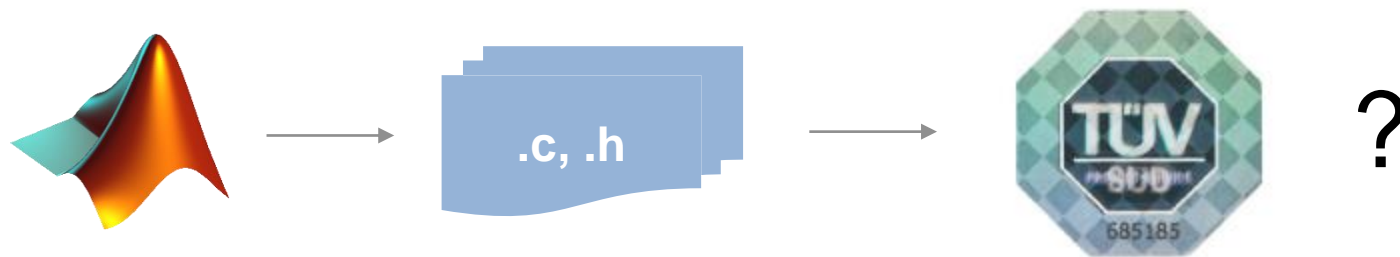
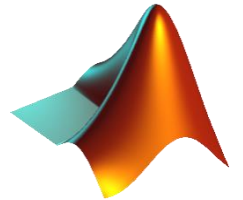
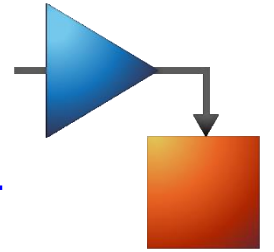


Toolchain Definition and Integration for ISO 26262-Compliant Development

Dave Hoadley, PhD
June 2020

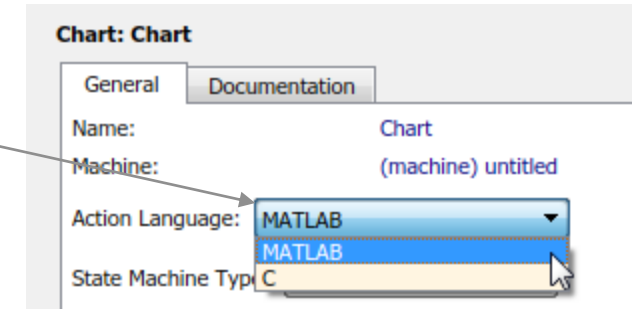
Introduction

- MathWorks tools like Simulink and Stateflow are established as [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?



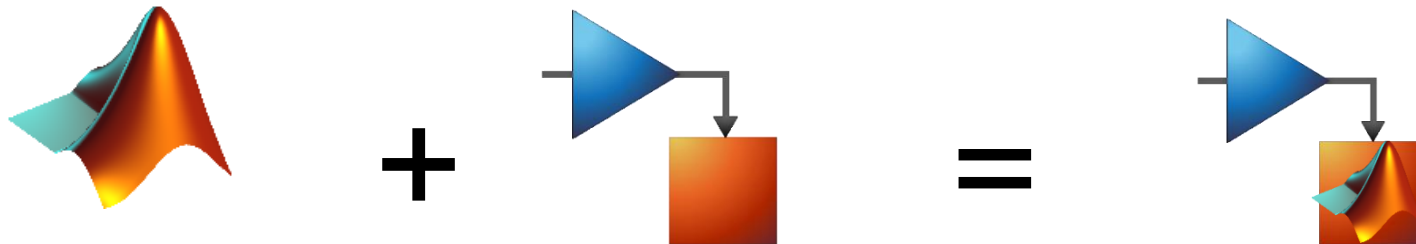
Yes! MATLAB and Simulink Integration

- 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 (Simulink) reference workflow



Algorithm Designer Win-win

- 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 \leftrightarrow design \leftrightarrow implementation \leftrightarrow validation
- Meet design & implementation standards
- Show intended and no unintended functionality
 - Coverage is key evidence

