



Collaborative Software Development in MATLAB and Simulink

Adam Sifounakis
MATLAB Language Product Manager



How complex are your projects?

- Hundreds of files?
- Many file dependencies?
- Complex setup required?
- ...?

How many people are involved in your project?

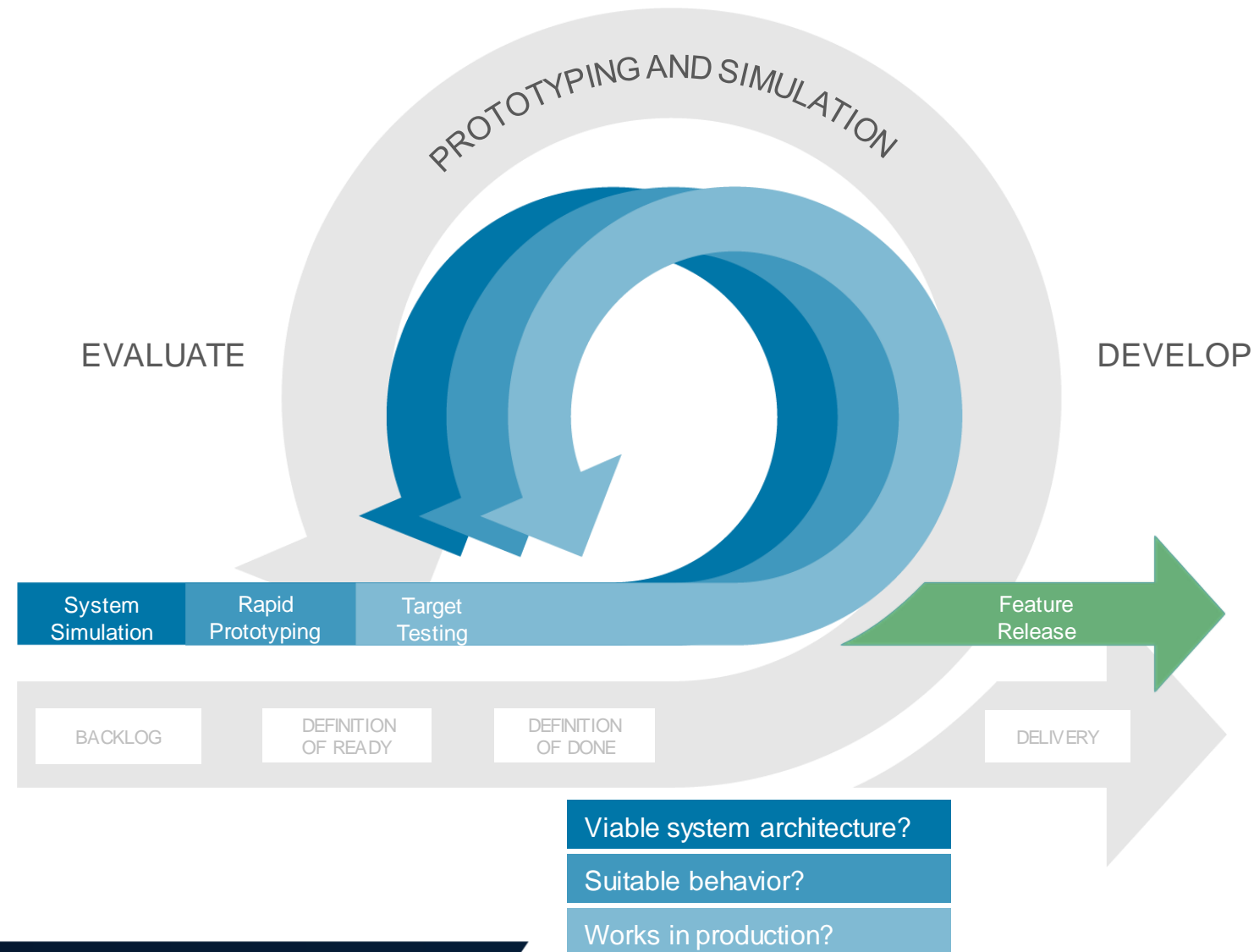
- Dozens of developers?
- Cross-disciplinary teams?
- Teams across the world?
- ...?

