Toolchain Definition and Integration for ISO 26262-Compliant Development

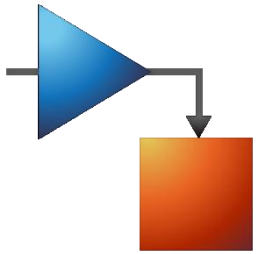
July 1

MathWorks
AUTOMOTIVE
CONFERENCE 2020

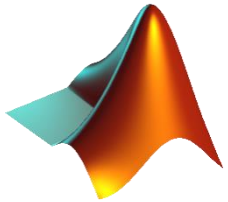
MathWorks®

Introduction

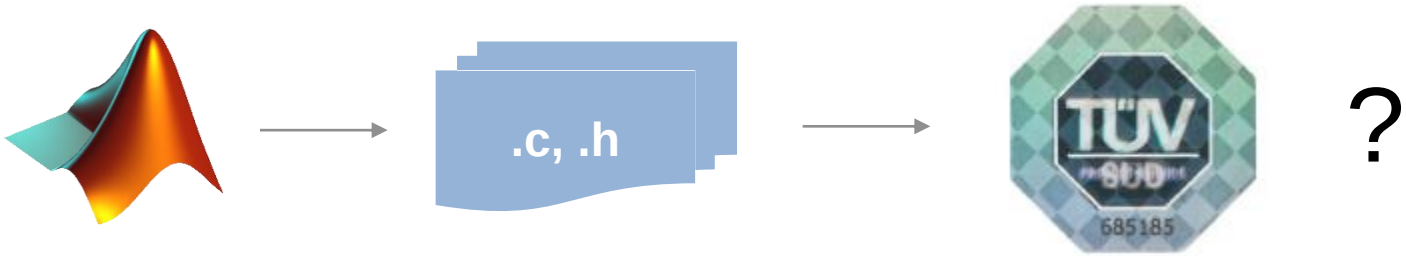
- MathWorks tools like Simulink and Stateflow are [suitable for generating code for ISO 26262 QM to ASIL-D applications]



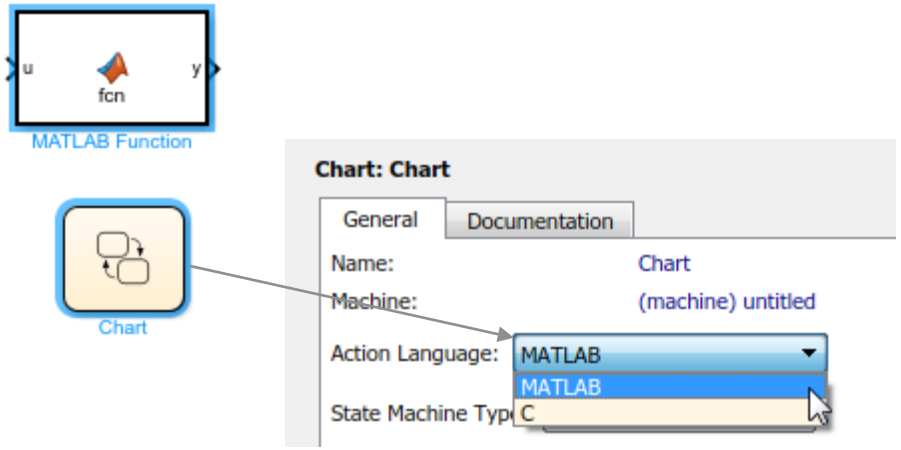
- MATLAB has emerged for AD/ADAS algorithm prototyping
  - A natural language for matrices, image processing, deep learning
  - MATLAB source (text) is also seamless to integrate with Agile workflow tools



- Can we generate certifiable code from MATLAB?



Yes! Through Simulink Integration

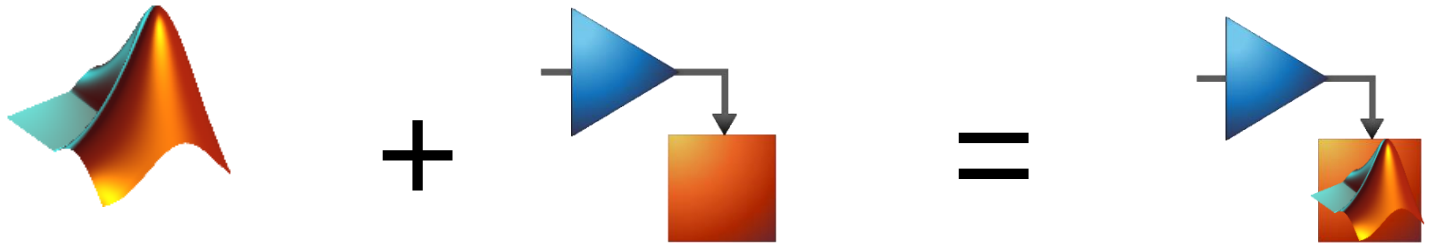


- Called by the MATLAB Function block and/or Stateflow
  - Inlined MATLAB operators
  - External functions
  - Long list of language features that support code generation
  - And functions, including toolboxes like Sensor Fusion, Stats and Machine Learning, Automated Driving, Deep Learning