Include in analysis

- MATLAB files
- C/C++ S-functions

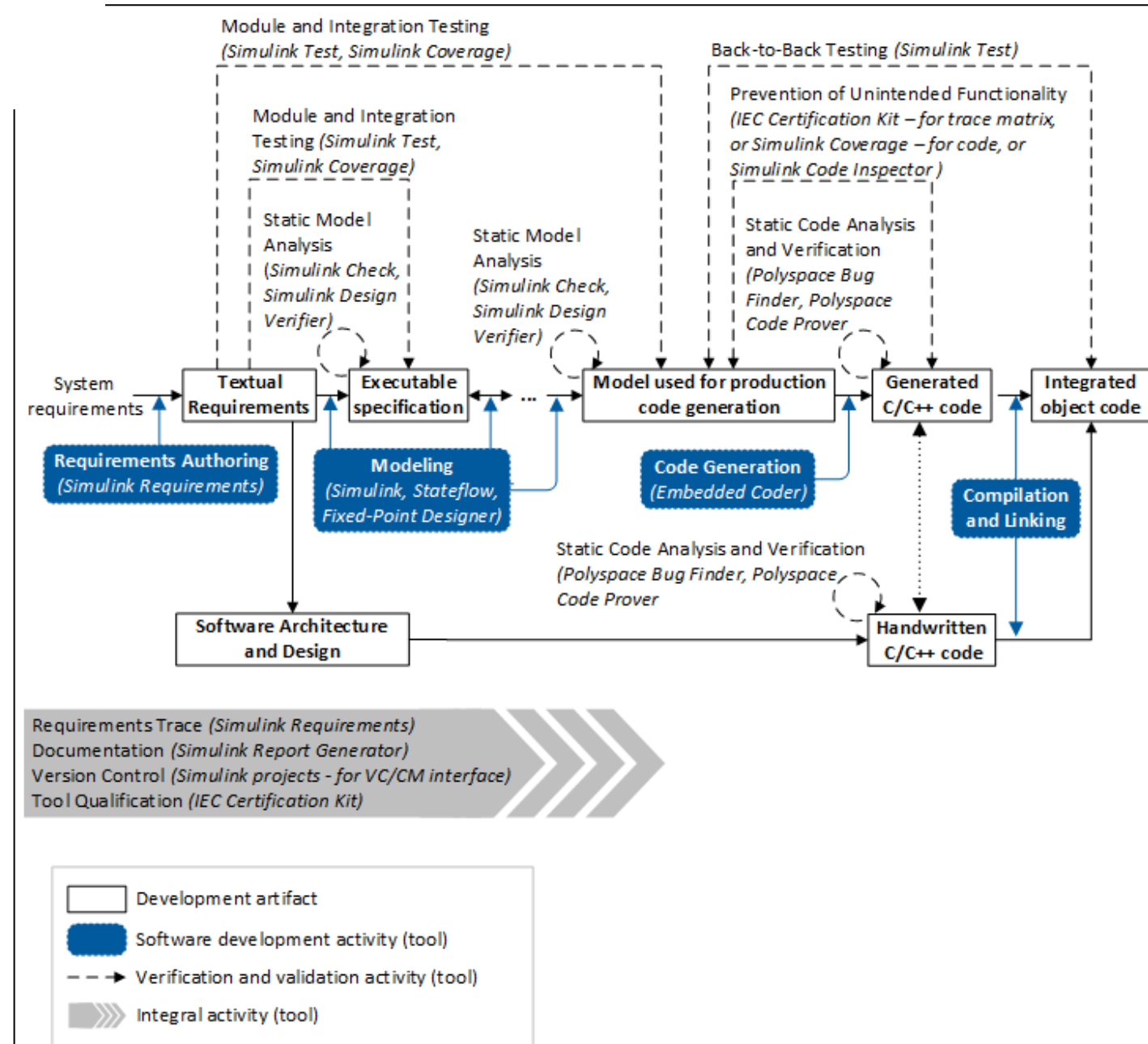
Coverage metrics

Structural coverage level: Decision

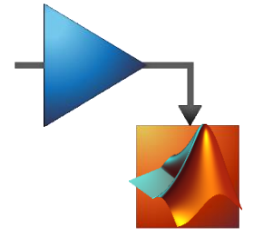
Other metrics

- Lookup table
- Signal range
- Signal size
- Objectives and constraints
- Saturation on integer overflow

Block Execution
Decision
 Condition Decision
 Modified Condition Decision Coverage (MCDC)



MATLAB + Simulink ISO 26262 Workflow



- Our reference workflow supports this combined language
 - + Requirements traceability
 - + Design standards
 - + Prove correct functionality
 - + Prove absence of unintended functionality