How do you ensure project quality?

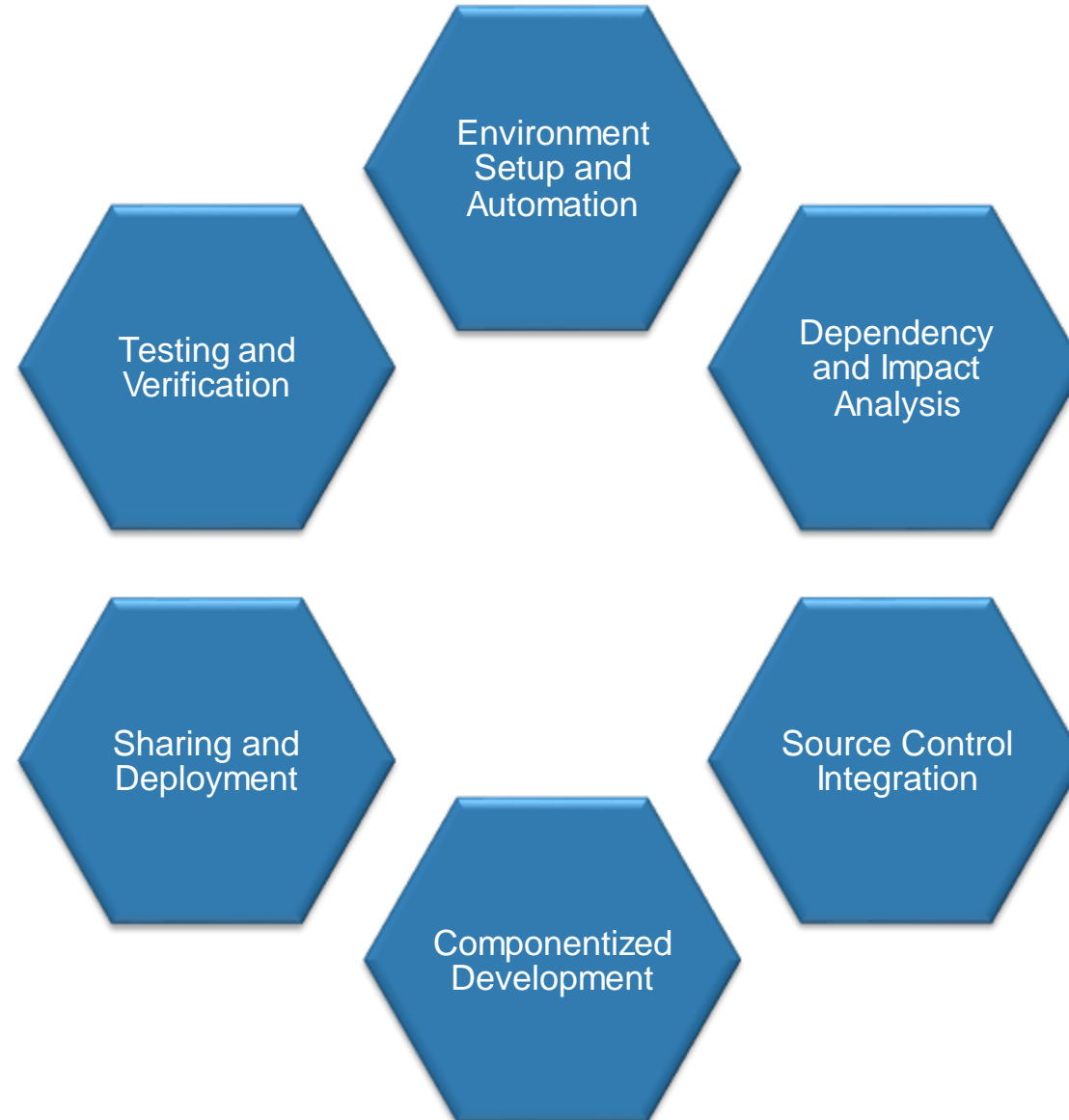
- Systematic testing?
- Coding/modeling standards?
- Regulatory oversight?
- Trust that it just works?
- ...?