- MATLAB code generation is supported by our IEC Certification Kit and reference workflow

Best practice

- We can combine these and have the best of both worlds
  - + Richness of the MATLAB language
  - + Rigor of the Simulink family of verification tools



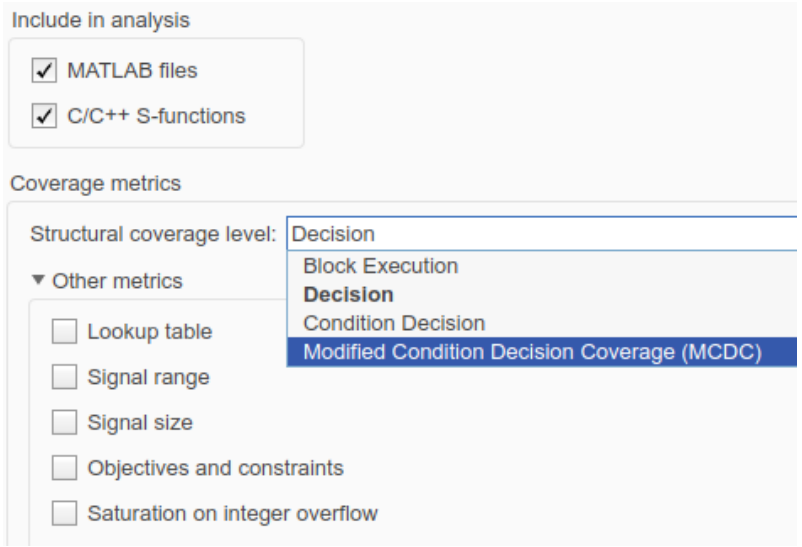
- "I'm a MATLAB user, is Simulink for me?"

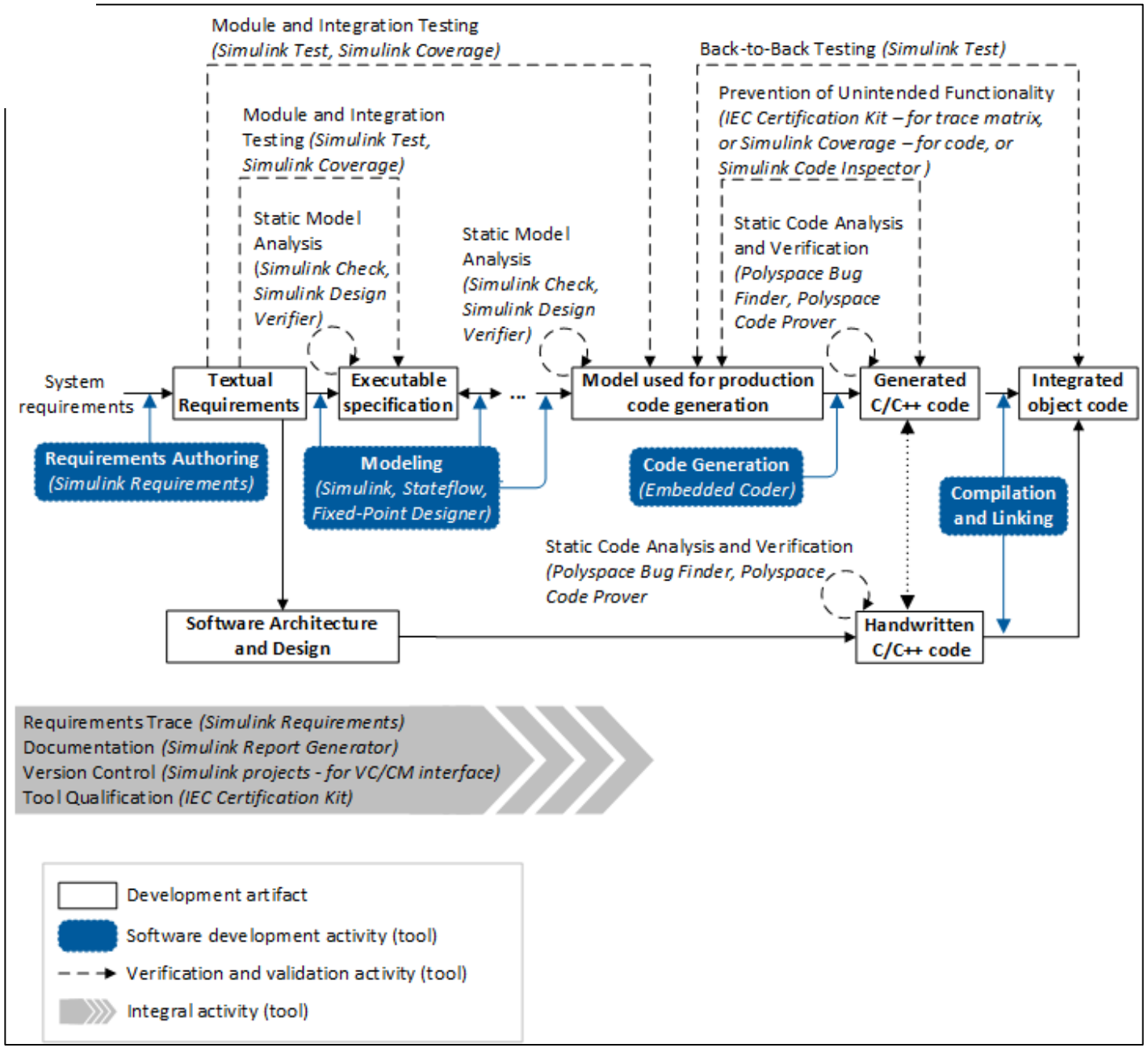
➔ If you need to provide **evidence of conformance**