[This Photo](#) by Unknown Author is licensed under [CC BY-SA](#)

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

Requirements Editor

File Edit Display Analysis Report Help

View: Requirements Search

Index	ID	Summary
tracker		
1	#1	Track object path with extended Kalm...
1.1	#3	Compute Phi, Q, and R
1.2	#4	Propagate the covariance matrix
1.3	#5	Propagate the track estimate
1.4	#6	Compute results
1.4.1	#7	Observation estimate
1.4.2	#8	linearize measurement matrix
1.4.3	#9	Estimate error
1.5	#10	Compute Kalman gain
1.6	#11	Update estimate
1.7	#12	Update covariance matrix

Type: Functional

Index: 1.4.3

Custom ID: #9

Summary: Estimate error

Description Rationale

Keywords:

Revision information:

Links

Implemented by:

- residual = meas - yhat;

Editor - Block: sldemo_radar_eml/MATLAB Function

EDITOR VIEW

FILE NAVIGATE BREAKPOINTS RUN SIMULINK

```

33
34 % 4 a). Compute observation estimates:
35 Rangehat = sqrt(xhat(1)^2+xhat(3)^2);
36 Bearinghat = atan2(xhat(3),xhat(1));
37
38 % 4 b). Compute observation vector y and linearized measur
39 yhat = [Rangehat;
40         Bearinghat];
41 M = [ cos(Bearinghat)      0 sin(Bearinghat)
42       -sin(Bearinghat)/Rangehat 0 cos(Bearinghat)/Rangehat 0
43
44 % 4 c). Compute residual (Estimation Error)
45 residual = meas - yhat;
46
47 % 5. Compute Kalman Gain:
48 W = P*M'*inv(M*P*M'+ R);

```

EXTKALMAN

Code Generation Report

Find: Match Case

Highlight code for block: '<S1>:1:45'

Contents

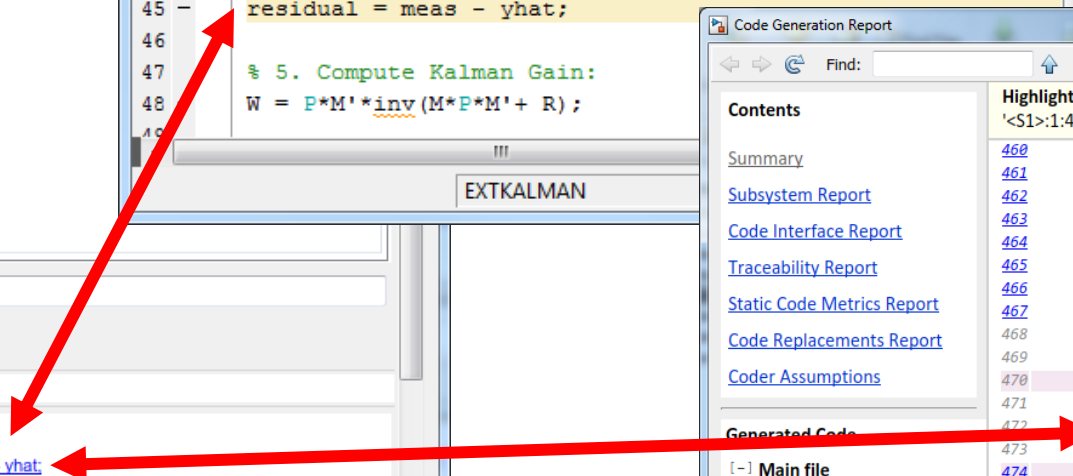
- Summary
- Subsystem Report
- Code Interface Report
- Traceability Report
- Static Code Metrics Report
- Code Replacements Report
- Coder Assumptions