Develop quality software with MATLAB and Simulink

- Good software development practices help improve code and model quality
- The tools and practices we discuss today support Agile development workflows




Robust, collaborative development requires...



Robust, collaborative development requires...



Agenda

	Setting up your development environment
	Managing team workflows
	Developing better code and models
	Testing and verification

Development Challenges

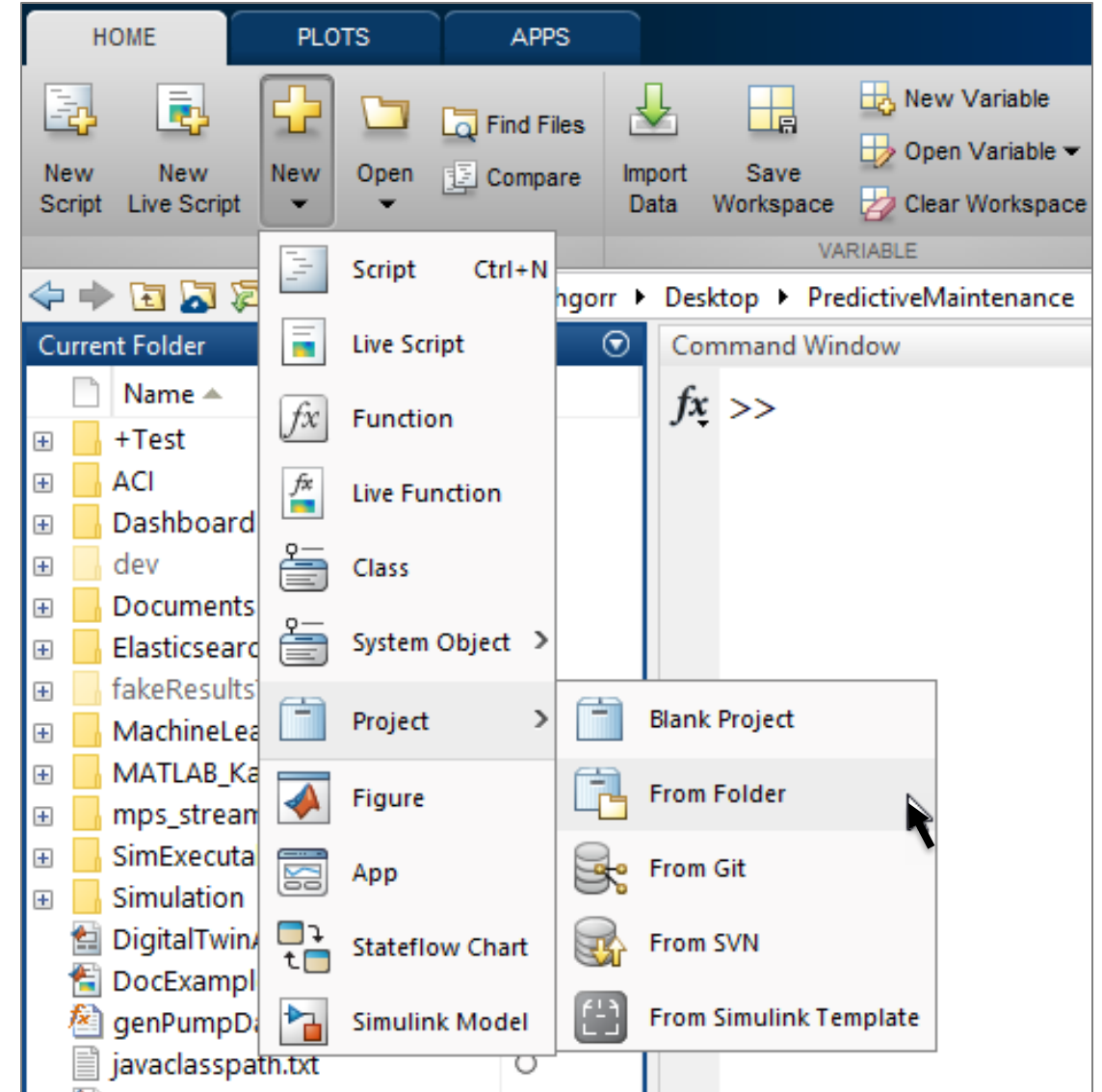
“It works on my computer, but not on yours...”