➔ To define **architecture** around MATLAB algorithms

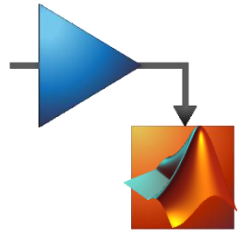
Verification workflow

- Trace requirements ⇔ design ⇔ implementation ⇔ validation
- Meet design & implementation standards
- Show intended and no unintended functionality
  - Coverage is key evidence





MATLAB + Simulink ISO 26262 Workflow



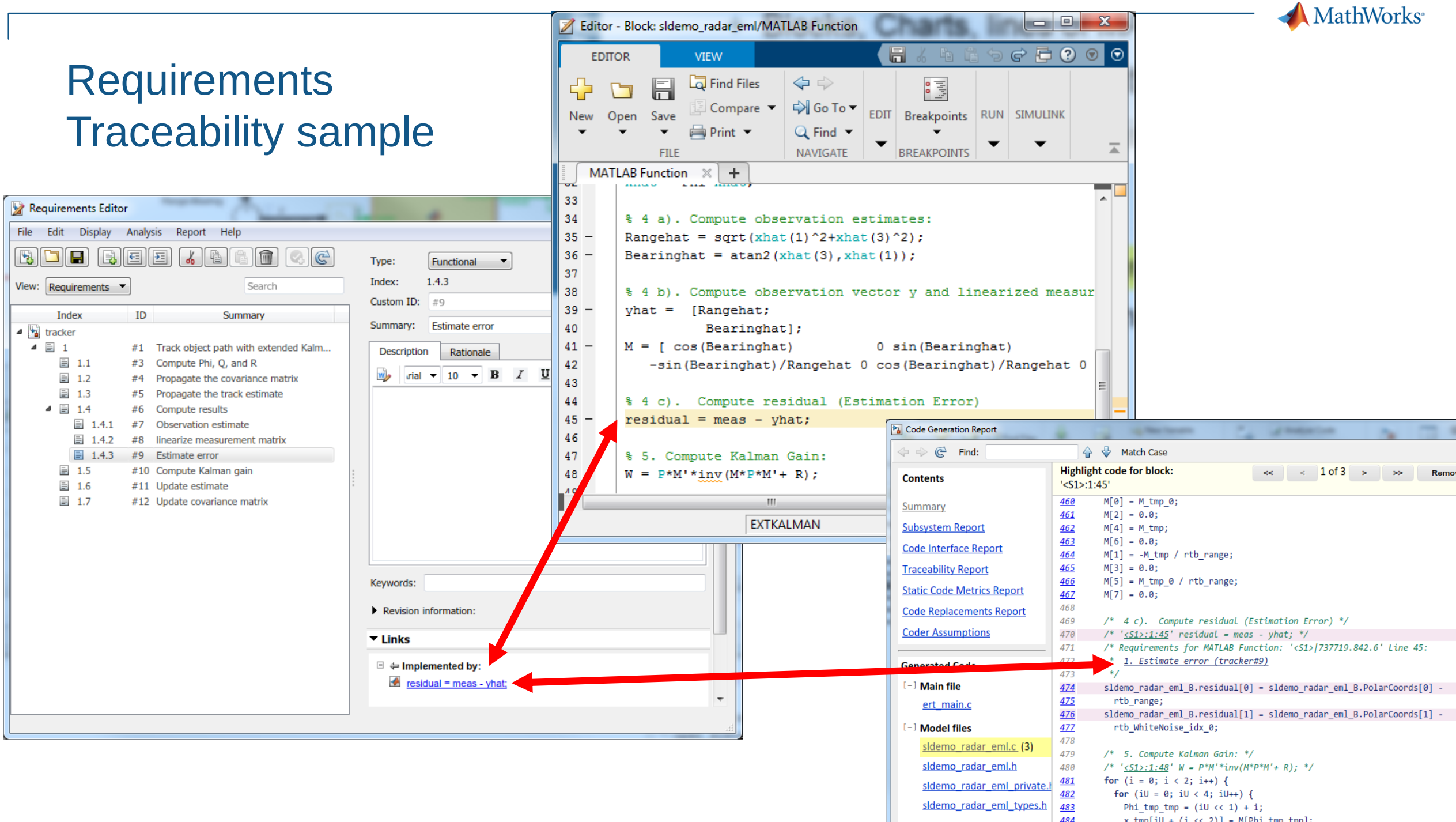
- Our ISO verification activities now support this combined language
  - + Requirements traceability
  - + Design standards
  - + Prove correct functionality
  - + Prove absence of unintended functionality

