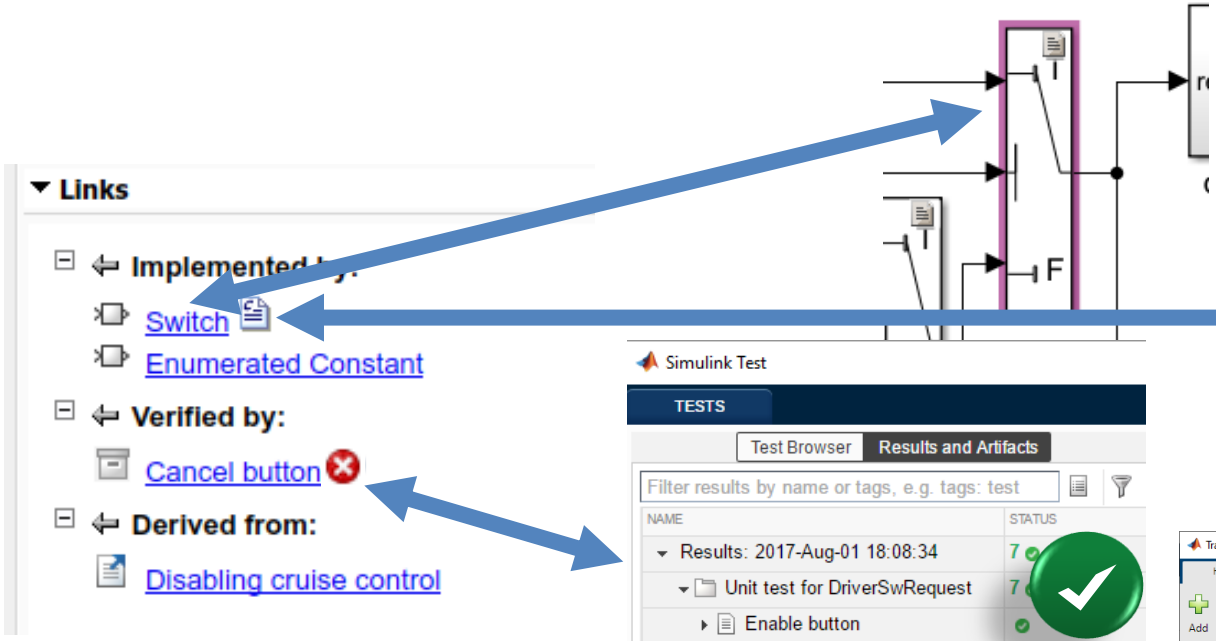
Reliable and high performance, with flexible choice of targets



Automatic Code Generation

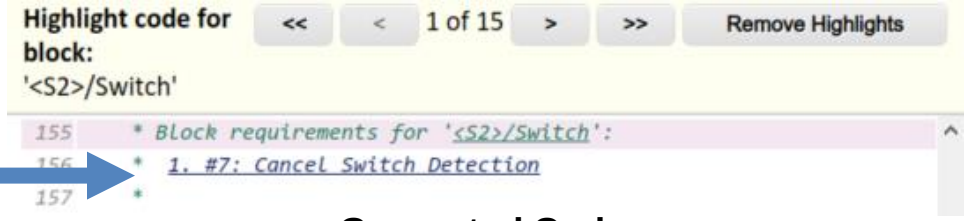


Requirements Traceability to Model, Code and Test Cases



Requirements

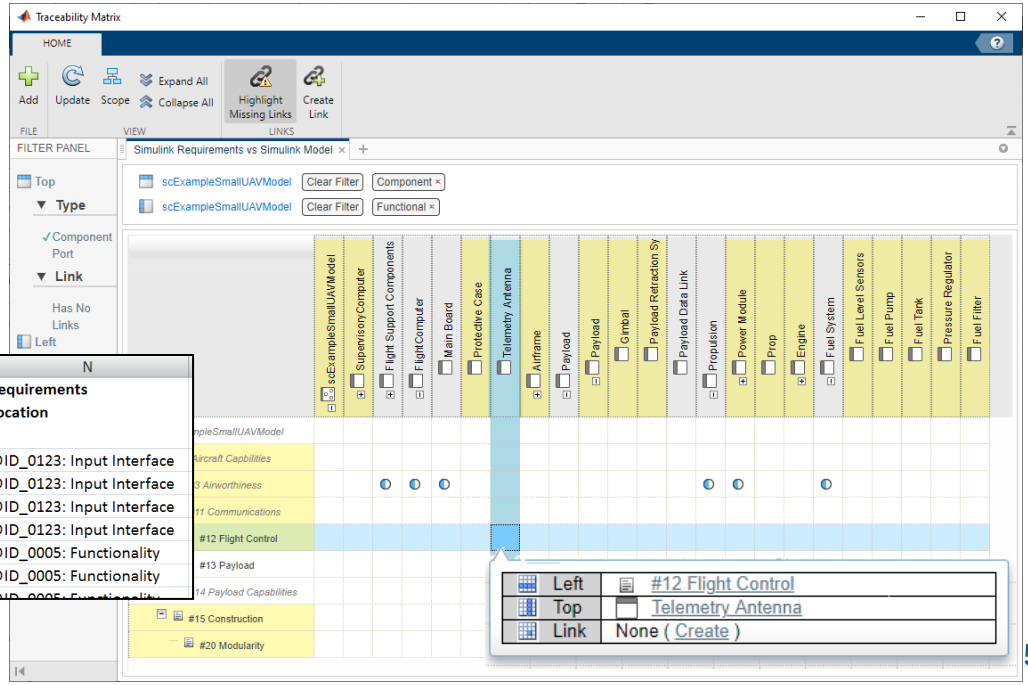
Tests & Test Results

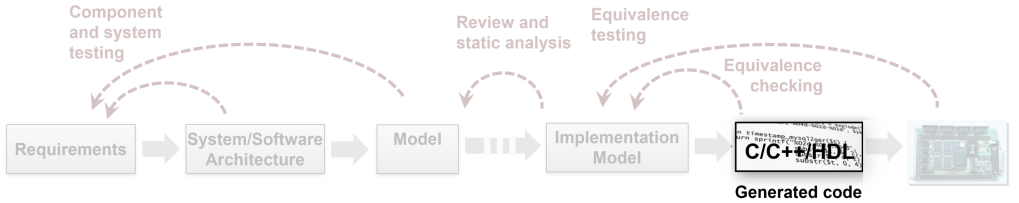


Generated Code

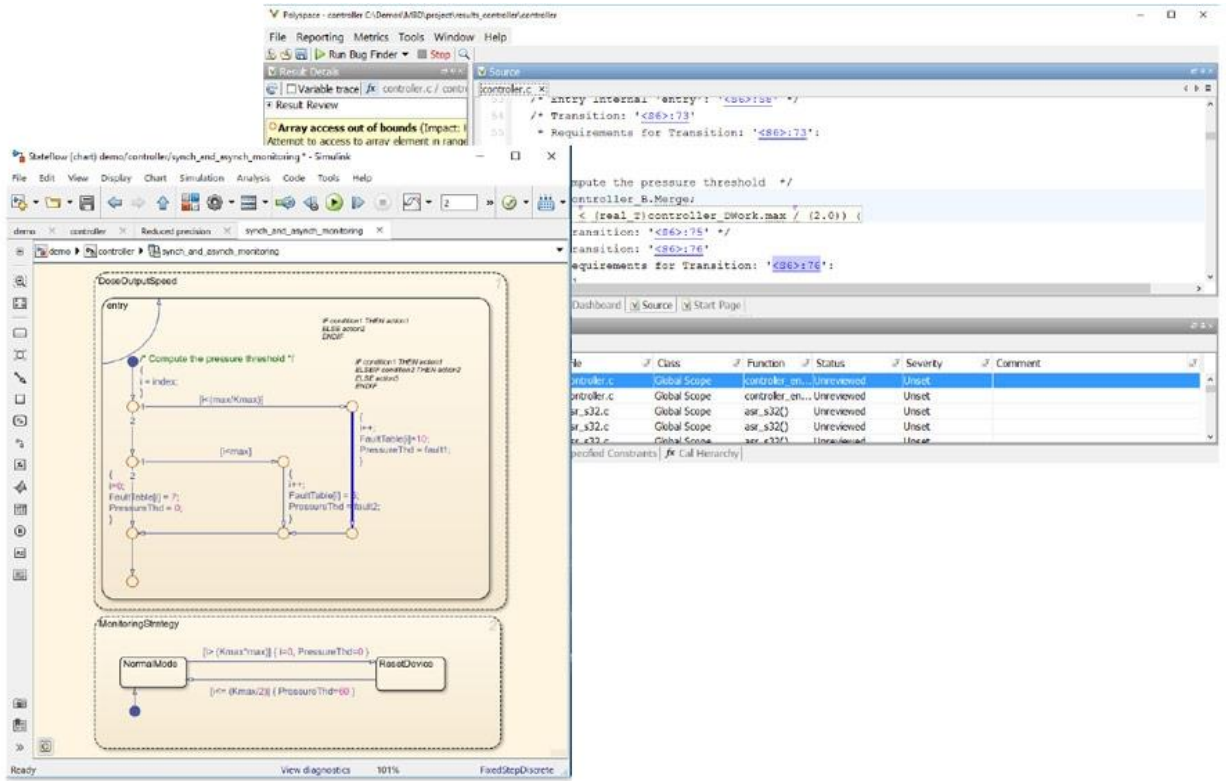
	B	C	D	E	F	G	H	I	J	K	L	M	N
1	Model Object Type	Model Object Path	Model Object Subsystem	Model Object SID	Model Object Optimized	Model Optimization Rationale	Code File Name	Code Function	Code Line Number	Code File Location	Code Comment Checksum	Requirements Source	Requirements Location
2	Inport	model1	model1	model1:4			model1.c	Global	22	C:\work\model1_ert_rtw	CS_533947273	C:\work\req1.doc	@ID_0123: Input Interface
3	Inport	model1	model1	model1:4			model1.c	model1_step	38	C:\work\model1_ert_rtw	CS_116158590	C:\work\req1.doc	@ID_0123: Input Interface
4	Inport	model1	model1	model1:4			model1.c	model1_step	43	C:\work\model1_ert_rtw	CS_116158590	C:\work\req1.doc	@ID_0123: Input Interface
5	Inport	model1	model1	model1:4			model1.h	Global	49	C:\work\model1_ert_rtw	CS_533947273	C:\work\req1.doc	@ID_0123: Input Interface
6	Gain	model1	model1	model1:2			model1.c	Global	23	C:\work\model1_ert_rtw	CS_405027874	C:\work\req1.doc	@ID_0005: Functionality
7	Gain	model1	model1	model1:2			model1.c	Global	27	C:\work\model1_ert_rtw	CS_104796431	C:\work\req1.doc	@ID_0005: Functionality

Traceability Matrix



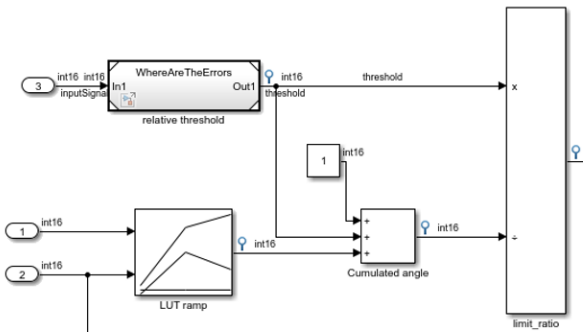


Static Code Analysis with Polyspace



Polyspace is independent of the origin of code

```
#include <assert.h>
int speed(int k)
{
  int i,j,v;
  i = 2;
  j = k+5;
  while (i < 10) {
   i++;
   j+=3;
  }
  return 1 / (i-j);
}
```



Hand Code

C, C++

Polyspace

Violations
Defects
Runtime errors
Reports

C, C++

Model-Based Design
(MATLAB, Simulink, Stateflow)

Model-Based V&V tools
Code Generation tools

Can you find a bug?

```
1  int new_position(int sensor_pos1, int sensor_pos2)
2  {
3  int actuator_position;
4  int x, y, tmp, magnitude;
5
6  actuator_position = 2; /* default */
7  tmp = 0;               /* values */
8  magnitude = sensor_pos1 / 100;
9  y = magnitude + 5;
10
11 while (actuator_position < 10)
12     {
13     actuator_position++;
14     tmp += sensor_pos2 / 100;
15     y += 3;
16     }
17 if ((3*magnitude + 100) > 43)
18     {
19     magnitude++;
20     x = actuator_position;
21     actuator_position = x / (x - y);
22     }
23 return actuator_position*magnitude + tmp; /* new value */
24 }
```

Could there be a bug on this line?

Consider the operation: `x / (x - y)`

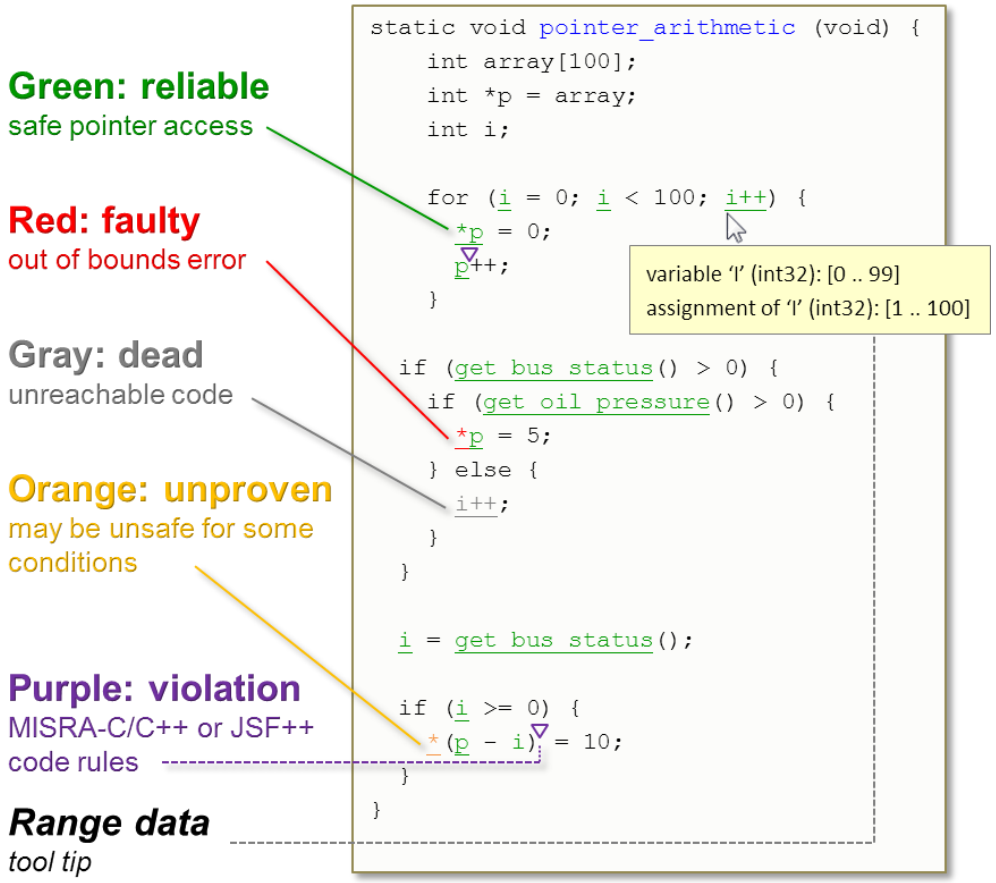
Potential run-time errors

- Variables x and y may not be initialized
- An overflow on subtraction
- If x == y, then a divide by zero will occur

How to prove that run-time errors *do* or *do not* exist?