- Incomplete set of files?
- Which files are missing?
- Different environment?
- How to get started with a project?
- ...



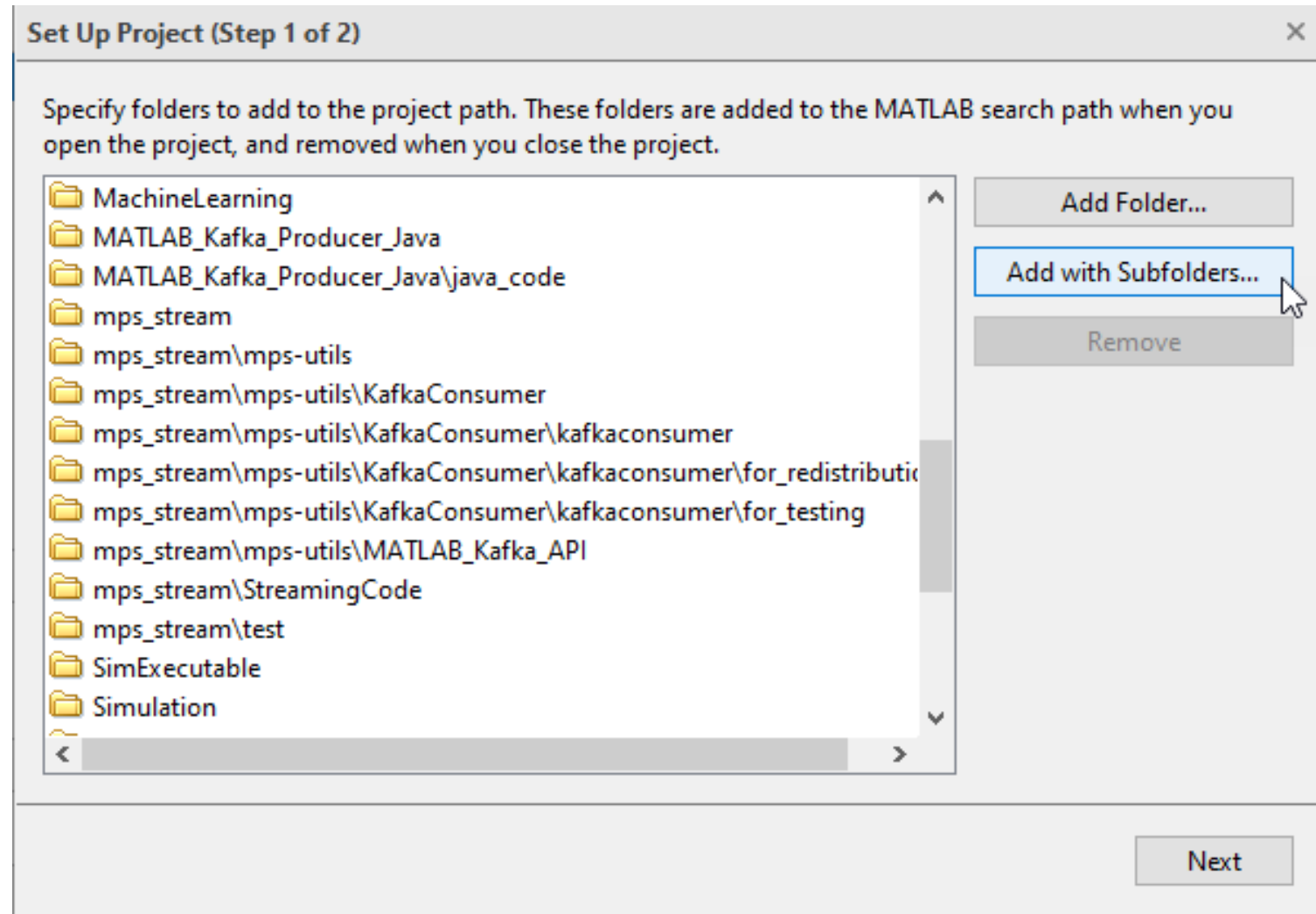
Managing Your Work with Projects