Traceability

Simulink Requirements

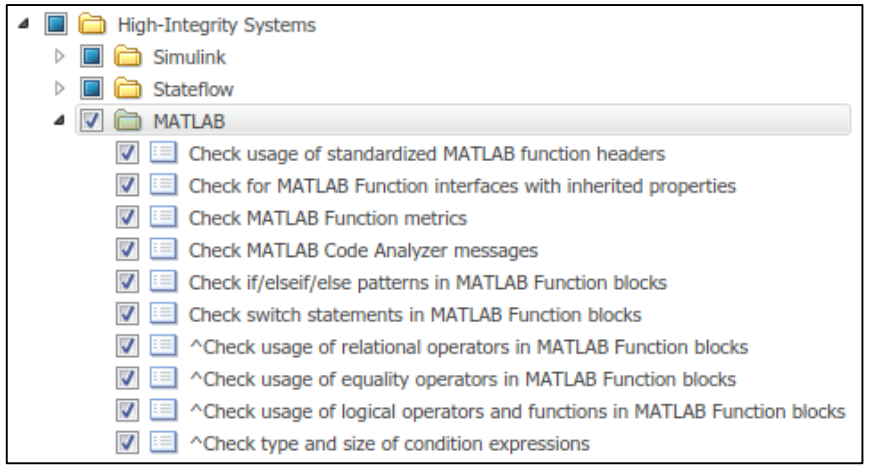
- Simulink Requirements supports authoring, importing/exporting, and linking requirements to **model elements**, test cases (Simulink Test)
  - Blocks, Charts, lines of MATLAB code
- Requirements Traceability report for evidence
- MATLAB source and user comments can be included as generated comments

Requirements Traceability sample



Design and Code Standards

Simulink Check



+ Simulink Check has checks for good MATLAB style and improving code compliance
  + Enforcement of low complexity
  + Enforcement of comment density
  + Strong data typing between MATLAB and Simulink
  + Find logical operators with mixed data types
+ Some MATLAB & Embedded Coder settings for MISRA-C

– MATLAB style guides are limited in scope (MAAB, NASA)

Demonstrate correct functionality

Simulink Requirements

Simulink Test

Simulink Design Verifier

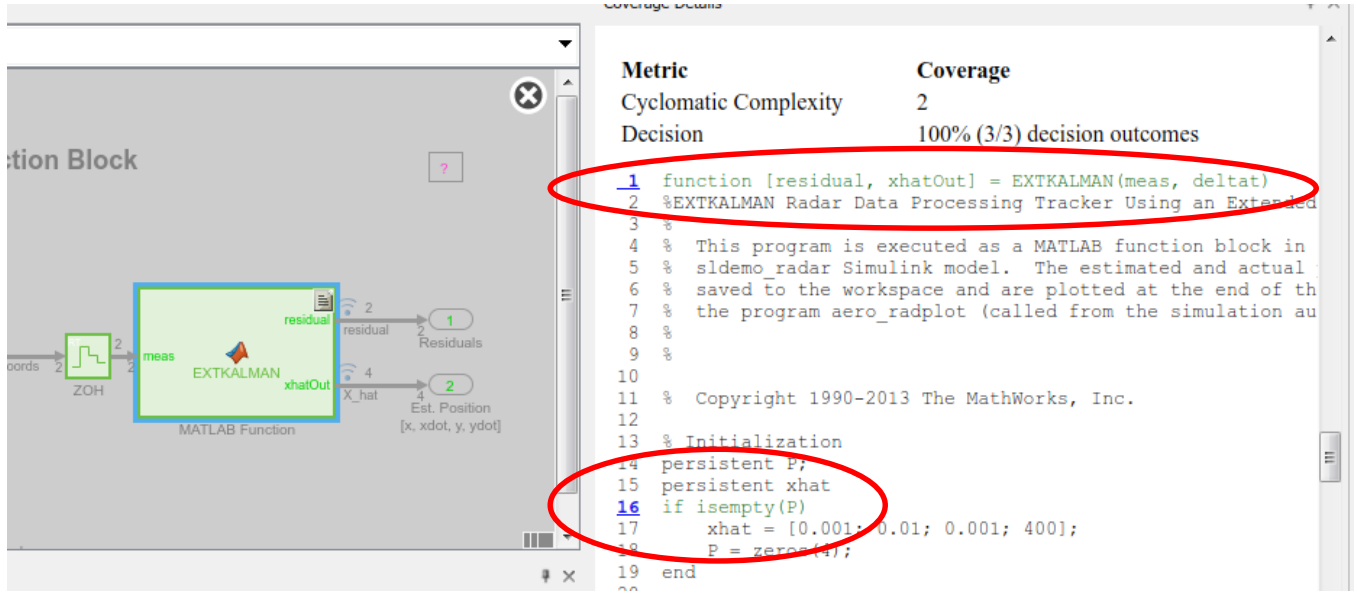
- Requirements-based test authoring, execution via Simulink Test
- Simulink Design Verifier (SLDV) property proving
- SLDV design error detection
- Back to back testing for model to code for Software-in-the-Loop (SIL), Processor-in-the-Loop (PIL)