Static Code Analysis with Polyspace

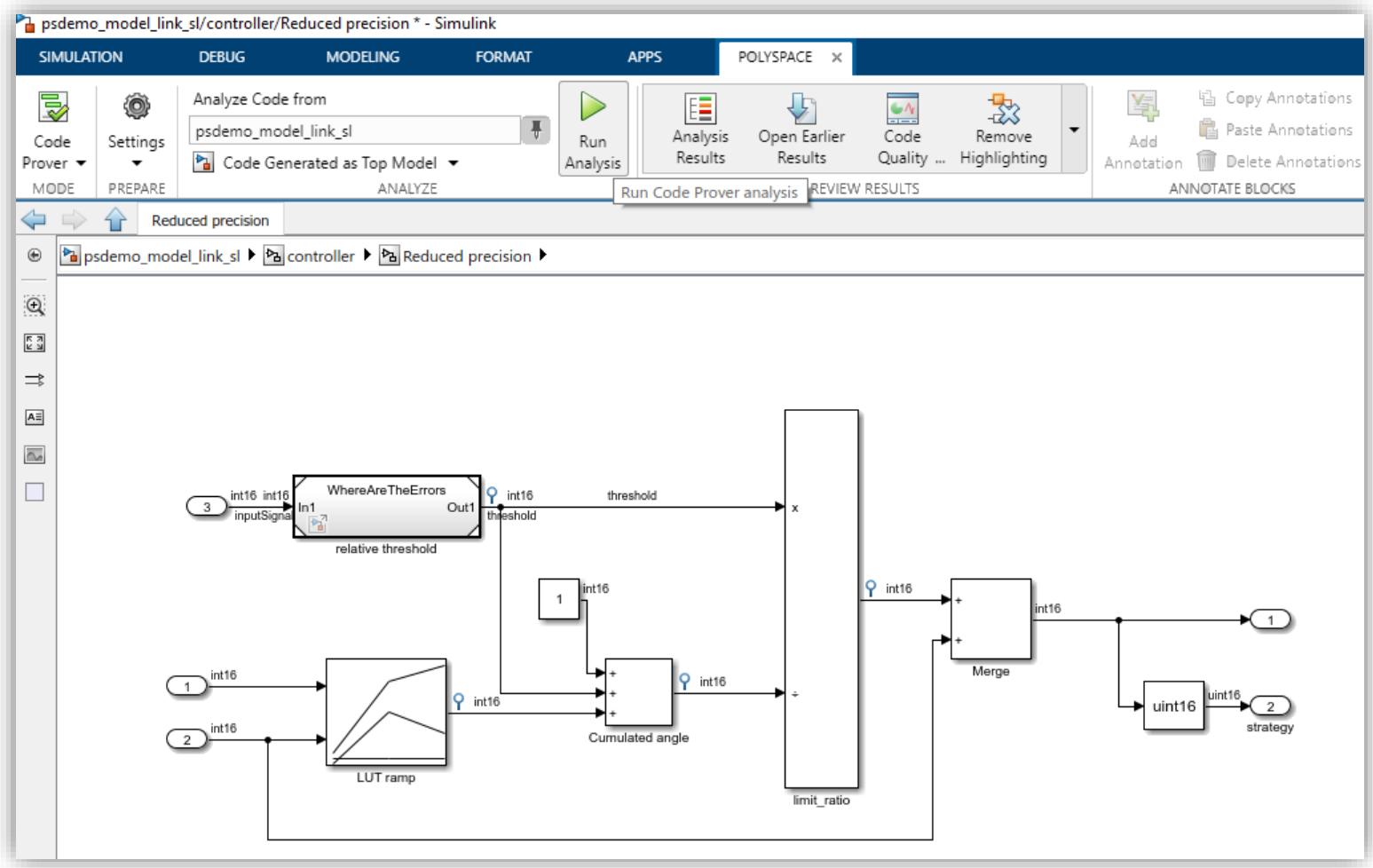
- Code metrics and standards
  - Comment density, cyclomatic complexity,…
  - MISRA and Cybersecurity standards
  - Support for DO-178, ISO 26262, ….
- Bug finding and code proving
  - Check data and control flow of software
  - Detect bugs and security vulnerabilities
  - Prove absence of runtime errors



Results from Polyspace Code Prover

Polyspace is Integrated with Simulink

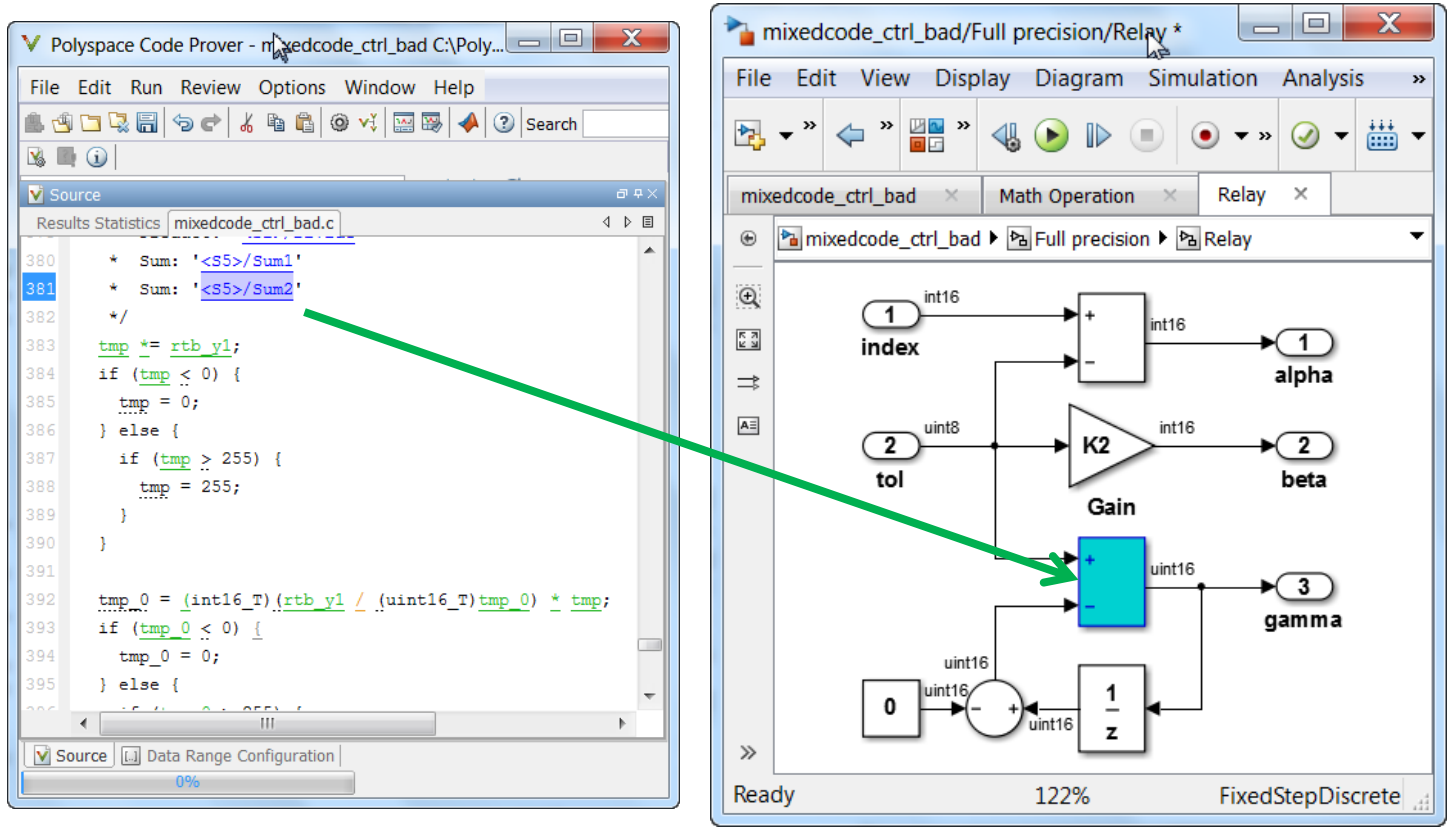
1. Launch Polyspace from Simulink



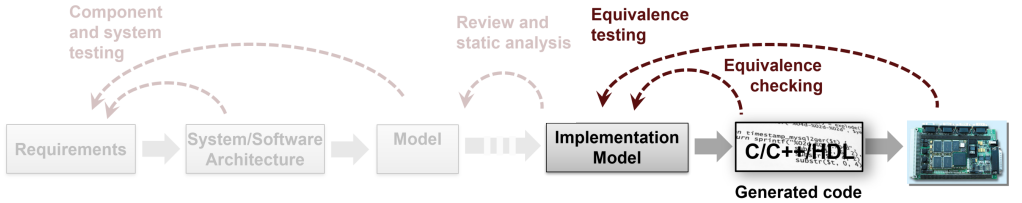
Polyspace is Integrated with Simulink

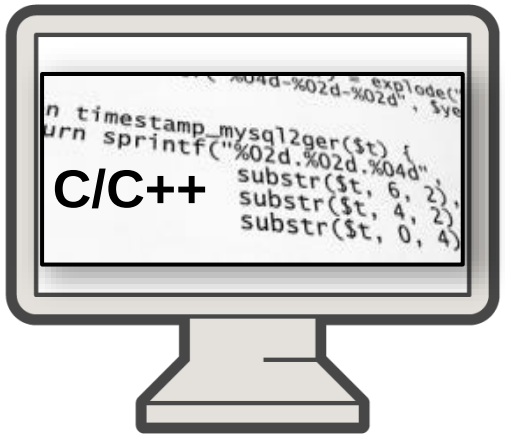
1. Launch Polyspace from Simulink

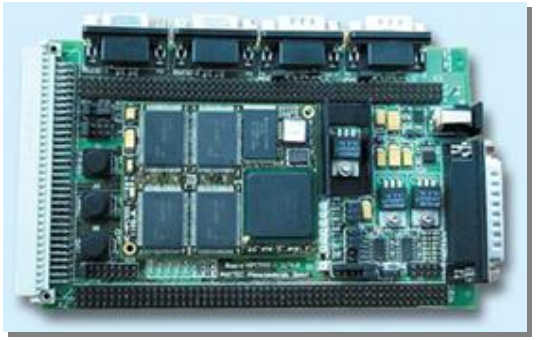
2. Navigate from Code to Model



Equivalence Testing







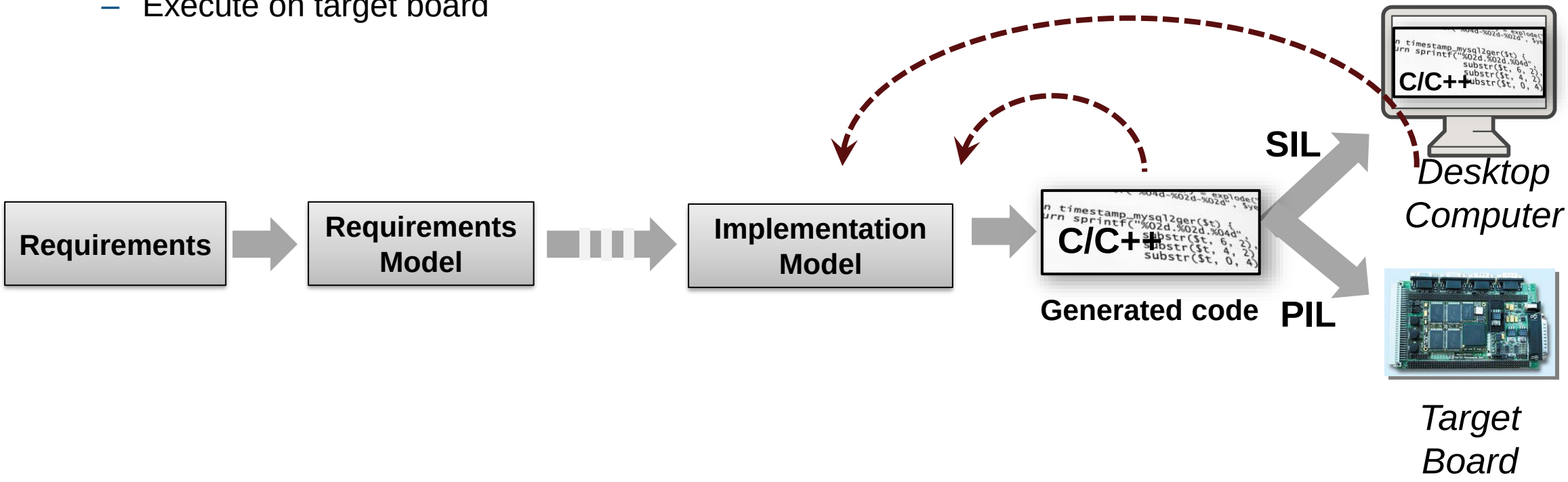
Software in the Loop (SIL)

Processor in the Loop (PIL)

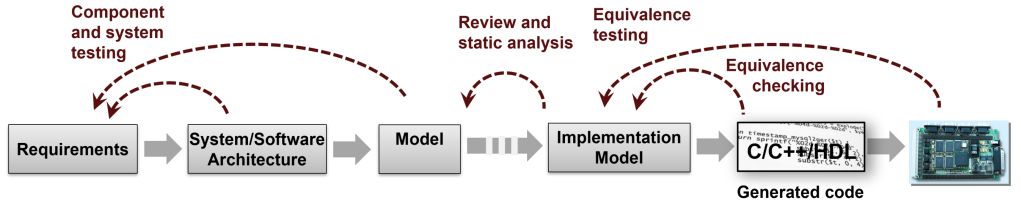
Is the code functionally equivalent to model?
Is all the code tested?