1. Create project



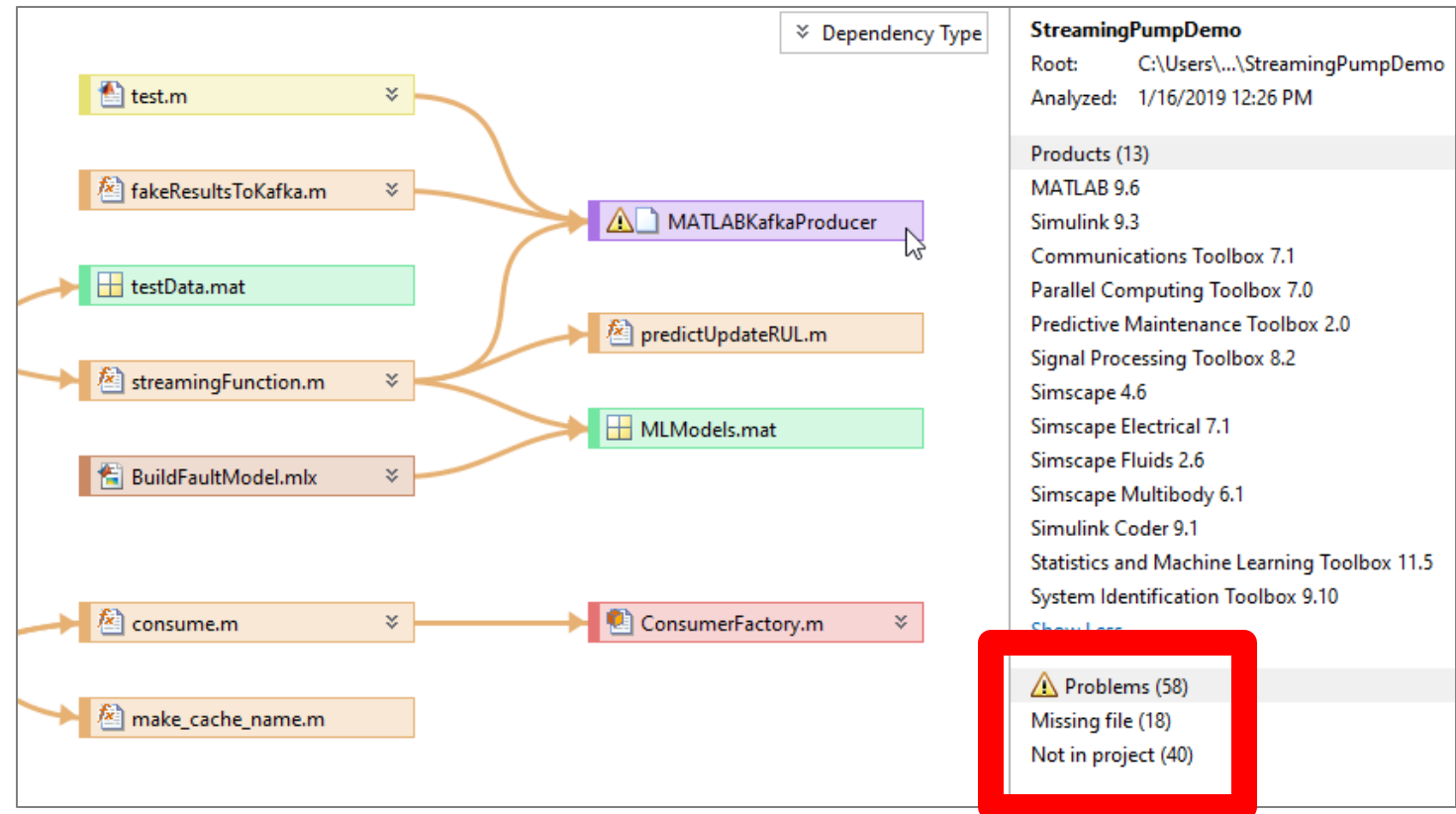
Managing Your Work with Projects

1. Create project
2. Set path and startup tasks



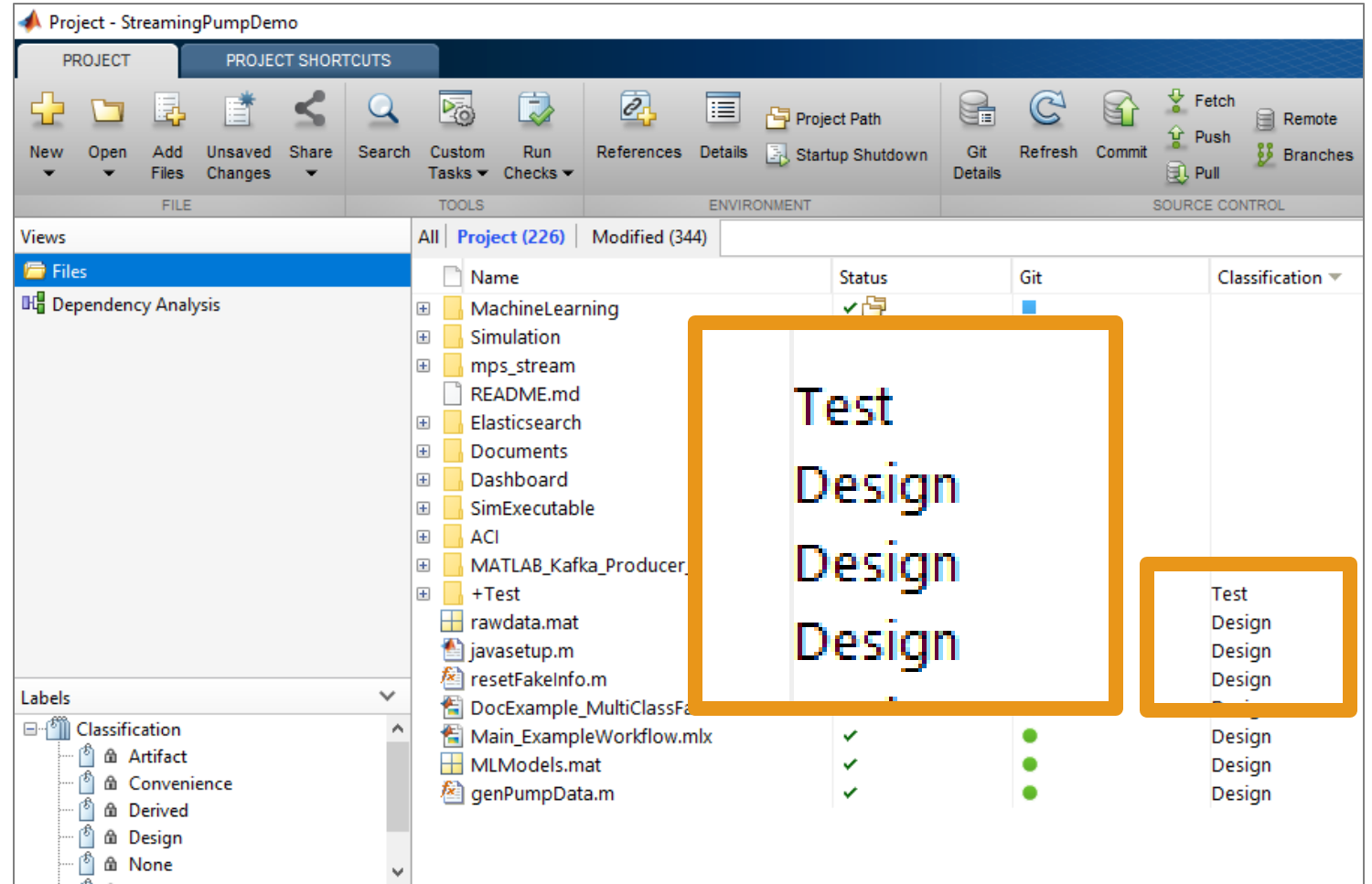