Demonstrate no unintended functionality

Simulink Coverage

Simulink Design Verifier

- Simulink Coverage to show completeness of test cases
  - Model coverage
  - Code coverage for SIL/PIL
- SLDV can generate missing tests



Summary so far

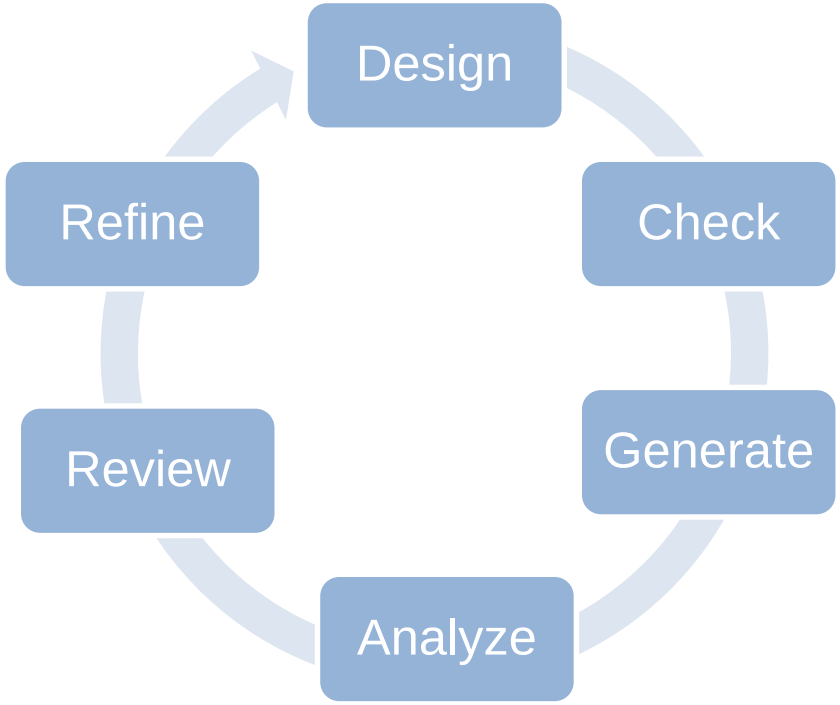
- Customers are successfully using MATLAB in ISO 26262-compliant products today
- Our verification workflow and tools support MATLAB called by Simulink
- But… there are some gaps remaining
  - Potential issues with MISRA-C compliance of code generated from MATLAB
  - Achieving MATLAB vs C code coverage
  - Simplifying Simulink model reviews

Code standards compliance

- Practice is to
  - run model checks
  - generate code
  - analyze compliance

Simulink Check

Polyspace Bug Finder



- Issues discovered?
  - document and proceed
  - rework the algorithm
  - rewrite a compliant function (toolboxes)

- Result is an allowed function list (language subset)
- Process gets more efficient over time

Code coverage

- MATLAB functions can be complex in C/C++

```
% 5. Compute Kalman Gain:
W = P*M'*inv(M*P*M'+ R);
```

- One test case gets coverage in MATLAB, but more required to show no unintended functionality in the generated C