Generated Code

```

460 M[0] = M_tmp_0;
461 M[2] = 0.0;
462 M[4] = M_tmp;
463 M[6] = 0.0;
464 M[1] = -M_tmp / rtb_range;
465 M[3] = 0.0;
466 M[5] = M_tmp_0 / rtb_range;
467 M[7] = 0.0;
468
469 /* 4 c). Compute residual (Estimation Error) */
470 /* '<S1>:1:45' residual = meas - yhat; */
471 /* Requirements for MATLAB Function: '<S1>|737719.842.6' Line 45:
472 * 1. Estimate error (tracker#9)
473 */
474 sldemo_radar_eml_B.residual[0] = sldemo_radar_eml_B.PolarCoords[0] -
475 rtb_range;
476 sldemo_radar_eml_B.residual[1] = sldemo_radar_eml_B.PolarCoords[1] -
477 rtb_WhiteNoise_idx_0;
478
479 /* 5. Compute Kalman Gain: */
480 /* '<S1>:1:48' W = P*M'*inv(M*P*M'+ R); */
481 for (i = 0; i < 2; i++) {
482     for (iU = 0; iU < 4; iU++) {
483         Phi_tmp_tmp = (iU < 1) + i;
484         x tmp[iU + (i < 2)] = M[Phi tmp tmp];

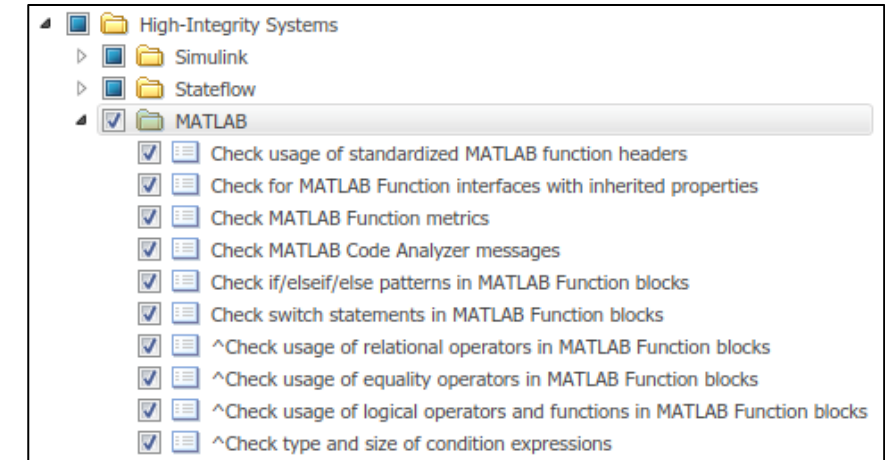
```



Design and Code Standards

Simulink Check

- + Simulink Check has checks for 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 guidelines are emerging (JMAAB)
- More MATLAB checks are needed



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

The screenshot displays the Simulink Coverage tool interface. On the left, a Simulink model is shown with a 'MATLAB Function' block highlighted in green. The block is labeled 'EXTKALMAN' and has two outputs: 'residual' and 'xhatOut'. The 'residual' output is connected to a 'Residuals' scope, and the 'xhatOut' output is connected to an 'Est. Position' scope. On the right, the 'Coverage Details' window is open, showing a table of metrics and their coverage. The 'Decision' metric is highlighted with a red circle, showing '100% (3/3) decision outcomes'. Below the table, the MATLAB code for the 'EXTKALMAN' function is displayed, with two lines circled in red: line 1, 'function [residual, xhatOut] = EXTKALMAN(meas, deltat);' and line 16, 'if isempty(P)'. The code also includes comments and initialization logic for a persistent variable 'P'.