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

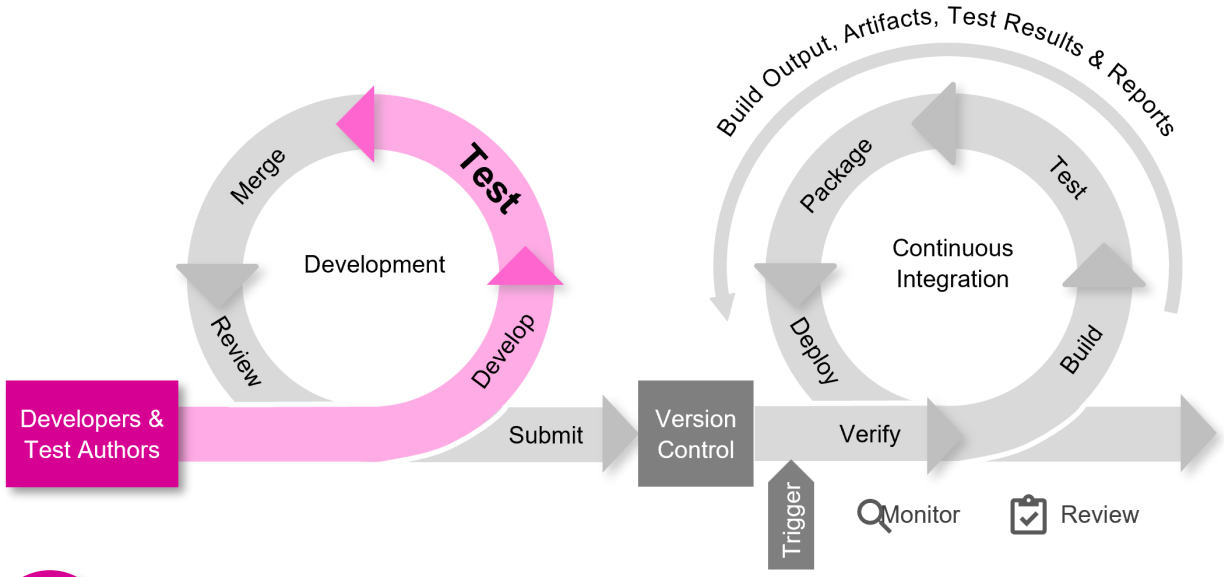


High Integrity Verification Workflow

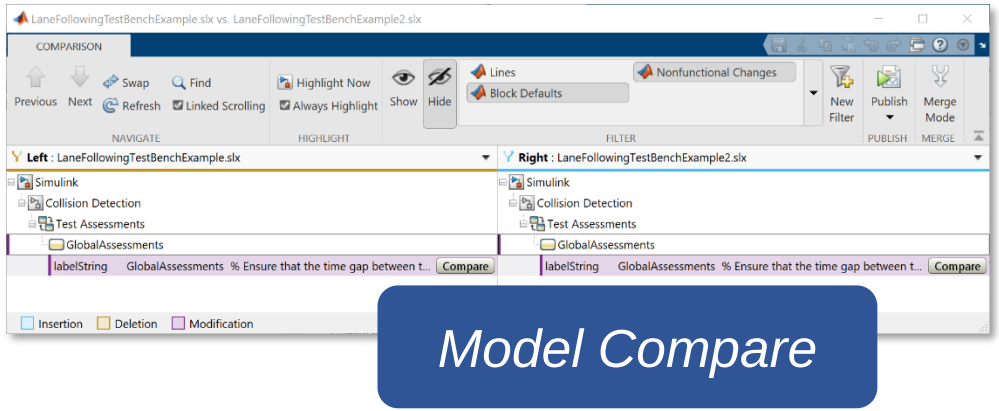


2 Test

Continuous Integration Workflow with Model-Based Design

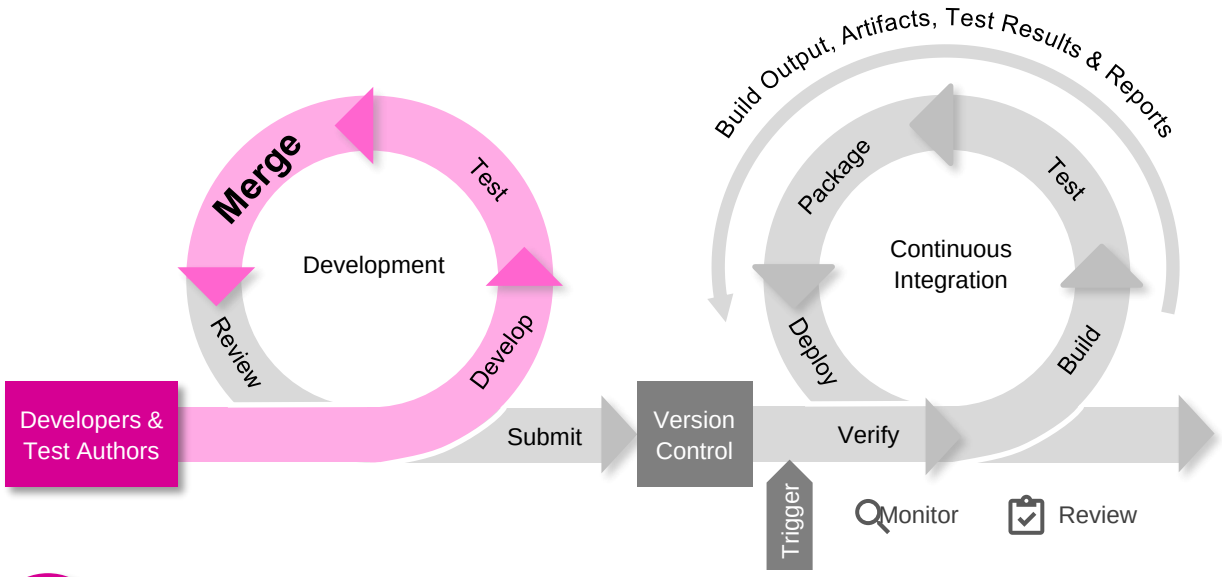


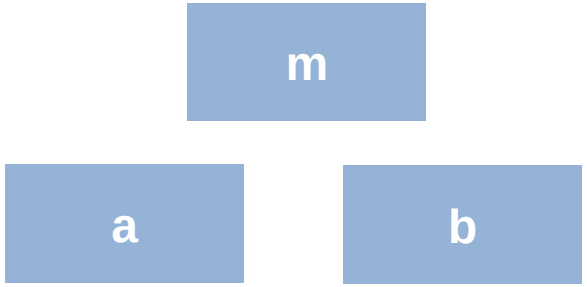
3 **Merge**



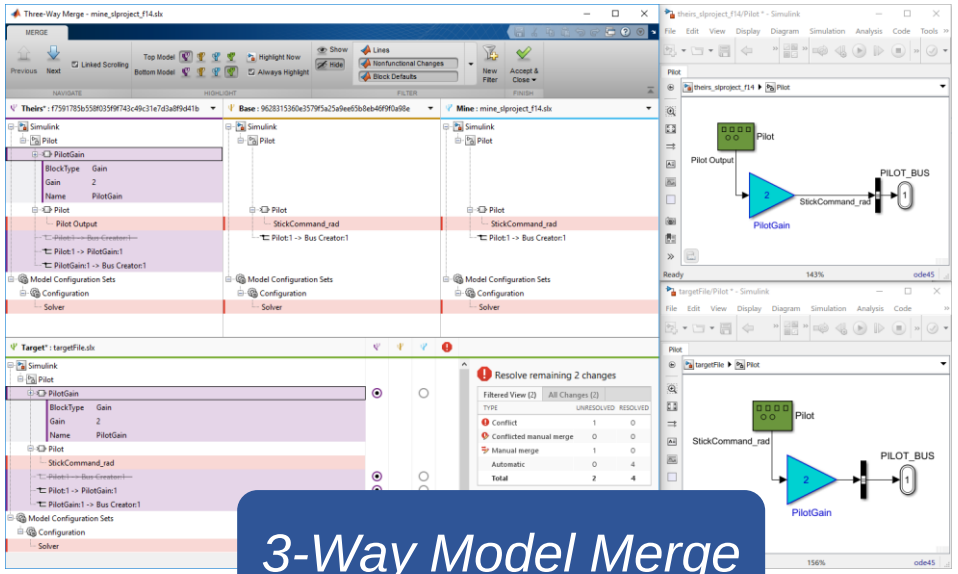
Model Compare

Continuous Integration Workflow with Model-Based Design



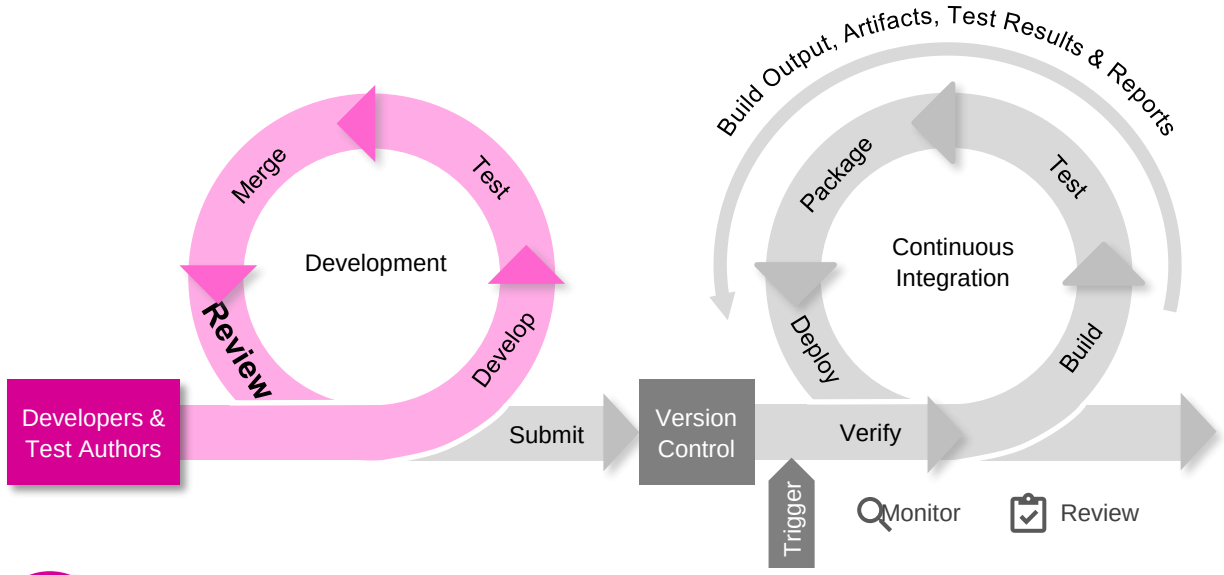


3 **Merge**

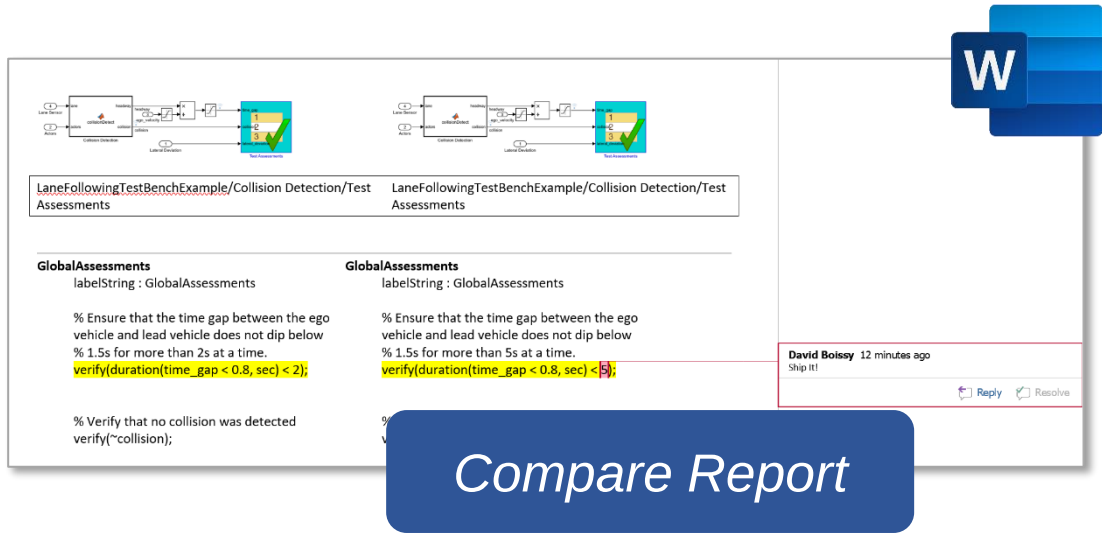


3-Way Model Merge

Continuous Integration Workflow with Model-Based Design

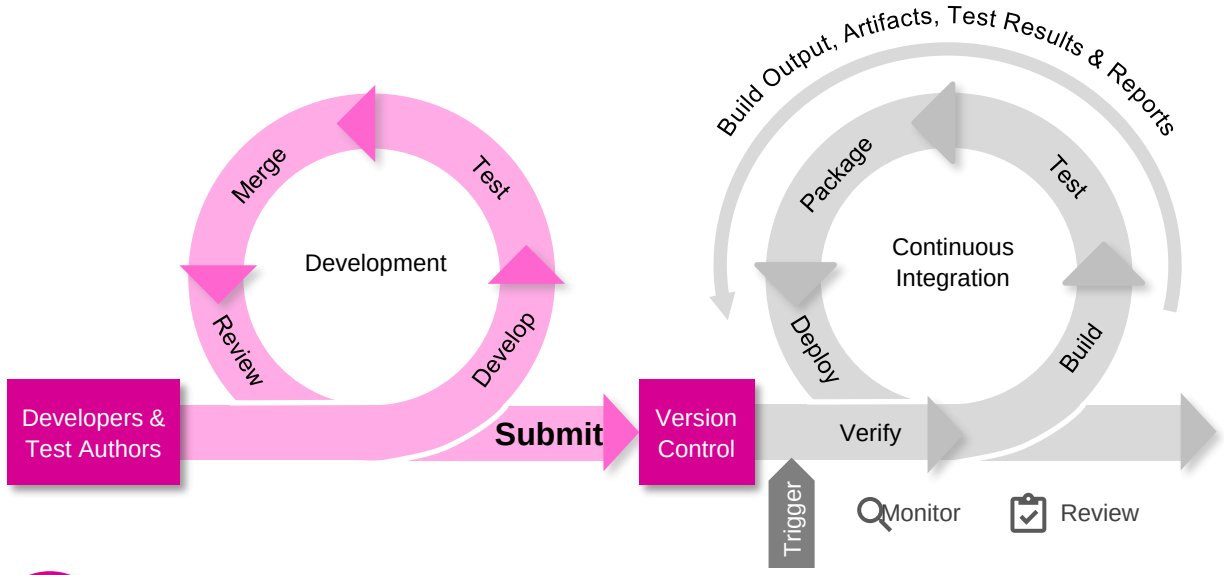


4 Review

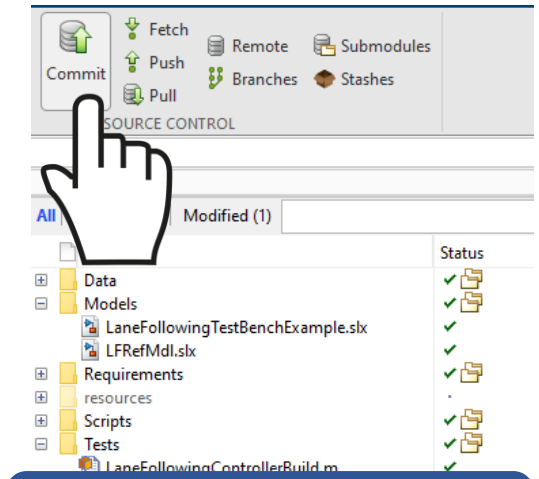


Compare Report

Continuous Integration Workflow with Model-Based Design

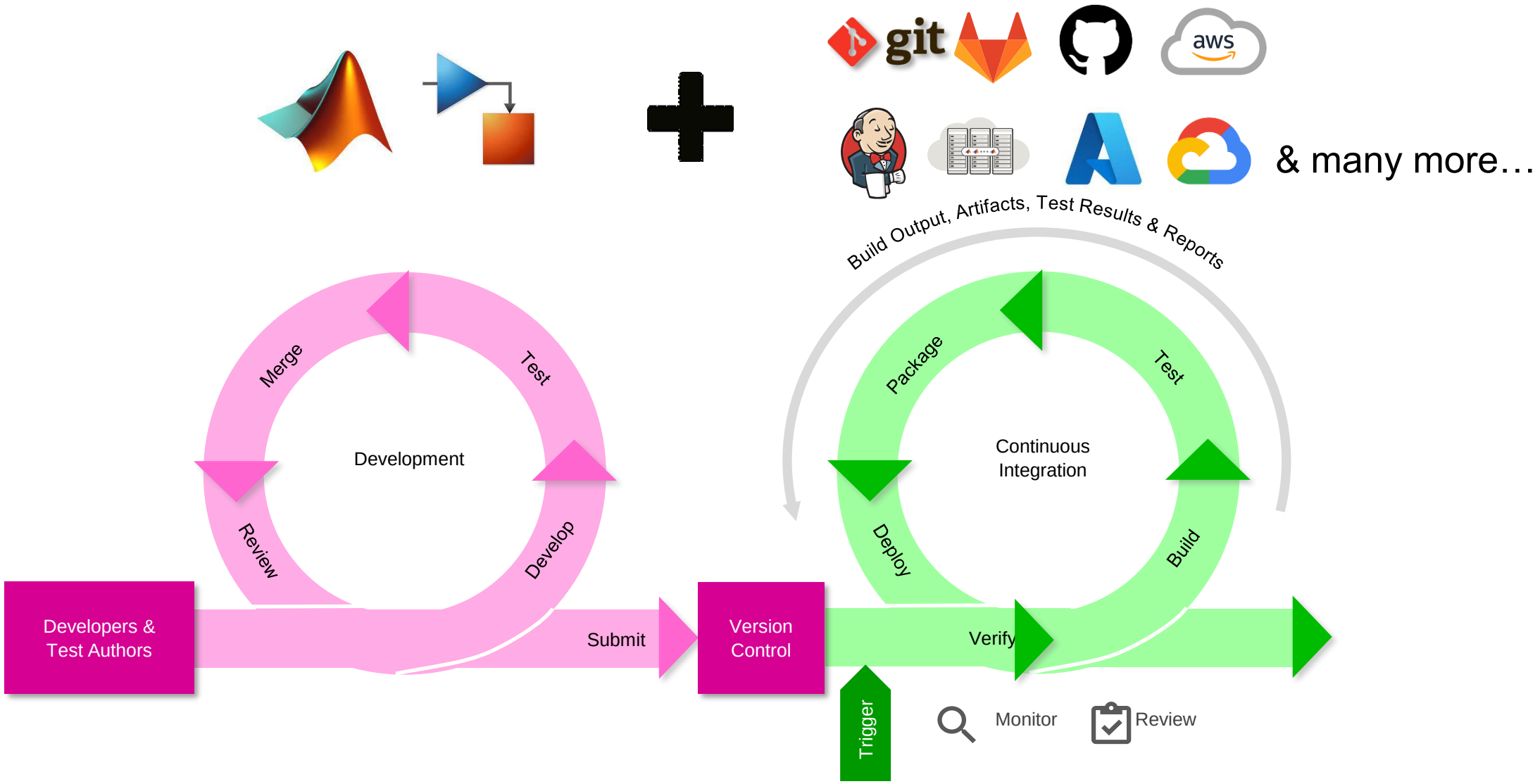


5 **Submit**



Projects

From Desktop to Cloud for Model Based Design using CI/CD



What are the benefits of CI?

What problems does it attempt to solve?

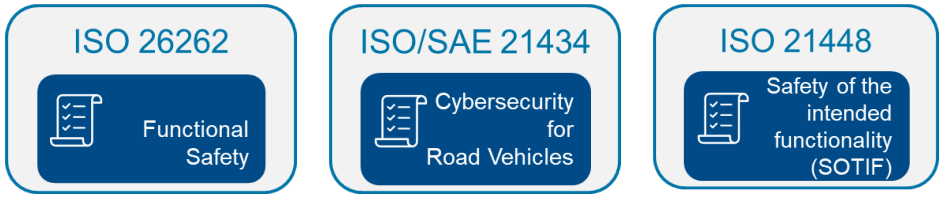
How does Simulink fit into the CI ecosystem?

How to best leverage CI with MBD?