- Strategies include
  - Develop unit tests for feature/function
  - Implement a simpler replacement

```
480     /*  5. Compute Kalman Gain: */
481     /* '<S1>:1:48' W = P*M'*inv(M*P*M'+ R); */
482     for (i = 0; i < 2; i++) {
483       for (iU = 0; iU < 4; iU++) {
484         Phi_tmp_tmp = (int32_T)((int32_T)(iU << 1) + i);
485         x_tmp[(int32_T)(iU + (int32_T)(i << 2))] = M[Phi_tmp_tmp];
486         M_0[Phi_tmp_tmp] = 0.0;
487         Phi_tmp = (int32_T)(iU << 2);
488         M_0[Phi_tmp_tmp] += sldemo_radar_eml_DWork.P[Phi_tmp] * M[i];
489         M_0[Phi_tmp_tmp] += sldemo_radar_eml_DWork.P[(int32_T)(Phi_tmp + 1)] *
490           0.0;
491         M_0[Phi_tmp_tmp] += sldemo_radar_eml_DWork.P[(int32_T)(Phi_tmp + 2)] *
492           M[(int32_T)(i + 4)];
493         M_0[Phi_tmp_tmp] += sldemo_radar_eml_DWork.P[(int32_T)(Phi_tmp + 3)] *
494           0.0;
495       }
496     }
497
498     for (i = 0; i < 2; i++) {
499       for (iU = 0; iU < 2; iU++) {
500         Phi_tmp_tmp = (int32_T)(i << 2);
501         Phi_tmp = (int32_T)((int32_T)(i << 1) + iU);
502         Phi_1[Phi_tmp] = (((x_tmp[(int32_T)(Phi_tmp_tmp + 1)] * M_0[(int32_T)(iU
503           + 2)] + x_tmp[Phi_tmp_tmp] * M_0[iU]) + x_tmp[(int32_T)(Phi_tmp_tmp +
504           2)] * M_0[(int32_T)(iU + 4)]) + x_tmp[(int32_T)(Phi_tmp_tmp + 3)] *
505                        M_0[(int32_T)(iU + 6)]) + R[Phi_tmp];
506       }
507     }
508
509     if (fabs(Phi_1[1]) > fabs(Phi_1[0])) {
510       rtb_range = Phi_1[0] / Phi_1[1];
511       rtb_WhiteNoise_idx_0 = 1.0 / (rtb_range * Phi_1[3] - Phi_1[2]);
512       M_tmp = Phi_1[3] / Phi_1[1] * rtb_WhiteNoise_idx_0;
513       M_tmp_0 = -rtb_WhiteNoise_idx_0;
514       y_idx_2 = -Phi_1[2] / Phi_1[1] * rtb_WhiteNoise_idx_0;
515       rtb_WhiteNoise_idx_0 *= rtb_range;
516     } else {
517       rtb_range = Phi_1[1] / Phi_1[0];
518       rtb_WhiteNoise_idx_0 = 1.0 / (Phi_1[3] - rtb_range * Phi_1[2]);
519       M_tmp = Phi_1[3] / Phi_1[0] * rtb_WhiteNoise_idx_0;
520       M_tmp_0 = -rtb_range * rtb_WhiteNoise_idx_0;
521       y_idx_2 = -Phi_1[2] / Phi_1[0] * rtb_WhiteNoise_idx_0;
522     }
```

Reviewing Simulink models

- Are you reviewing Simulink models?
  - 1-1 or 1-many at desk or in conference rooms?
  - Screen sharing apps?

- Modern workforces are often distributed and busy, making this a challenge

- Tools to manage the review process, such as Gerrit Code Review, are becoming a popular approach

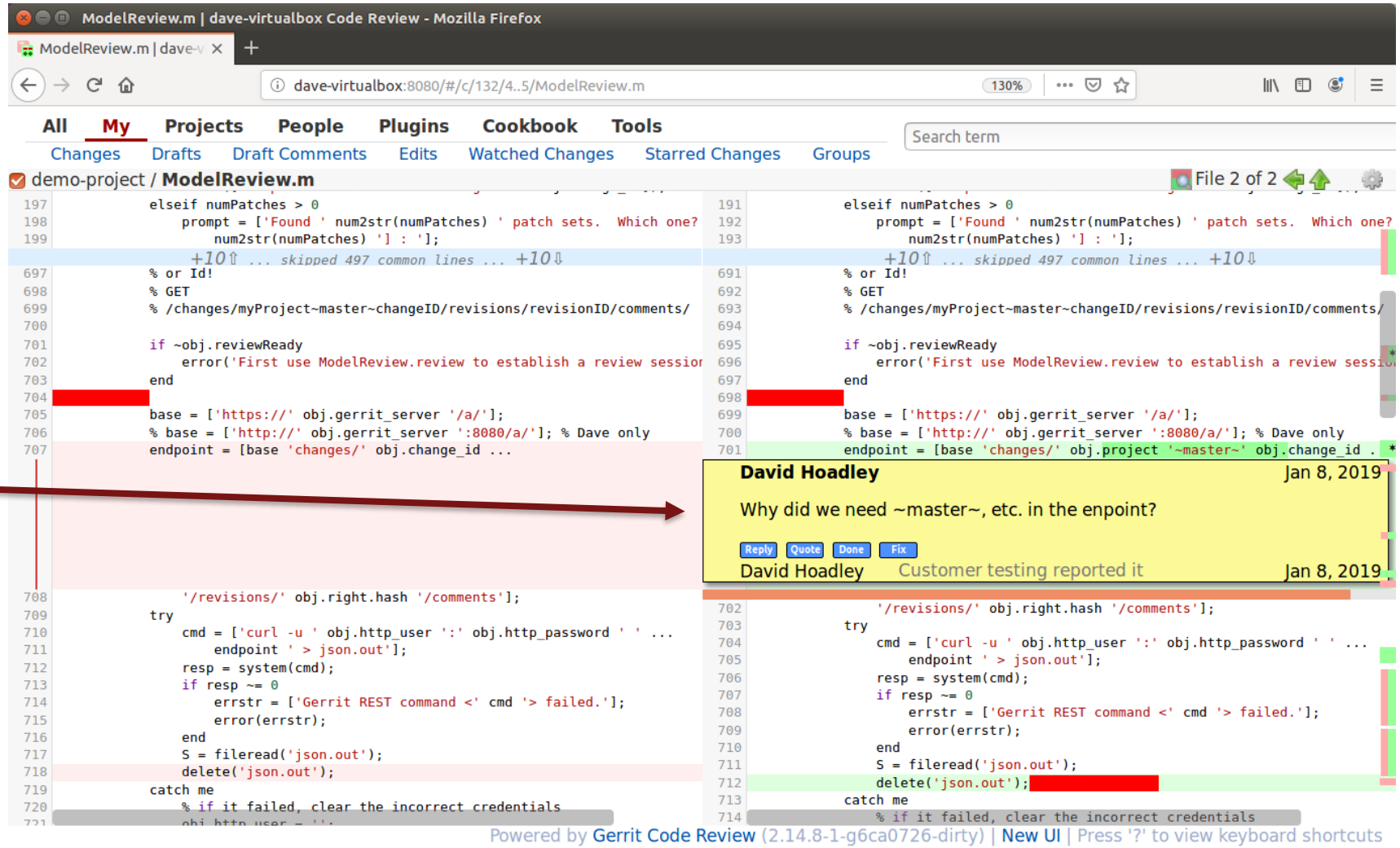
Text-based differences + review comments