Metric	Coverage
Cyclomatic Complexity	2
Decision	100% (3/3) decision outcomes

```

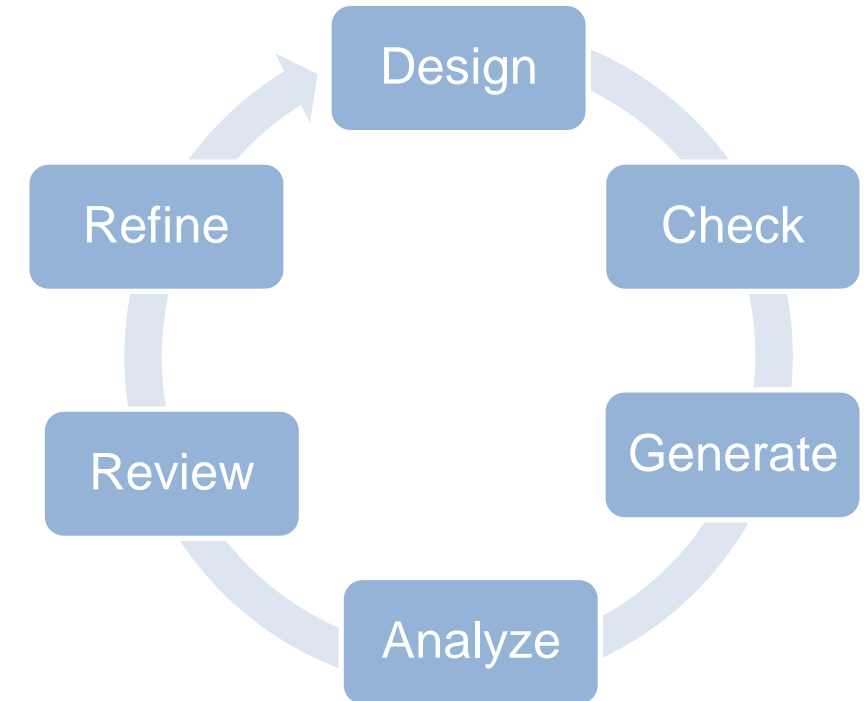
1 function [residual, xhatOut] = EXTKALMAN(meas, deltat)
2 %EXTKALMAN Radar Data Processing Tracker Using an Extended
3 %
4 % This program is executed as a MATLAB function block in
5 % sldemo_radar Simulink model. The estimated and actual
6 % saved to the workspace and are plotted at the end of th
7 % the program aero_radplot (called from the simulation au
8 %
9 %
10
11 % Copyright 1990-2013 The MathWorks, Inc.
12
13 % Initialization
14 persistent P;
15 persistent xhat
16 if isempty(P)
17     xhat = [0.001; 0.01; 0.001; 400];
18     P = zeros(4);
19 end
20
  
```

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 comparison reviews

Code standards compliance

- Practice is to
 - run model checks **Simulink Check**
 - generate code
 - analyze compliance **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

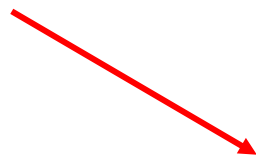
- MATLAB functions can be complex in C/C++

```

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



```

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  }
    
```

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

Reviewing Simulink models

- Classic approaches
 - 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



[This Photo](#) by Unknown Author is licensed under [CC BY-SA-NC](#)



[This Photo](#) by Unknown Author is licensed under [CC BY](#)

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)

The screenshot shows a web browser window displaying a Gerrit Code Review for a file named 'ModelReview.m'. The interface includes a navigation bar with tabs for 'All', 'My', 'Projects', 'People', 'Plugins', 'Cookbook', and 'Tools'. Below this is a search bar and a list of filters like 'Changes', 'Drafts', 'Draft Comments', etc. The main content area shows a diff between two versions of the code. The left side represents the original code, and the right side shows the proposed changes. A review comment is visible on the right side of the diff, associated with a specific line of code. The comment is from David Hoadley, dated Jan 8, 2019, and asks: 'Why did we need ~master~, etc. in the endpoint?'. Below the comment are buttons for 'Reply', 'Quote', 'Done', and 'Fix'. The code diff shows changes in the 'endpoint' variable definition, with a red box highlighting the change in the original code and a green box highlighting the change in the proposed code.

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

Gain3	Gain3
Gain : Uo	Gain : -Uo
Sum1	Sum1

Dave Hoadley 4 minutes ago
Was there a requirement change to support this?

Reply
 Reso

Extending this concept *into* Simulink

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

The screenshot displays the Simulink Model Review UI. The main window shows a Simulink model titled "Aircraft Longitudinal Flight Control". A red arrow points from the "Block badge" in the list to the "Zw" block in the model. The right-hand panel, titled "ModelReviewUI", shows the "Path" field set to "patch_refs_changes_68_168_1_sl_aircraft/Gain/2". Below this, the "All Comments" section is empty, and the "Selected Comment" dropdown is set to "1". The "Draft" section contains the text "Is Zw the right gain?". At the bottom of the panel, the "Publish" button is circled in red.

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
- See [Best practices for Simulink and MATLAB for ISO-26262](#) for advice



Contact info and poll questions

- How are you reviewing Simulink models today?
 1. Ad hoc
 2. Screen sharing/model discussion
 3. Reviewing reports offline (html, etc.)
 4. Simulink comparison tool
 5. 3rd party model comparison tool
 6. Other

Please contact me with questions at dhoadley@mathworks.com and let me know if you would like to have a follow-up conversation