Benefits of Continuous Integration

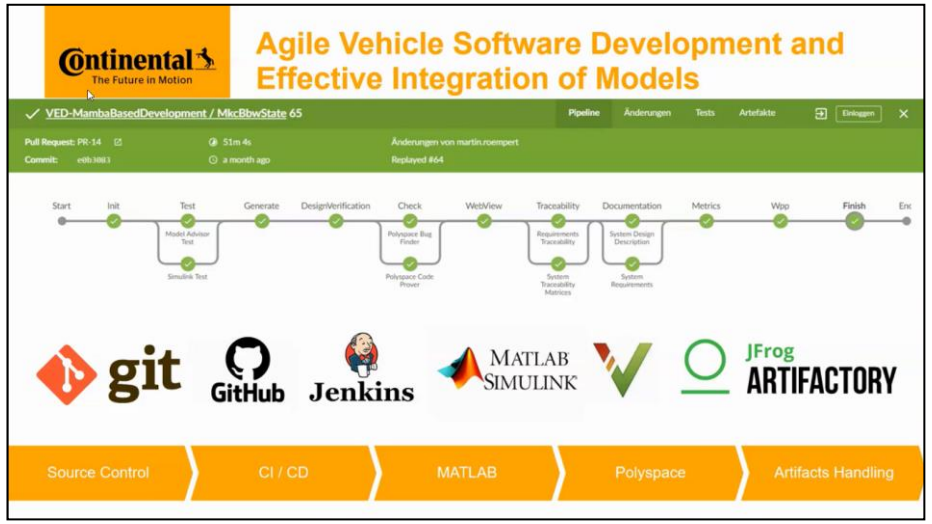
Model-Based Design enables high DevOps performance

- DevOps Goal: „**Reduce the time** between committing a change and placing it in production, **while ensuring high quality and compliance**„



	High Performers	Low Performers
Lead Time	**<1hour**	**>6months**
Change Failure Rate	**0-5%**	**15-30%**

Source: [state-of-devops-2021.pdf (google.com)]



Online Panel Discussion: [Agile Vehicle Software Development and Effective Integration of Models]

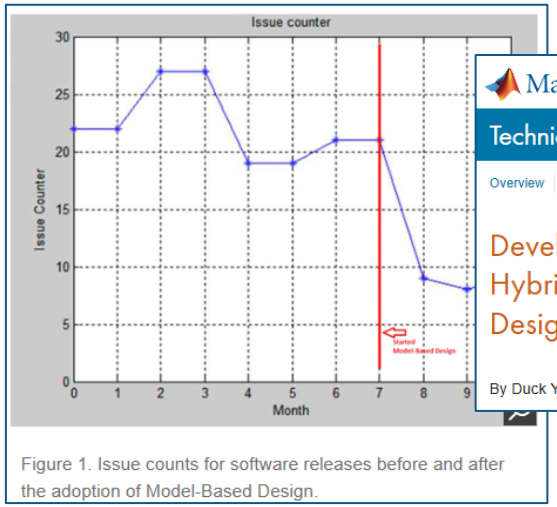
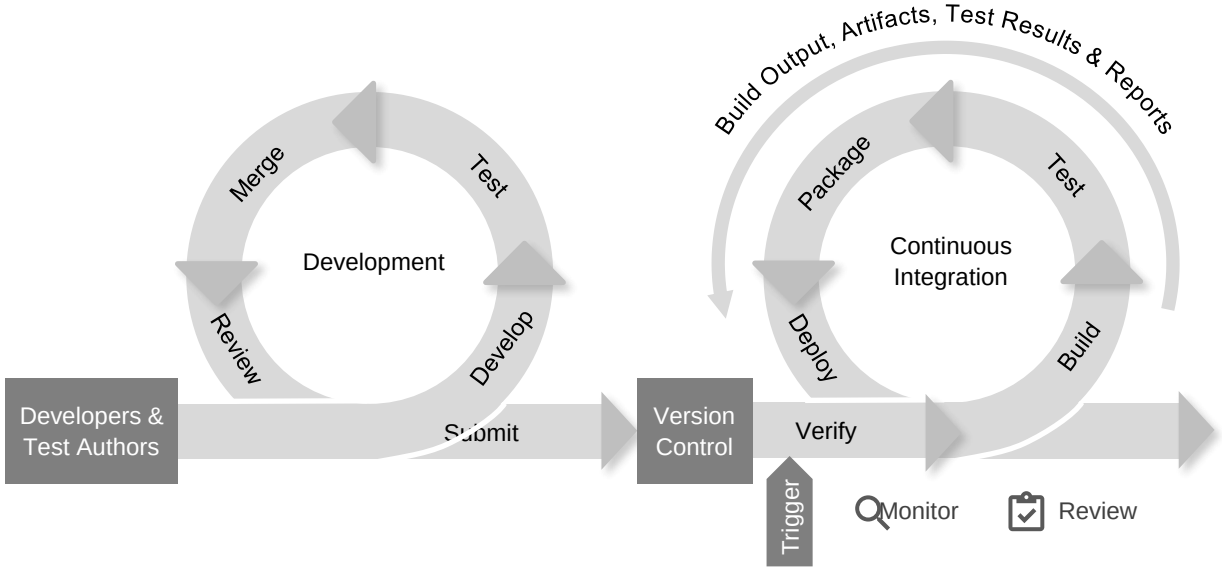


Figure 1. Issue counts for software releases before and after the adoption of Model-Based Design.

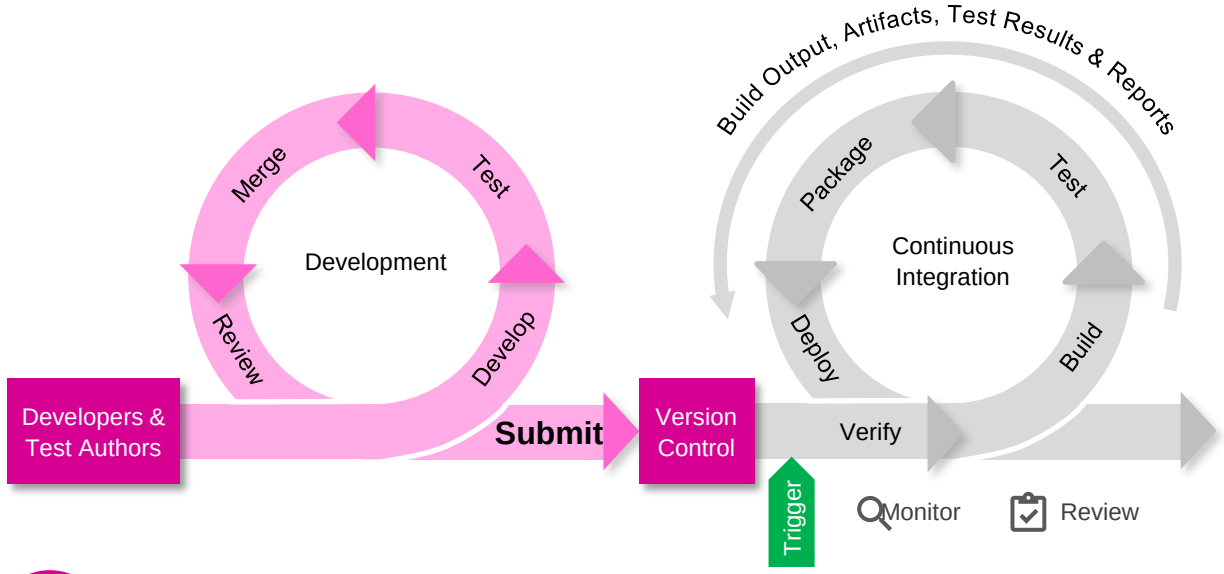
[Link to technical article]

CI workflow and tools are language- and domain-neutral



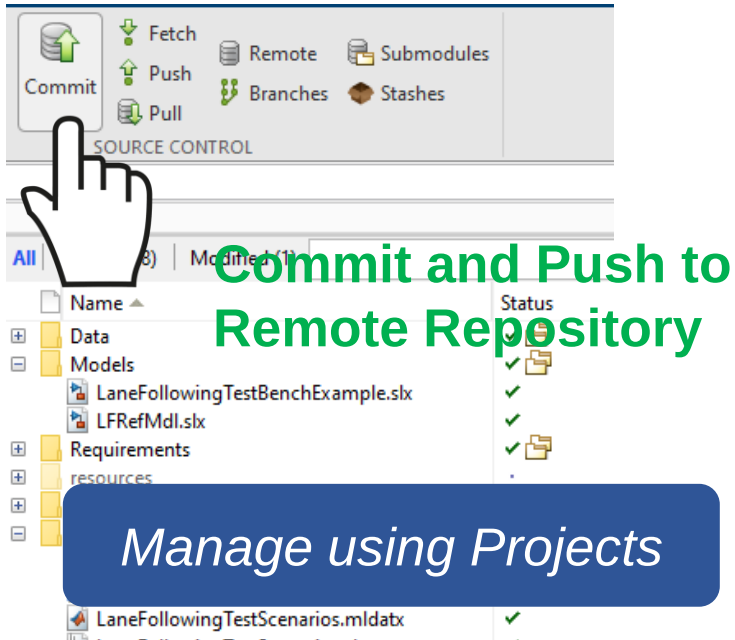
Each of these can “speak” MATLAB and Model-Based Design

Continuous Integration Workflow with Model-Based Design



1 **Development**

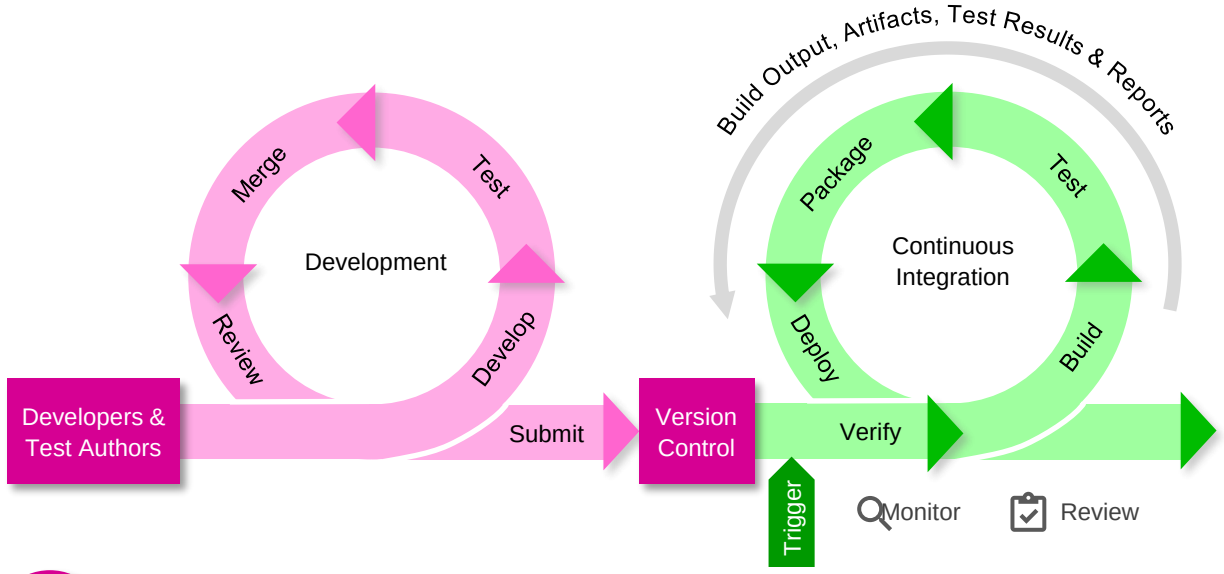
2 **Continuous Integration**



Commit and Push to Remote Repository

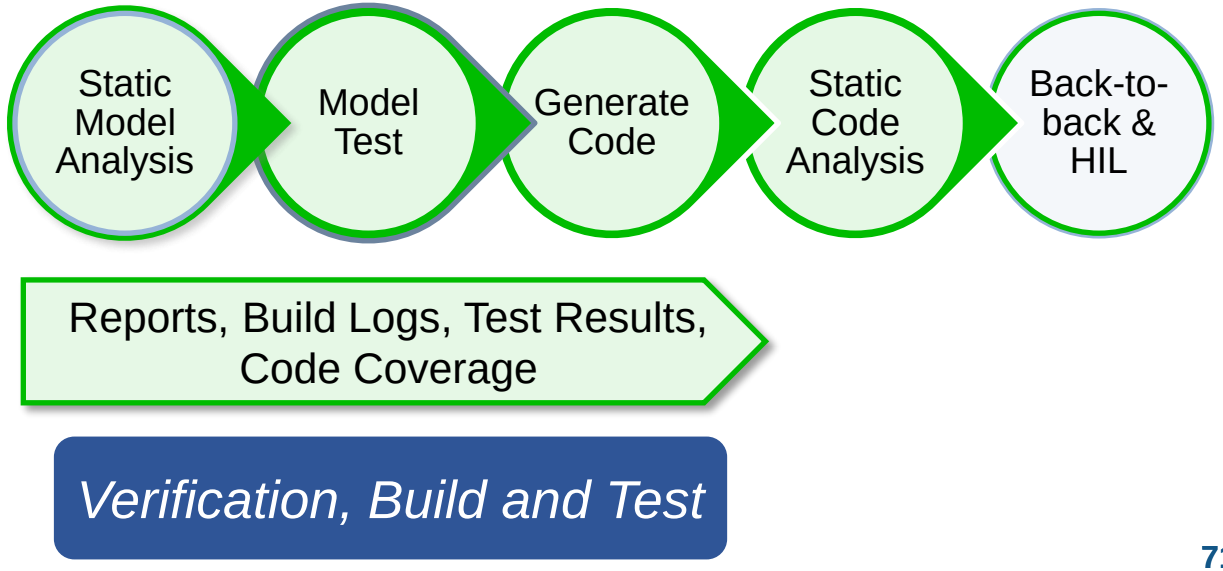
Manage using Projects

Continuous Integration Workflow with Model-Based Design

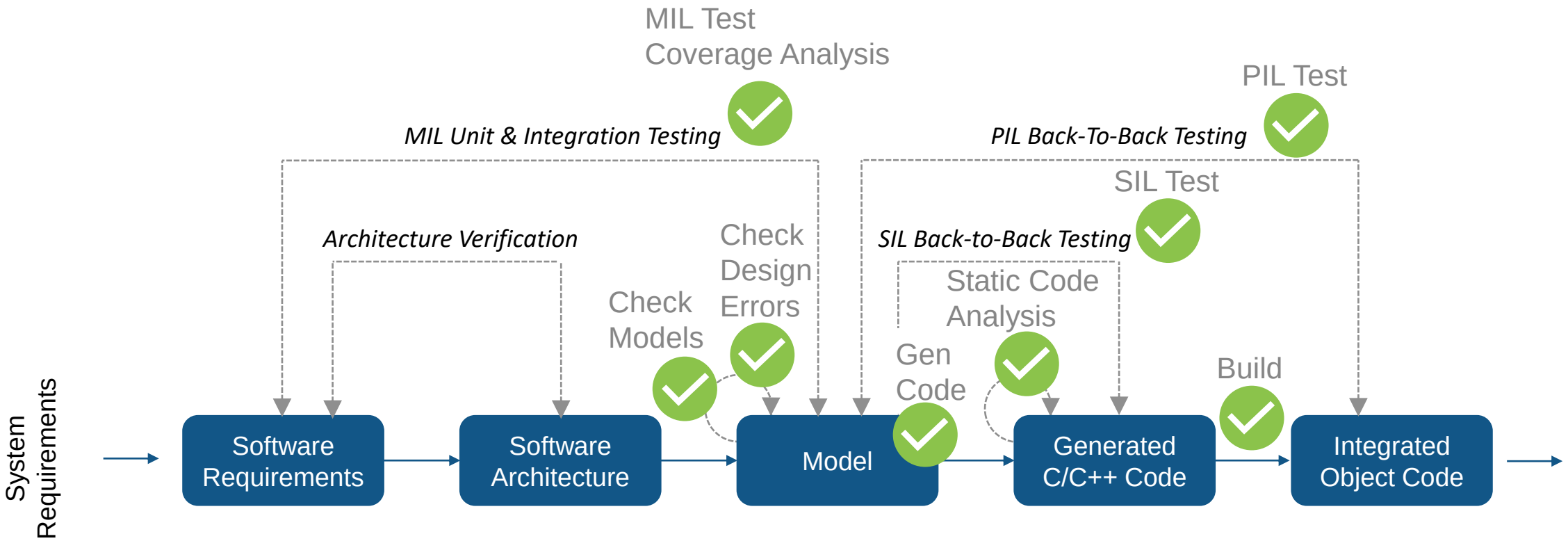


1 **Development**

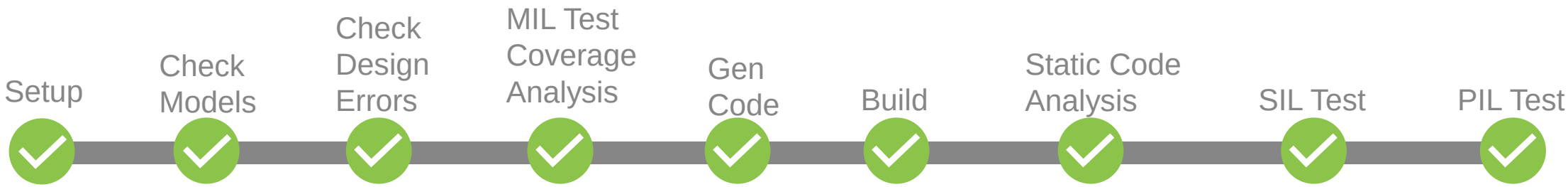
2 **Continuous Integration**



Model-Based Design Reference Workflow

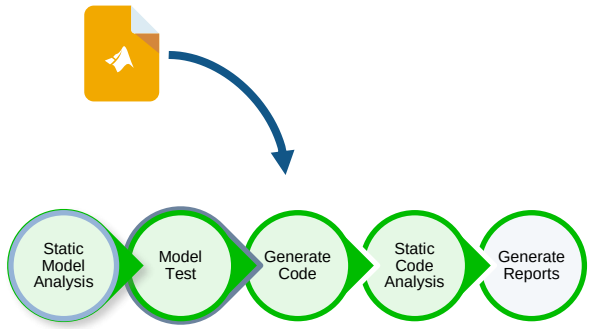


Model-Based Design Reference Workflow

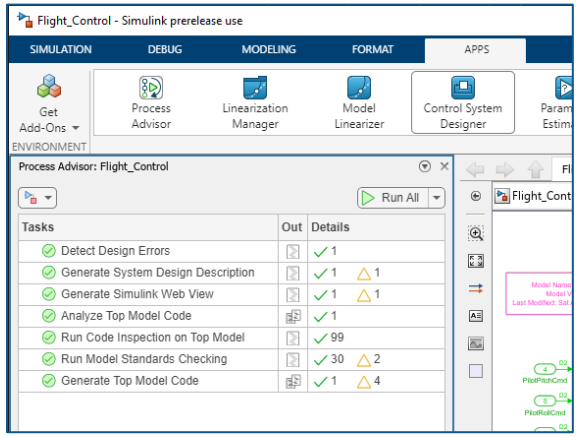


- Define Process and Automate
  - Identify Tasks
  - Define Sequence
  - Define Outputs
  - Script the Tools