Gerrit Code Review

Gerrit implements a web-based review and approval workflow for git patch revisions

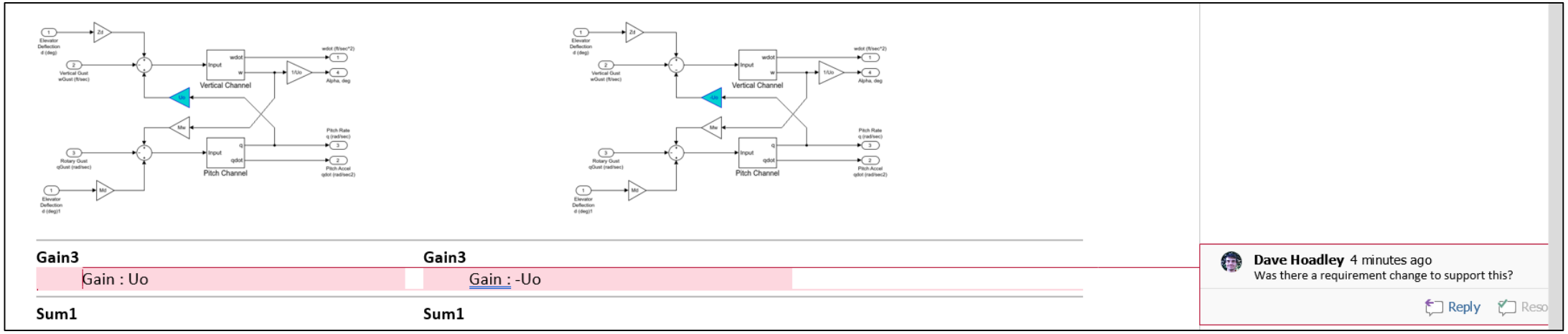
Review comments are shared **in the context** of the source

But, binary formats not supported (.slx)