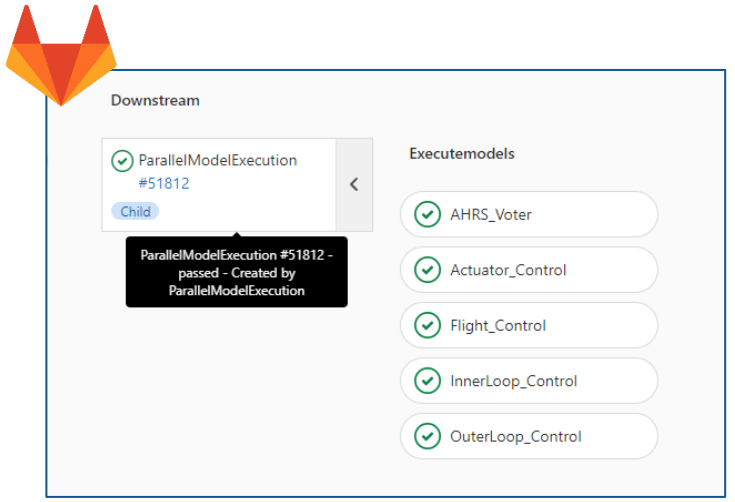
CI/CD Automation for Simulink Check Support Package



1) <u>Simple Setup</u>
- ✓ Prebuilt Model-Based Design pipeline
- ✓ Built-in Model-Based Design tool support
- ✓ Tailorable



2) <u>Desktop Integration with Process Advisor app</u>
- ✓ Local prequalification
- ✓ Local Debugging



3) <u>3rd Party CI Integration</u>
- ✓ Jenkins/Gitlab YAML
- ✓ Optimized Model-Based Design Builds
- ✓ CI Results Integration

Prebuilt & Tailorable MBD Pipeline

Built-in Library of Tasks

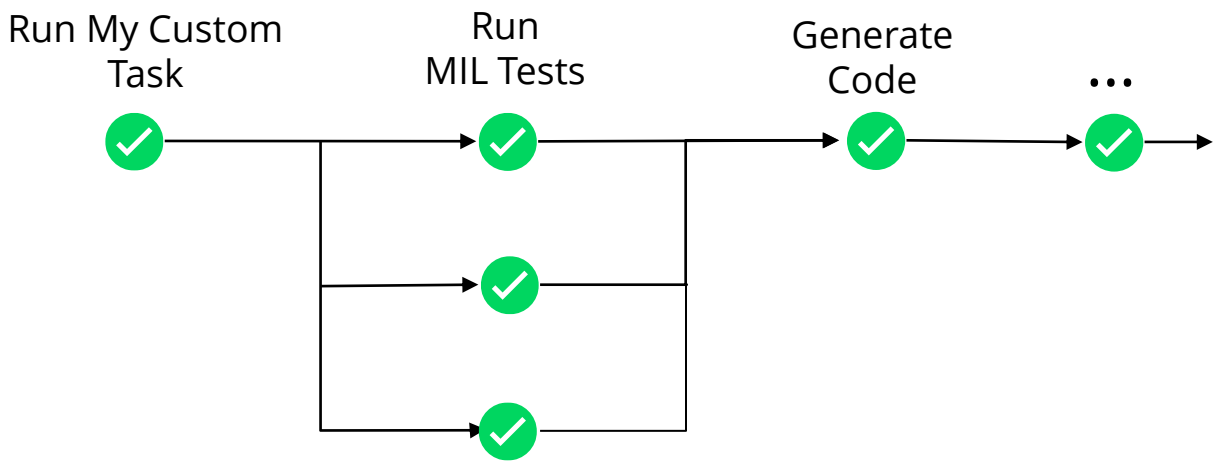
- Static Analysis
- Code Generation
- Testing

Zero Upfront Code

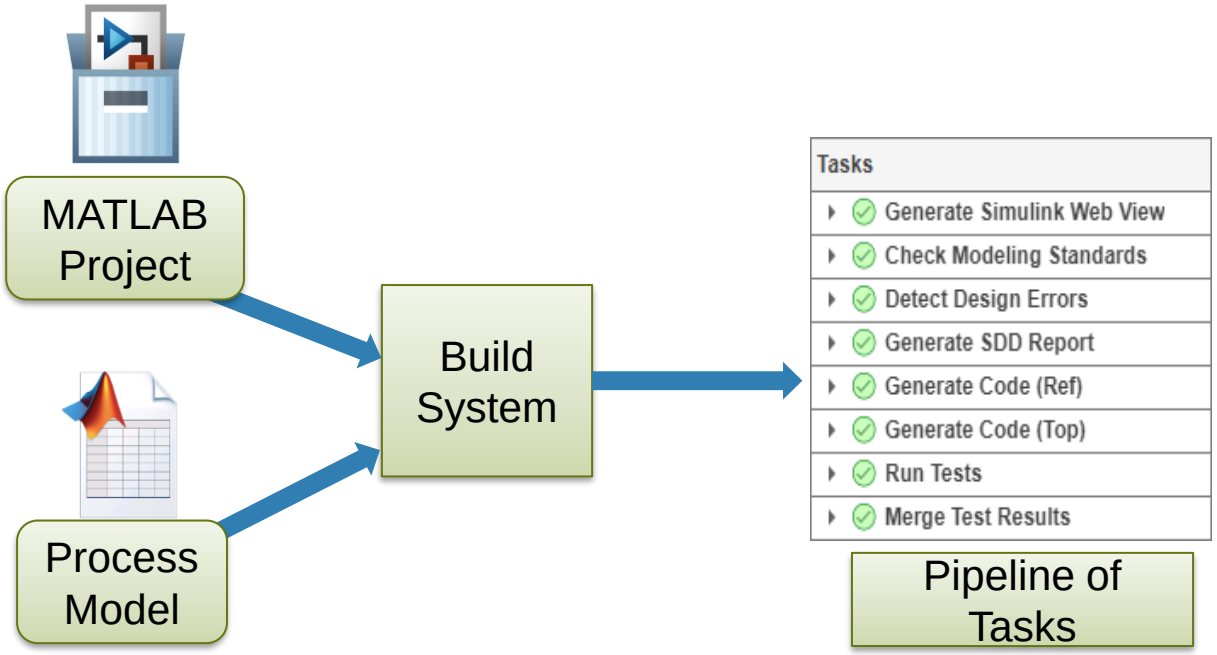
TASKS	TOOLS
Check Model Standards Compliance	Simulink Check
Run Tests	Simulink Test
Generate Source Code	Embedded Coder
Check Code Standards Compliance	Polyspace Bug Finder
Generate Software Design Description	Simulink Report Generator
Design Error Detection	Simulink Design Verifier
Verify Model Update & Simulation	Simulink
Check Model Metrics	Model Advisor

Fully Tailorable

- Modify existing steps
- Remove steps
- Add custom steps



MBD Pipeline Generation and Task Execution System



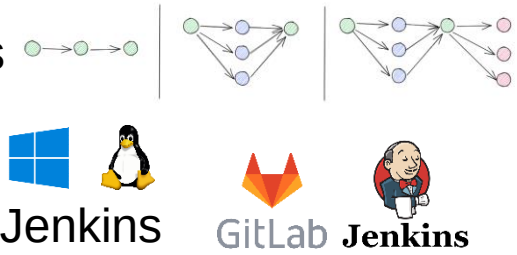
Capabilities

Execute in different workflows

- Interactive in Desktop (Process Advisor)
- Automated in CI
- Import results from CI into Desktop

Generate CI Pipeline

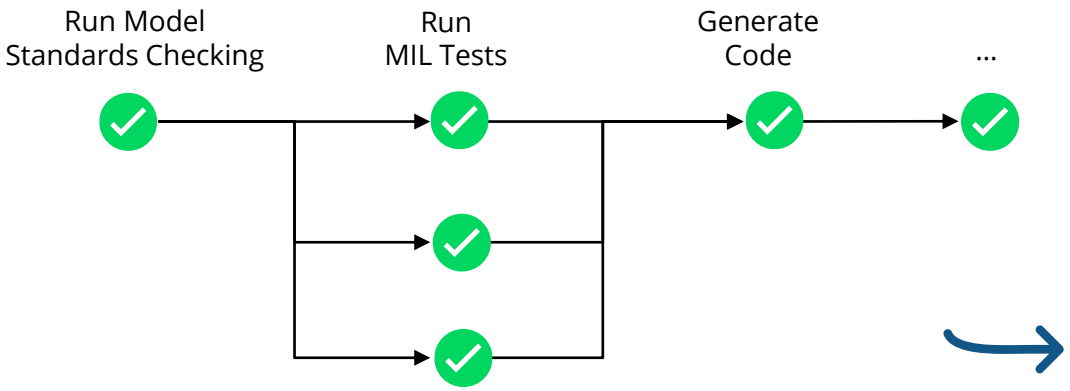
- Multiple Architectures
- Multiple Platforms
  - OS agnostic
  - Current: GitLab, Jenkins
  - Future: GitHub, Azure Pipelines



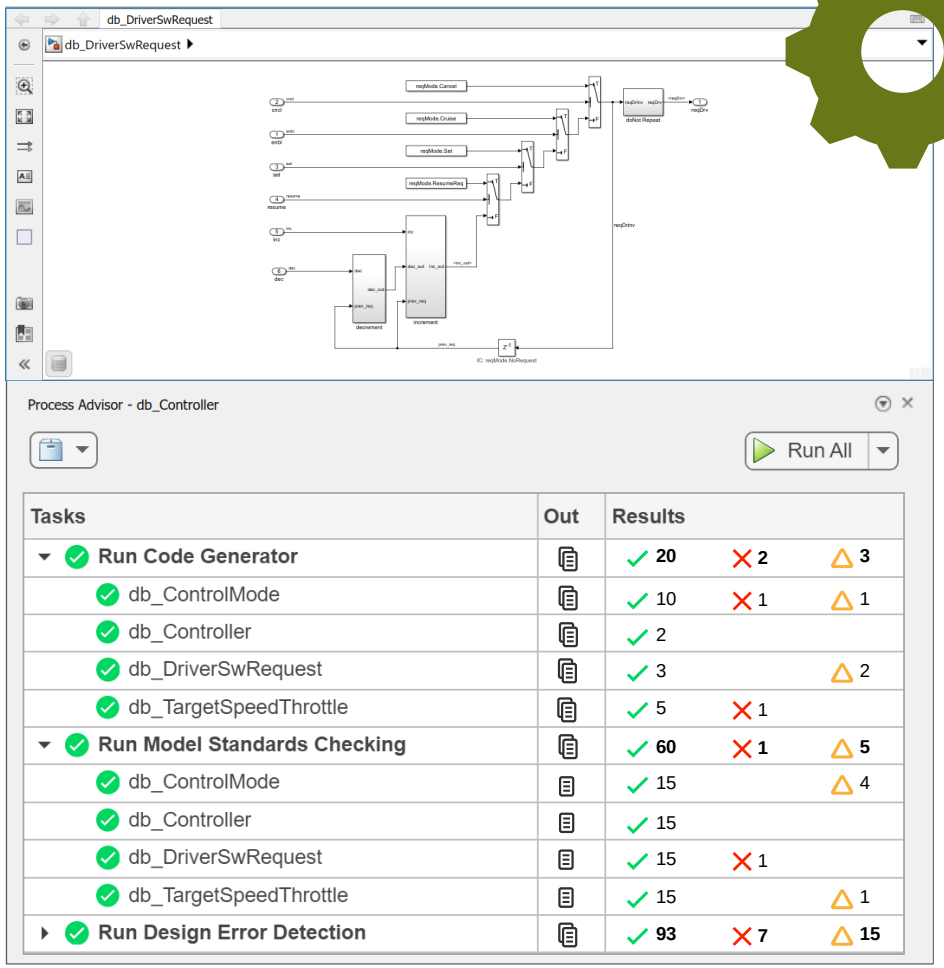
Smart Orchestration

- Incremental execution
- Repeatable results

Pre-qualification with Process Advisor



Process Model



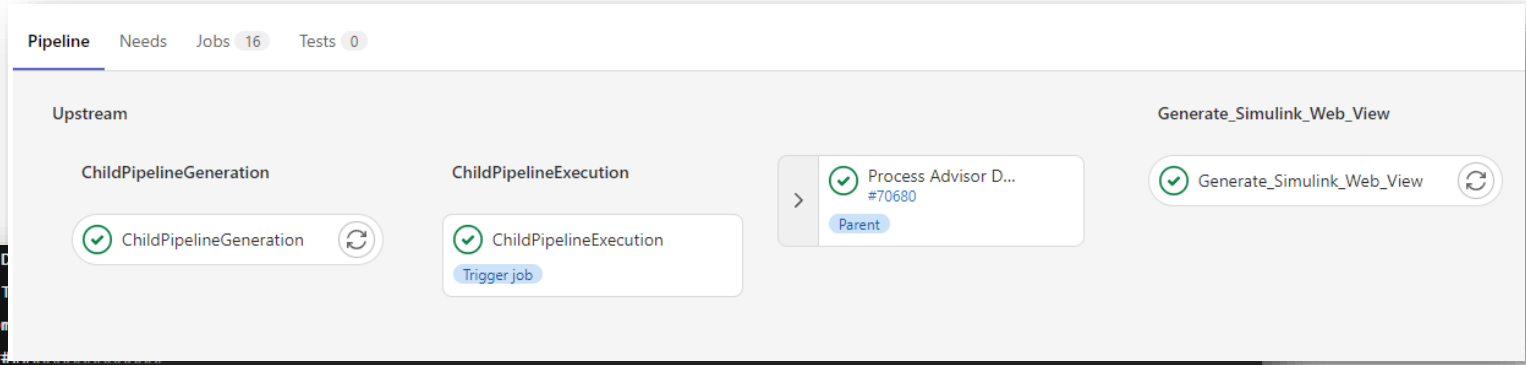
MBD
Build Tool

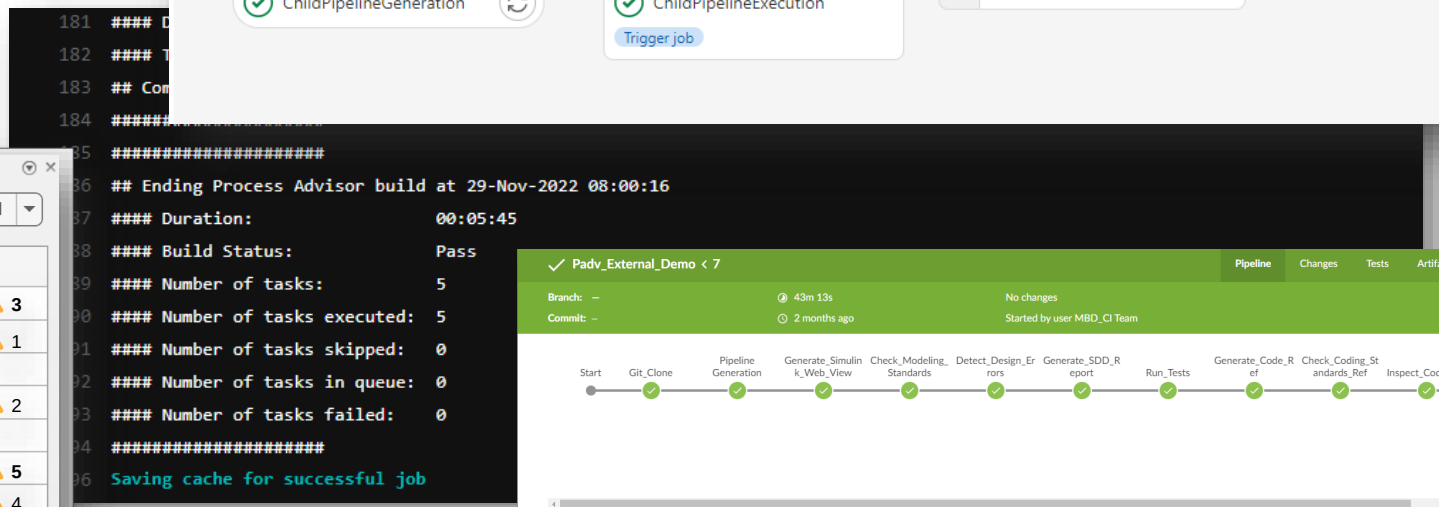
Process Advisor

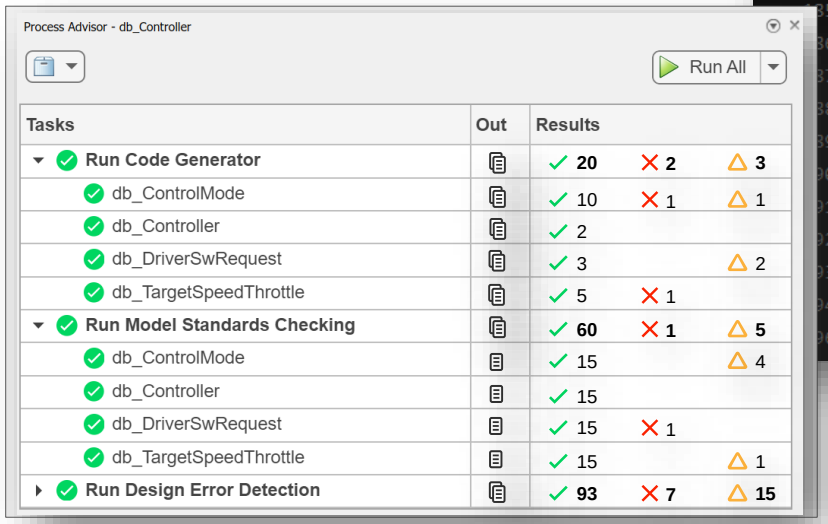
Local Desktop Workflow

Integration and Run with common CI Systems

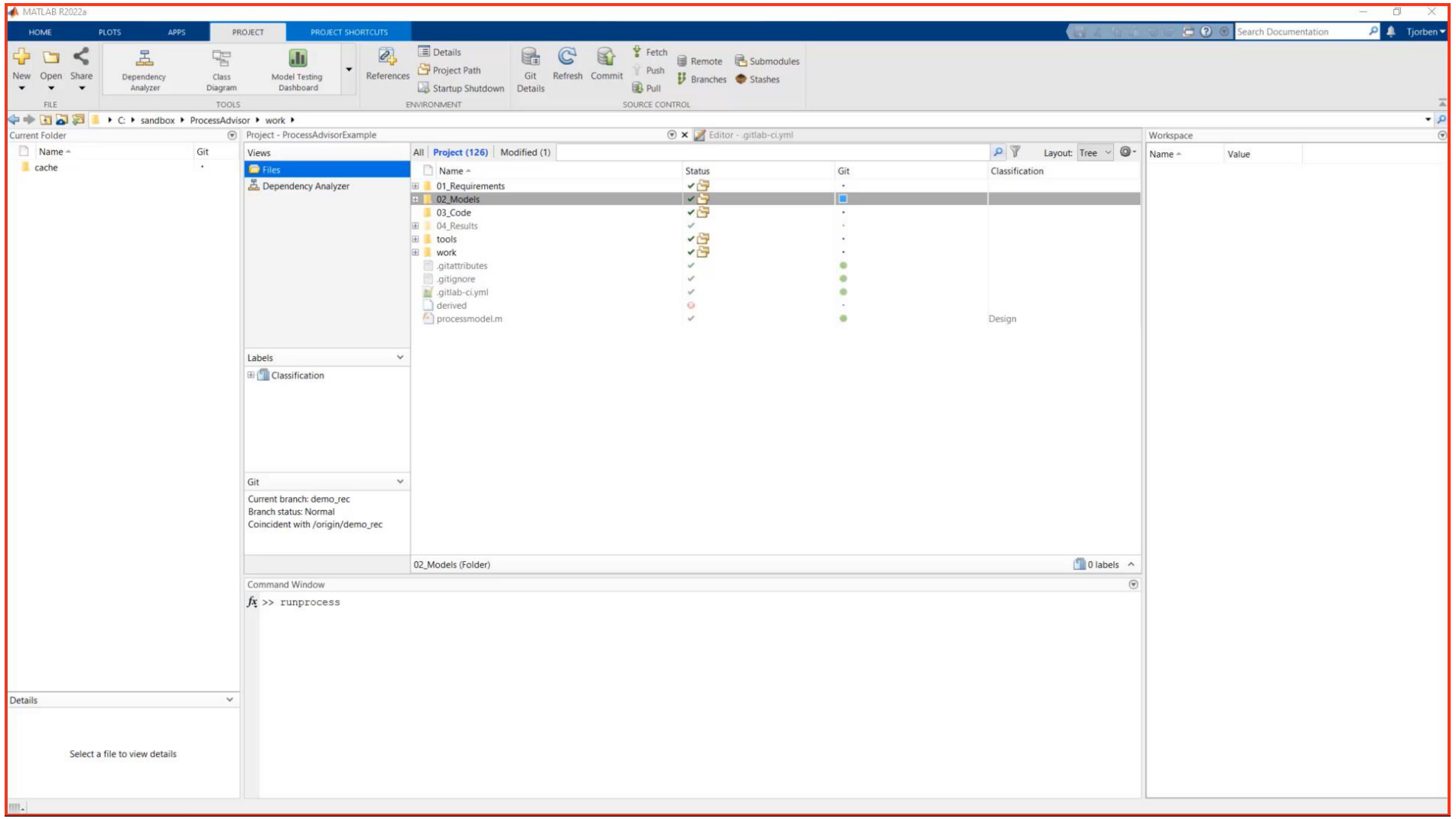
- Automated Pipeline Generation
- Execute Pipeline in CI Systems like Jenkins
- Publish Results
- Debug on Desktop



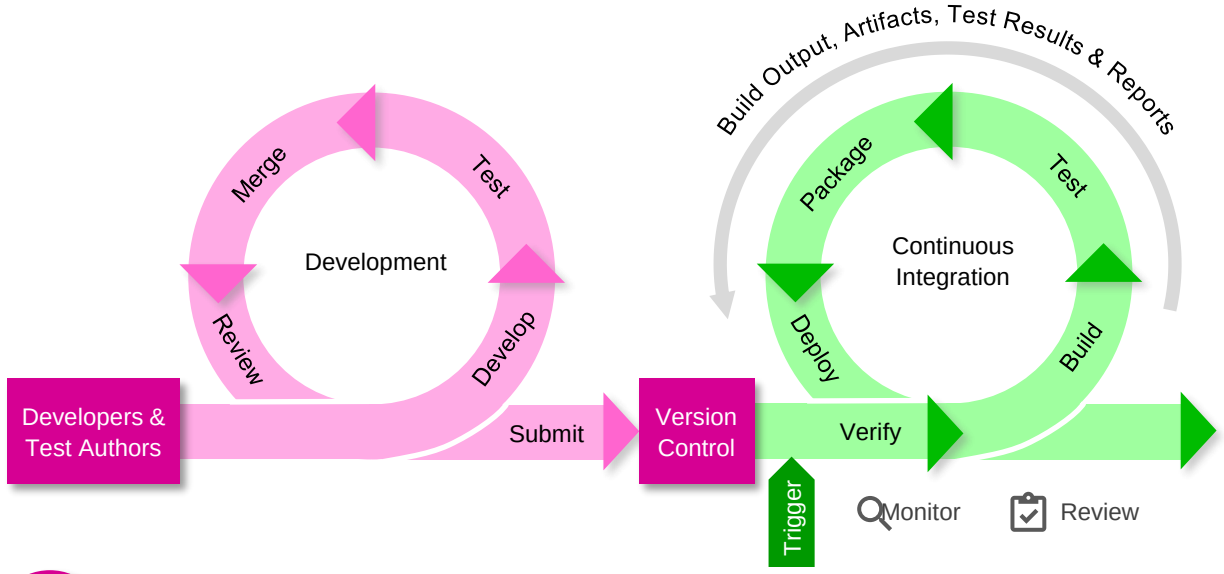




Continuous Integration Workflow with Model-Based Design – Invoke pipeline

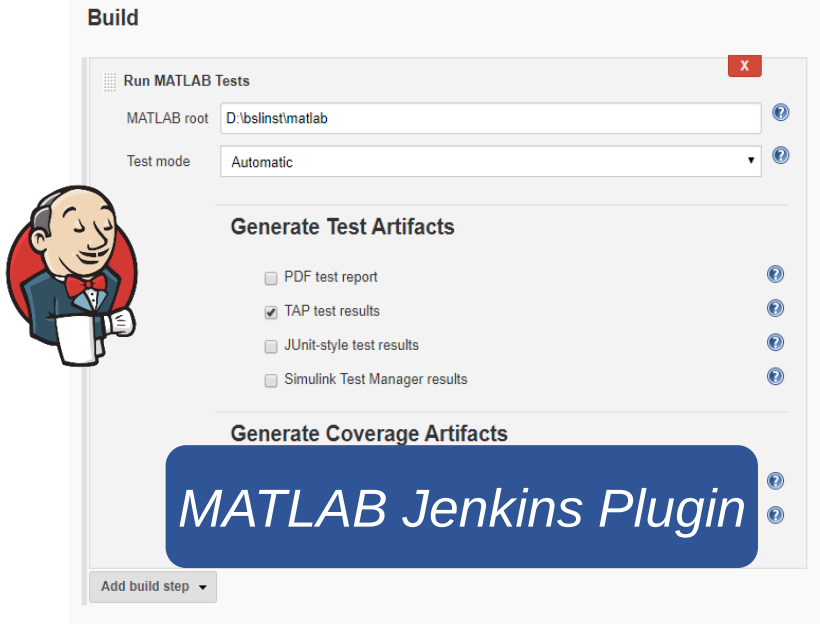


Continuous Integration Workflow with Model-Based Design

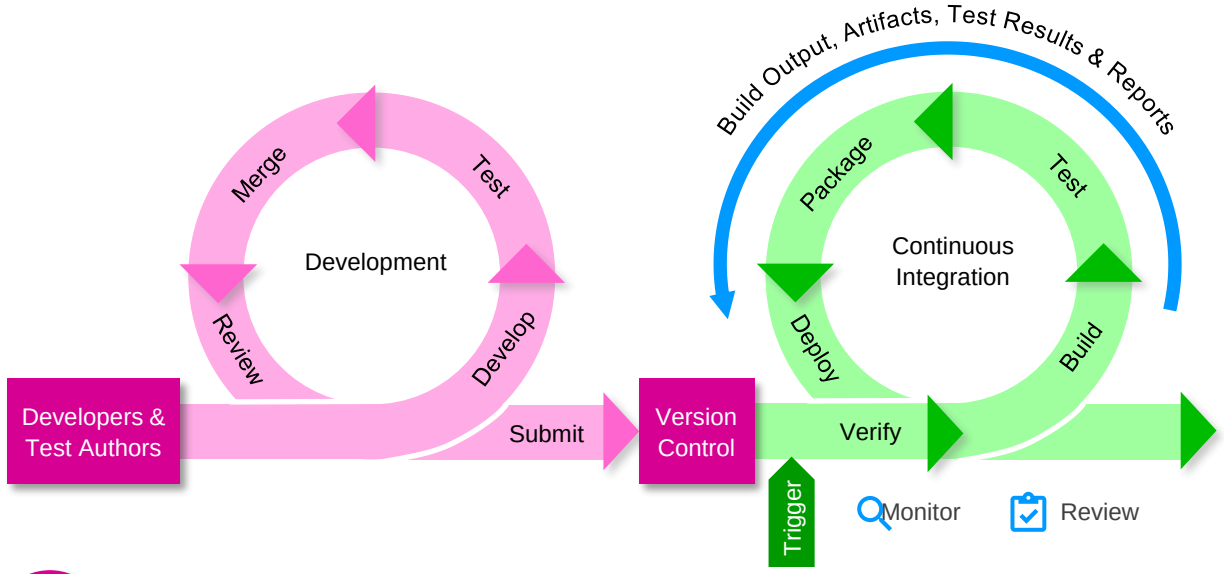


1 **Development**

2 **Continuous Integration**



Continuous Integration Workflow with Model-Based Design



1 **Development**

2 **Continuous Integration**

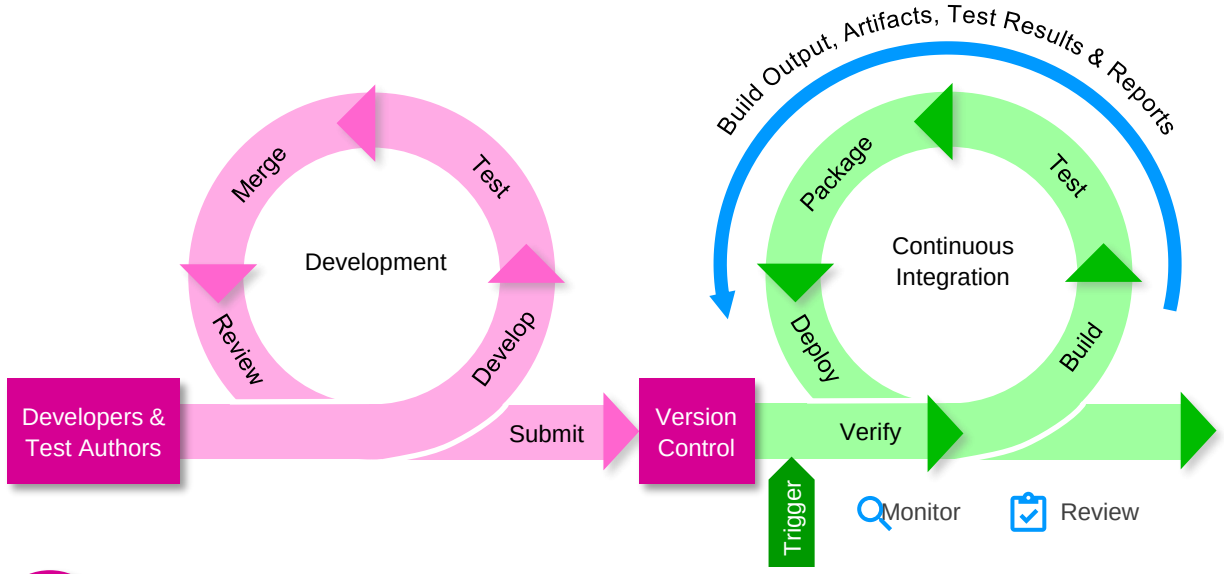
3 **Results Monitor and Review**

```
Failure Summary:

     Name                                                                                Failed
================================================================================================
     LaneFollowingTestScenarios > Scenarios/LFACC_Curve_CutInOut_TooClose    X
ERROR: MATLAB error Exit Status: 0x00000001
Build step 'Run MATLAB Tests' changed build result to FAILURE
Finished: FAILURE
```

Logs

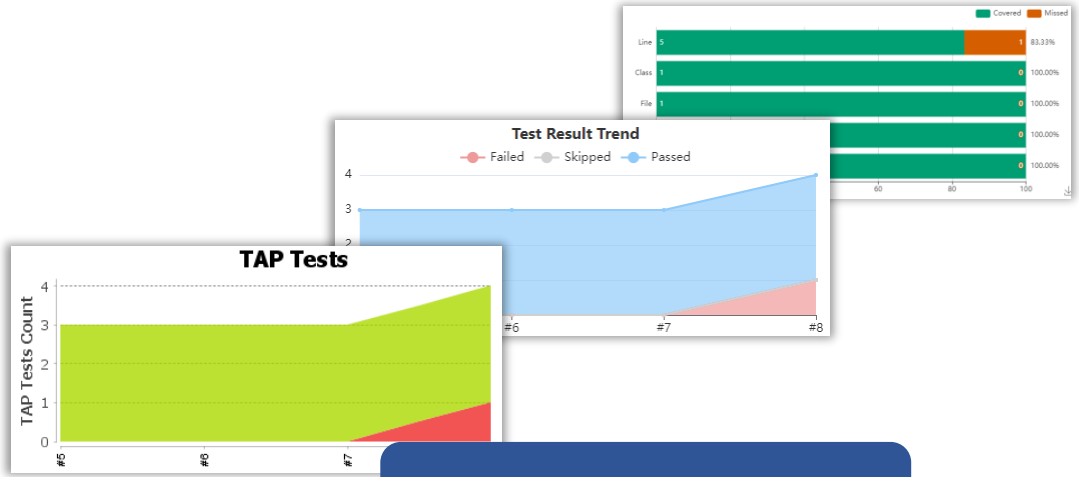
Continuous Integration Workflow with Model-Based Design