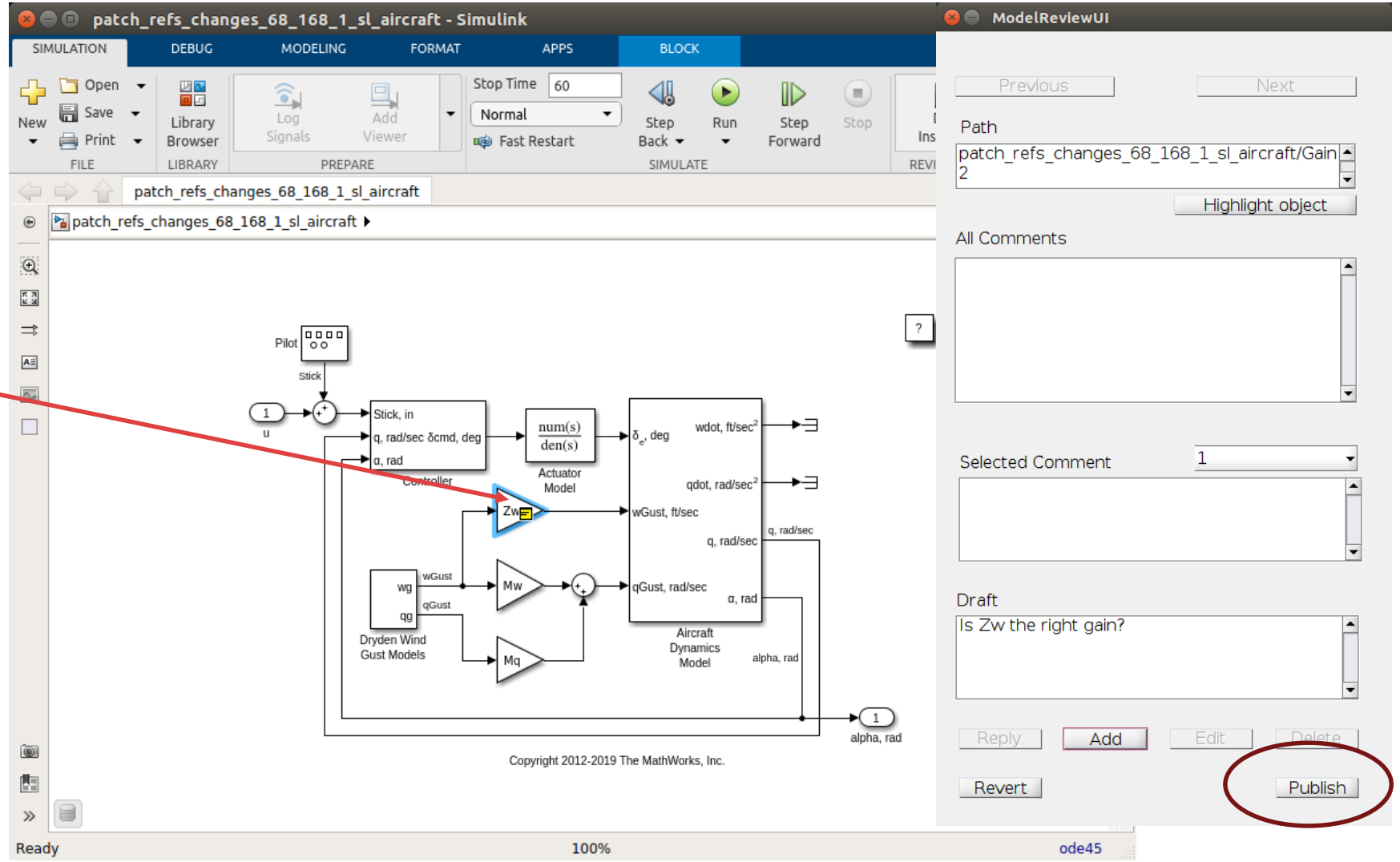
Model reviews with built-in features

- Configure SCM with external diff tool for MATLAB files
  - E.g., "C:\Program Files\MATLAB\R2019a\bin\win64\mlDiff.exe" %LOCAL %PWD %REMOTE
  - Note this will reuse a running MATLAB not start a new instance
- Publish model comparison to MS Word format
- Annotate and share Word document with comments/replies



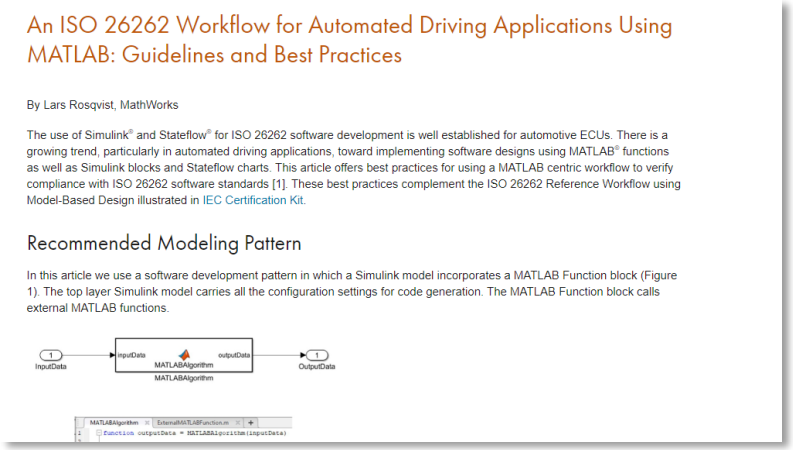
Extending this concept *into* Simulink

- Custom add-on to Simulink context menu
- Block badge indicates comment attached
- Publish to Gerrit when ready to share



Summary redux

- Customers are successfully using Simulink **AND MATLAB** in ISO 26262-compliant products today
- There are some gaps remaining
  - Potential issues with MISRA-C compliance of code generated from MATLAB
  - Achieving MATLAB to C code coverage
  - Simplifying Simulink model reviews MathWorks Consulting Services
- Best Practices Technical paper



An ISO 26262 Workflow for Automated Driving Applications Using MATLAB: Guidelines and Best Practices

By Lars Rosqvist, MathWorks

The use of Simulink® and Stateflow® for ISO 26262 software development is well established for automotive ECUs. There is a growing trend, particularly in automated driving applications, toward implementing software designs using MATLAB® functions as well as Simulink blocks and Stateflow charts. This article offers best practices for using a MATLAB centric workflow to verify compliance with ISO 26262 software standards [1]. These best practices complement the ISO 26262 Reference Workflow using Model-Based Design illustrated in IEC Certification Kit.

Recommended Modeling Pattern

In this article we use a software development pattern in which a Simulink model incorporates a MATLAB Function block (Figure 1). The top layer Simulink model carries all the configuration settings for code generation. The MATLAB Function block calls external MATLAB functions.

Q&A

Do you use MATLAB code in your ISO 26262 components?

- Yes
- No
- No, but planning to

Please contact us with questions

lrosqvis@mathworks.com

MathWorks®