1 **Development**

2 **Continuous Integration**

3 **Results Monitor and Review**



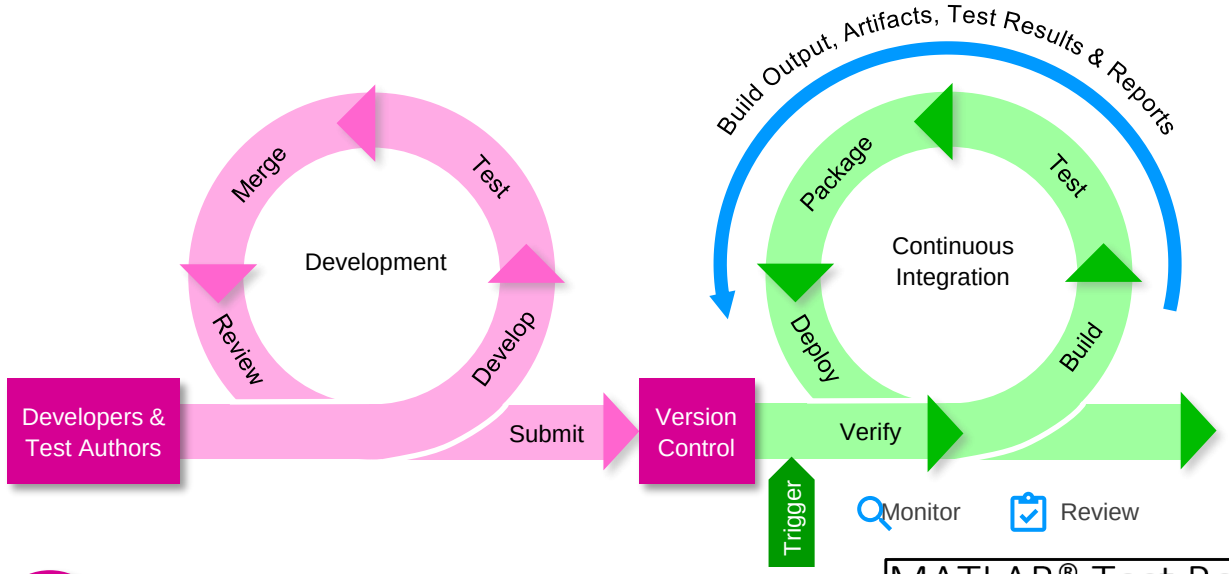
junit, TAP, Cobertura

Continuous Integration Workflow with Model-Based Design

ISO 26262
Functional Safety

ISO/SAE 21434
Cybersecurity for Road Vehicles

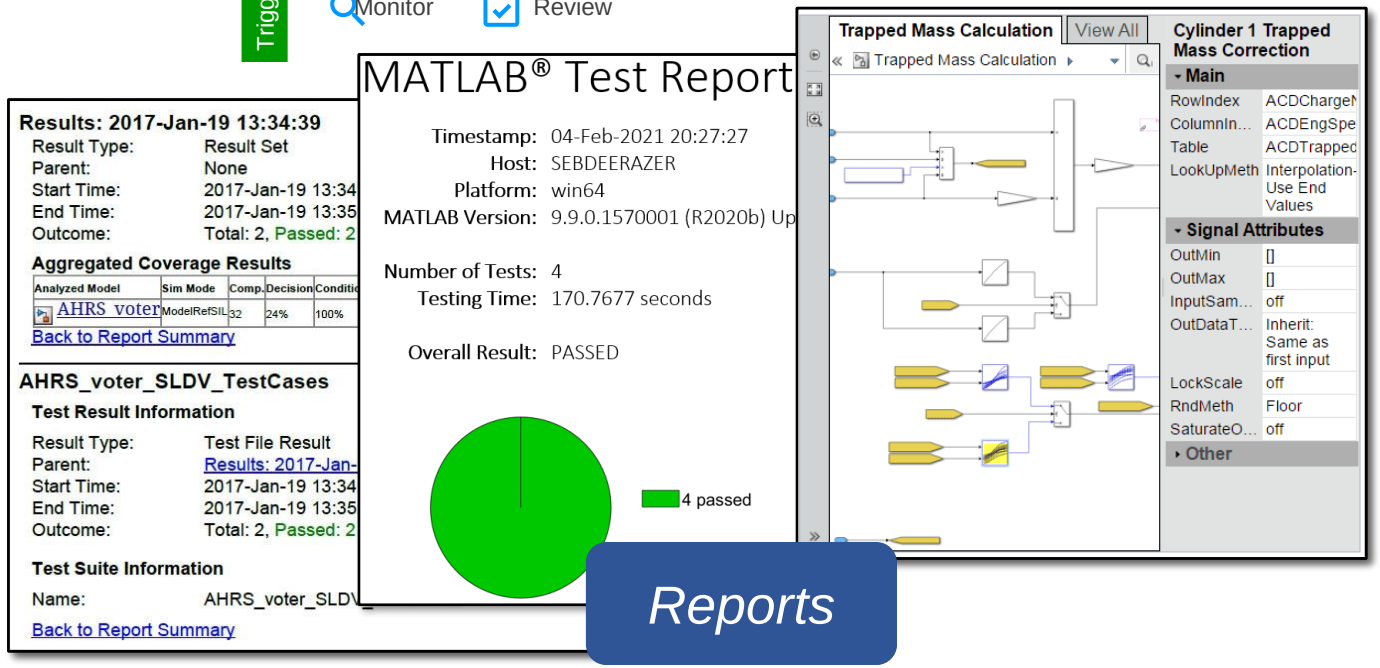
ISO 21448
Safety of the intended functionality (SOTIF)



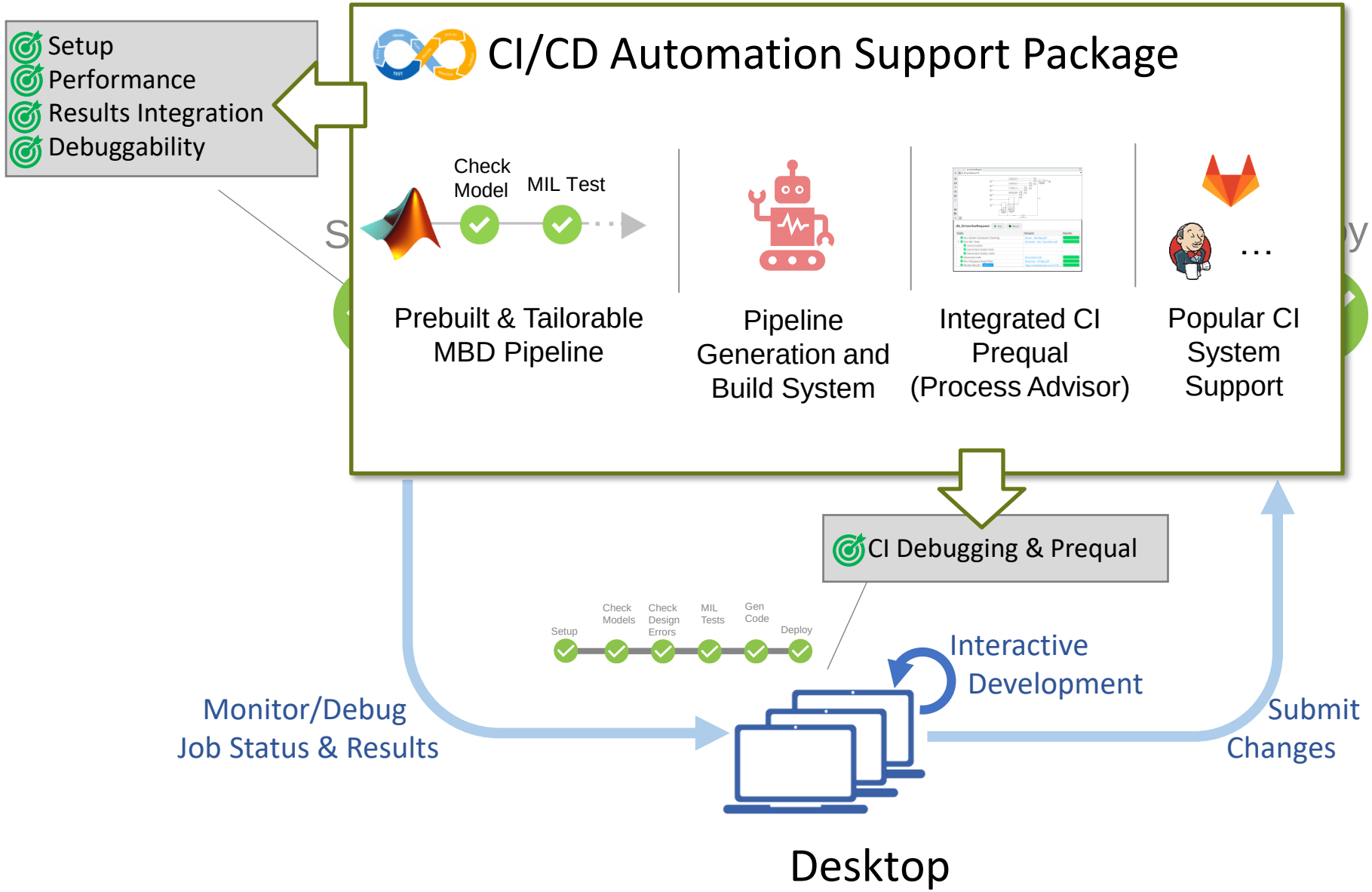
1 **Development**

2 **Continuous Integration**

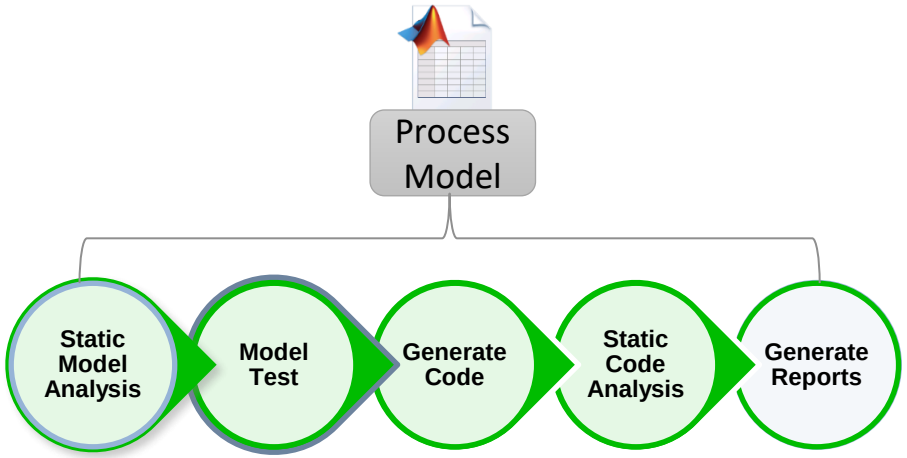
3 **Results Monitor and Review**



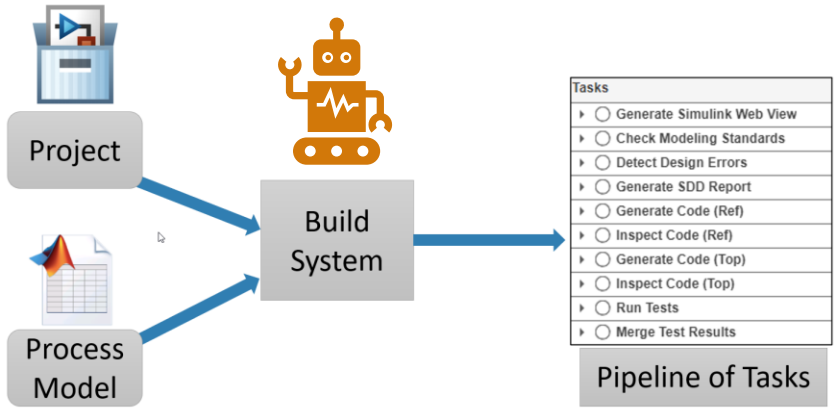
Simplifying Adoption and Optimizing CI/CD for Model-Based Design



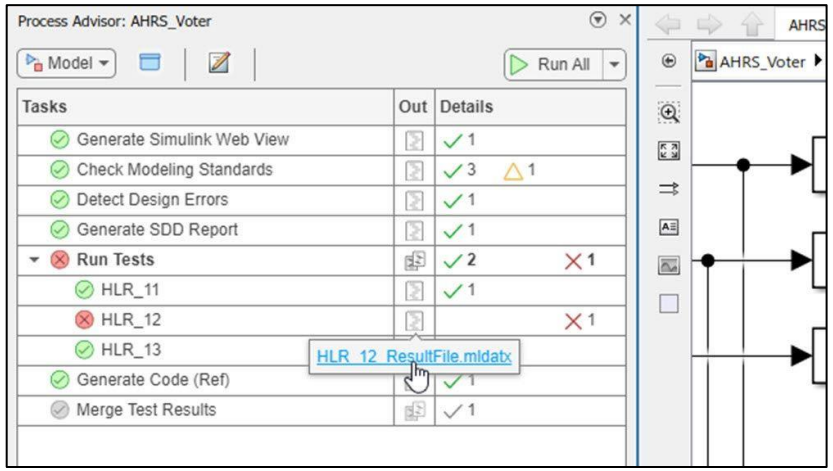
CI/CD Automation for Simulink Check Support Package



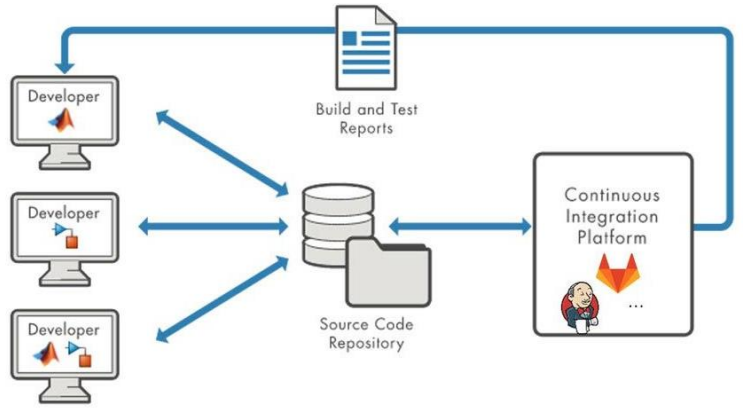
Prebuilt & Tailorable Model-Based Design Pipeline



Build system to generate and optimally execute the process in your CI system



Prequalification with Process Advisor

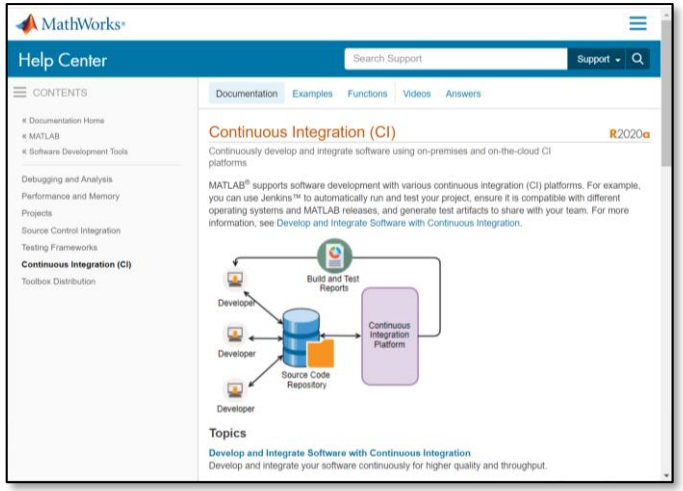


Examples to run process on common CI Systems

Learn more: [Continuous Integration for Model-Based Design]

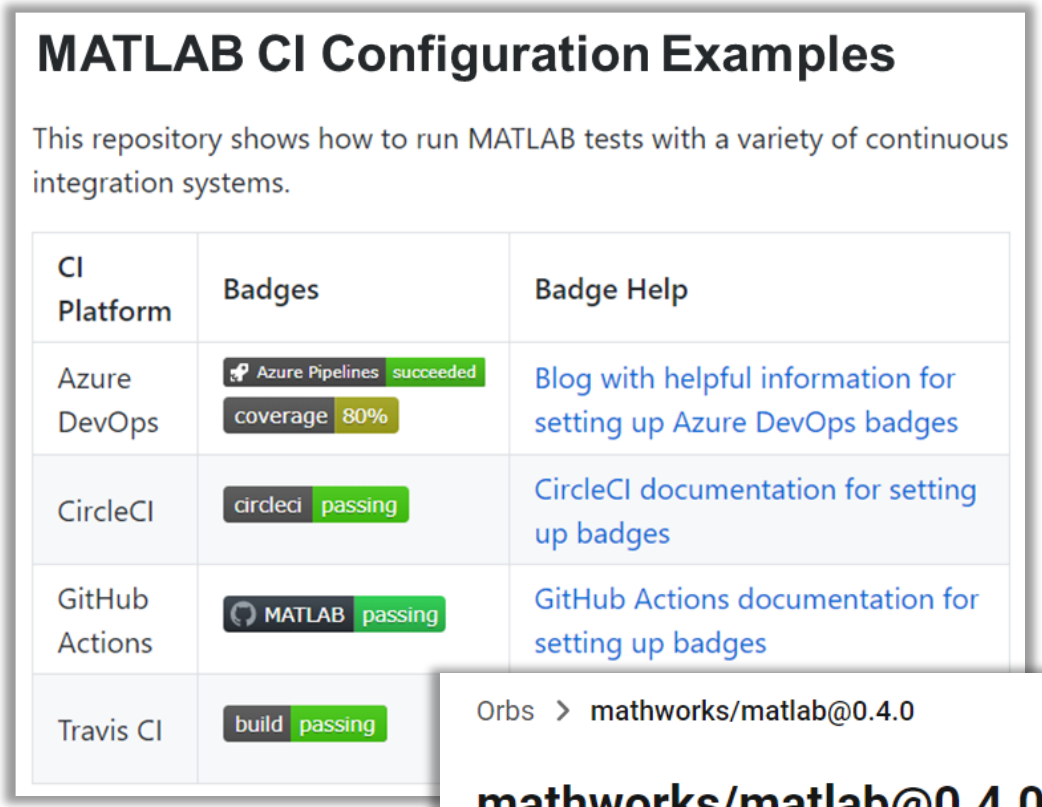
Other CI resources:

- Continuous Integration Solution Page
- Videos:
  - Continuous Integration with MATLAB and Simulink
  - Automotive DevOps for Model-Based Design with AWS
- Technical Articles:
  - Continuous Integration for Verification of Simulink Models
  - Continuous Integration for Verification of Simulink Models Using GitLab
  - Agile Model-Based Design: Accelerating Simulink Simulations in CI Workflows
- Documentation:
  - CI/CD Automation for Model-Based Design Support Package
  - Continuous Integration Documentation Hub
  - Tests for Continuous Integration
- Developer Zone: Continuous Integration



Getting Started: CI plugins and code examples

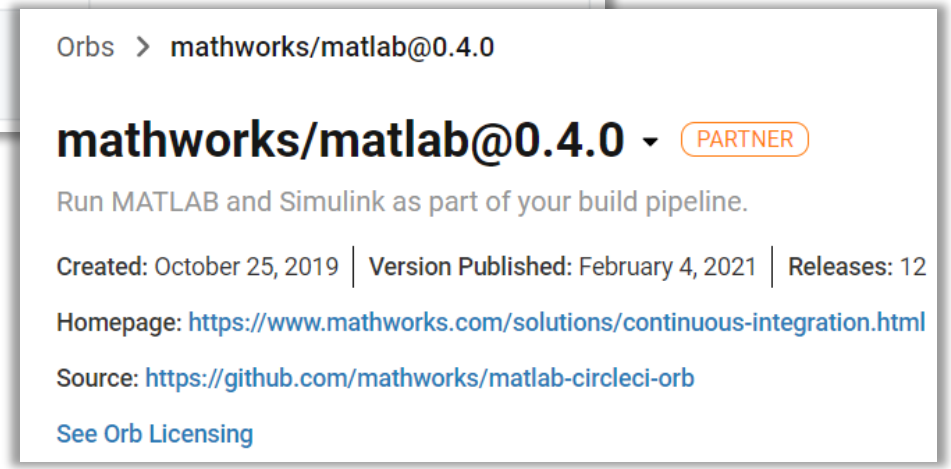
- Code examples
  - CI configuration examples
  - CI with Simulink
  - Code coverage using Codecov
- CI plugins
  - Azure DevOps
  - CircleCI
  - GitHub Actions
  - Jenkins
  - Travis CI
- Reference architectures (AWS, GCP, …)
  - https://github.com/mathworks-ref-arch



MATLAB CI Configuration Examples

This repository shows how to run MATLAB tests with a variety of continuous integration systems.

CI Platform	Badges	Badge Help
Azure DevOps	Azure Pipelines succeeded; coverage 80%	Blog with helpful information for setting up Azure DevOps badges
CircleCI	circleci passing	CircleCI documentation for setting up badges
GitHub Actions	MATLAB passing	GitHub Actions documentation for setting up badges
Travis CI	build passing	



Orbs > mathworks/matlab@0.4.0

mathworks/matlab@0.4.0 PARTNER

Run MATLAB and Simulink as part of your build pipeline.

Created: October 25, 2019 | **Version Published:** February 4, 2021 | **Releases:** 12

Homepage: https://www.mathworks.com/solutions/continuous-integration.html

Source: https://github.com/mathworks/matlab-circleci-orb

See Orb Licensing

Relevant Training Classes

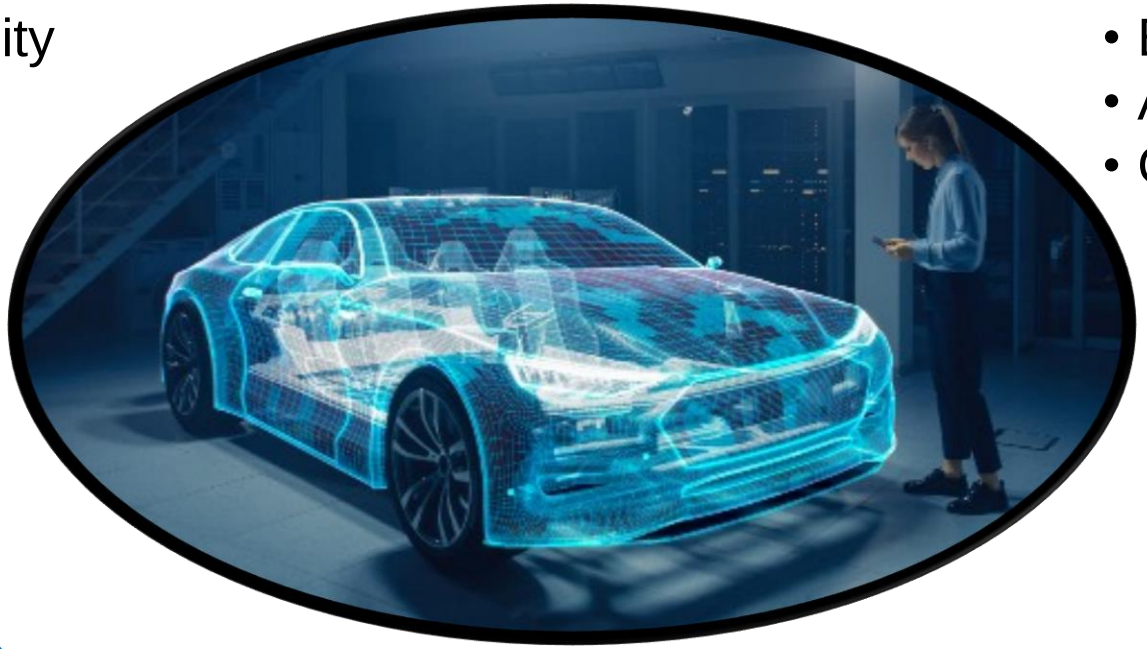
- Simulink Fundamentals – introduction to designing models using Simulink
- Simulink Model Management and Architecture – Requirements Toolbox, Simulink Projects, Architectural Choices, Data Management, Simulink Report Generator
- Simulation-Based Testing with Simulink – includes Simulink Test
- Design Verification with Simulink – Simulink Design Verifier
- Embedded Coder for Production Code Generation – generating and using code from Simulink models
- Polyspace for C/C++ Code Verification – static analysis of hand code and automatically-generated code
- **Applying Model-Based Design for ISO 26262** (available upon request)

Software-Defined Vehicle

demand

Customer expectations

- Clean and Safe mobility
- Digital Life continuity



Technology & Innovation

- Electrification
- Autonomy
- Connectivity

invest

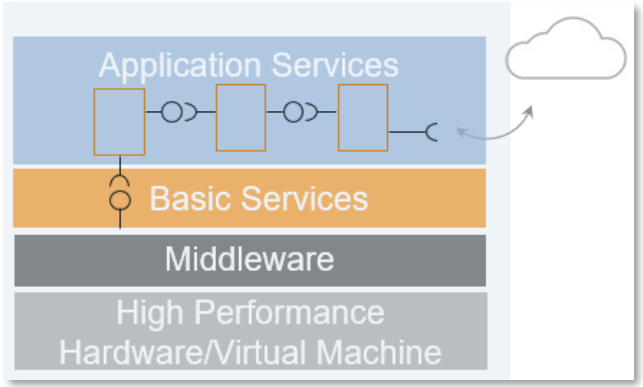
Business opportunity

- App stores, SW features on demand
- SW services subscription plans

monetize

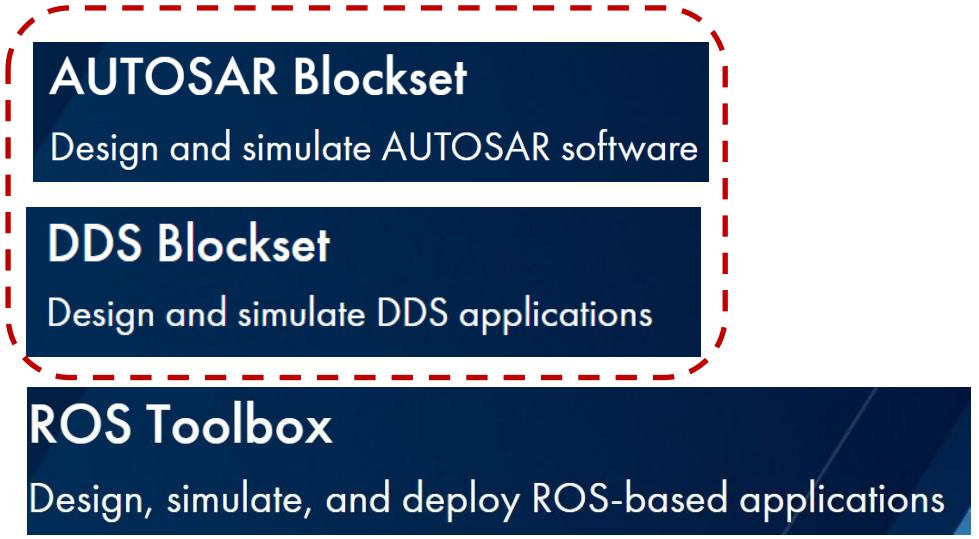
SOA – What's it all about?

- With SOA, applications are standalone processes that provide and/or require services distributed across the vehicle computing platform and the cloud

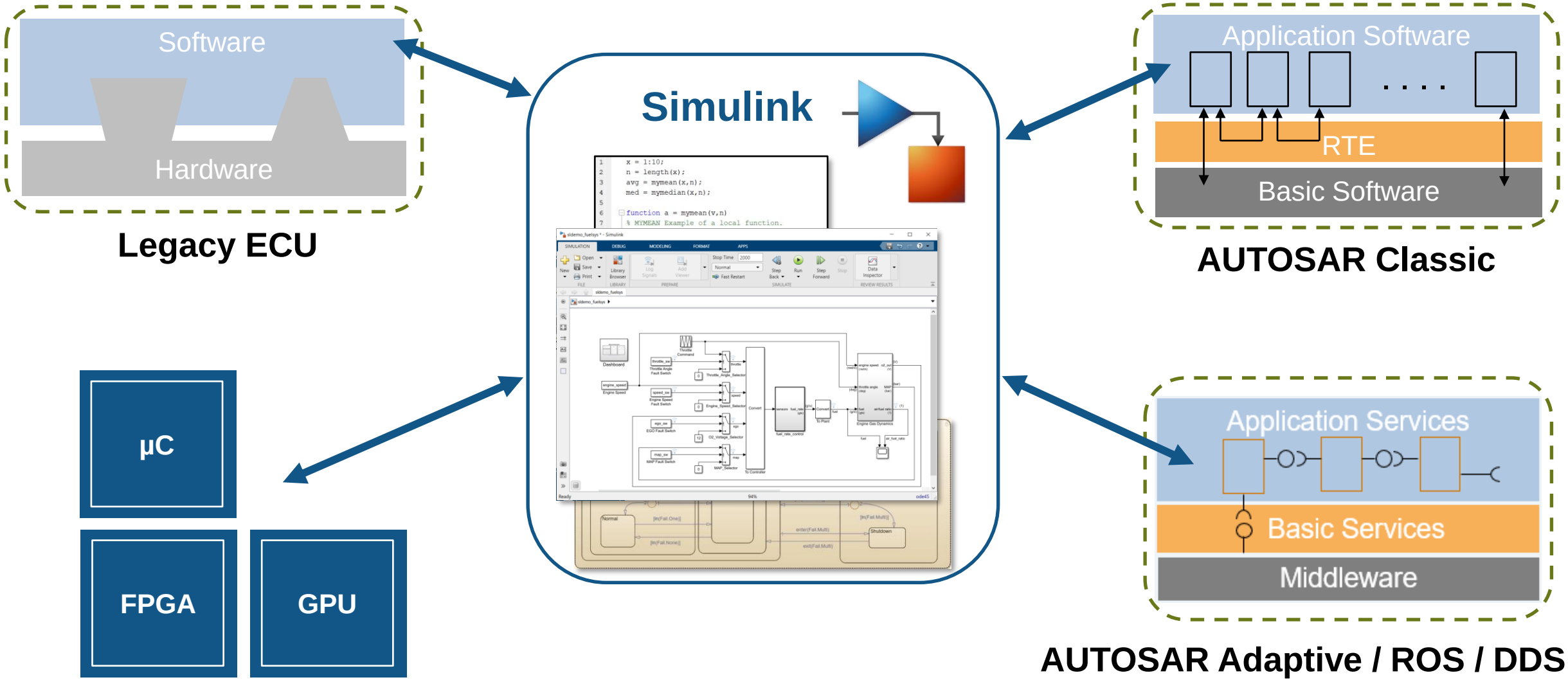


- SOA provides flexibility to add, remove, or update applications without impacting the entire, typically large, software system

- SOA is used by multiple industrial standards:
  - AUTOSAR Adaptive Platform
  - DDS (Data Distribution Services)
  - ROS (Robot Operating System)



Simulink: Deploy software to different targets and standards

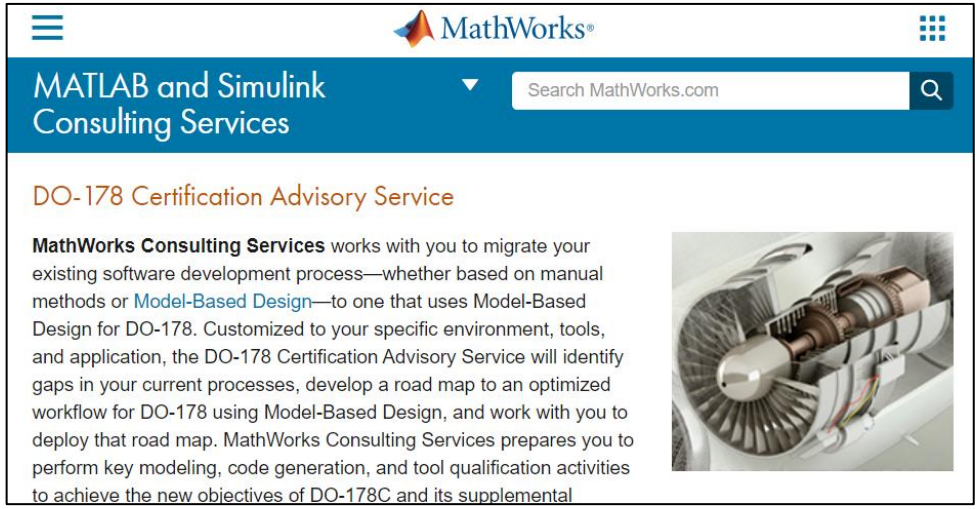


MathWorks Consulting

Certification Advisory Service Overview

- MathWorks Consulting will help you to:
  - Leverage Model-Based Design and supporting tools to their fullest extent to maximize ROI
  - Reduce duplicated and manual effort
  - Avoid common pitfalls by providing proven best practices for Model Based Design, system engineering and development of safety critical software with MBD

ISO 26262 Process Deployment Advisory Service



DO-178 Certification Advisory Service

MathWorks®

Accelerating the pace of engineering and science

For further details, Q&A and feedback kindly reach out to

Gaurav Ahuja
gahuja@mathworks.com
Application Engineering Group
LinkedIn

Rajat Arora
rarora@mathworks.com
Application Engineering Group
LinkedIn

[profile handle] [profile handle] [profile handle] [profile handle] [profile handle]