Managing Your Work with Projects

1. Create project
2. Set path and startup tasks
3. Explore dependencies



Managing Your Work with Projects

1. Create project
2. Set path and startup tasks
3. Explore dependencies
4. Label files

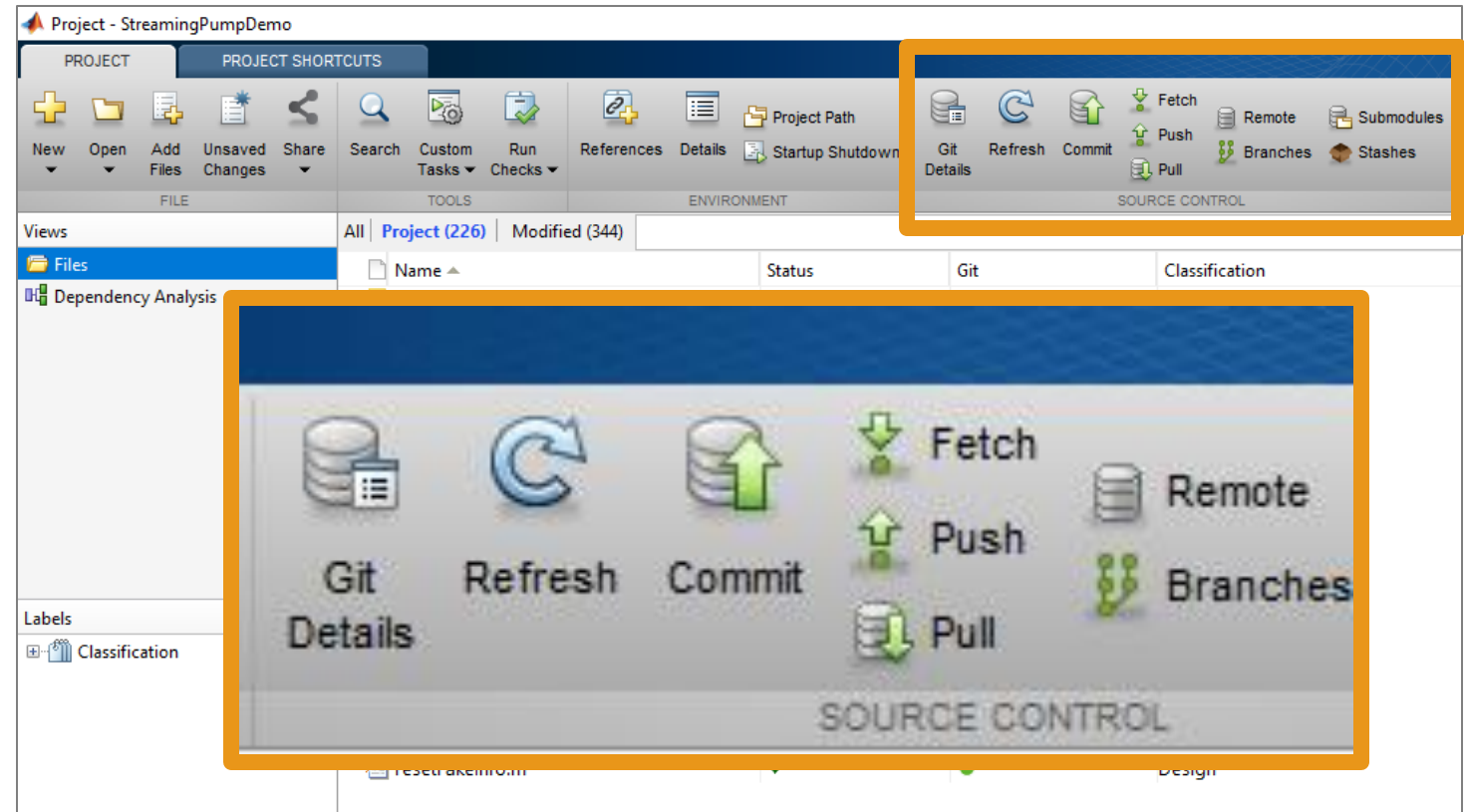


Identify and run tests locally

...and on Continuous Integration (CI) servers

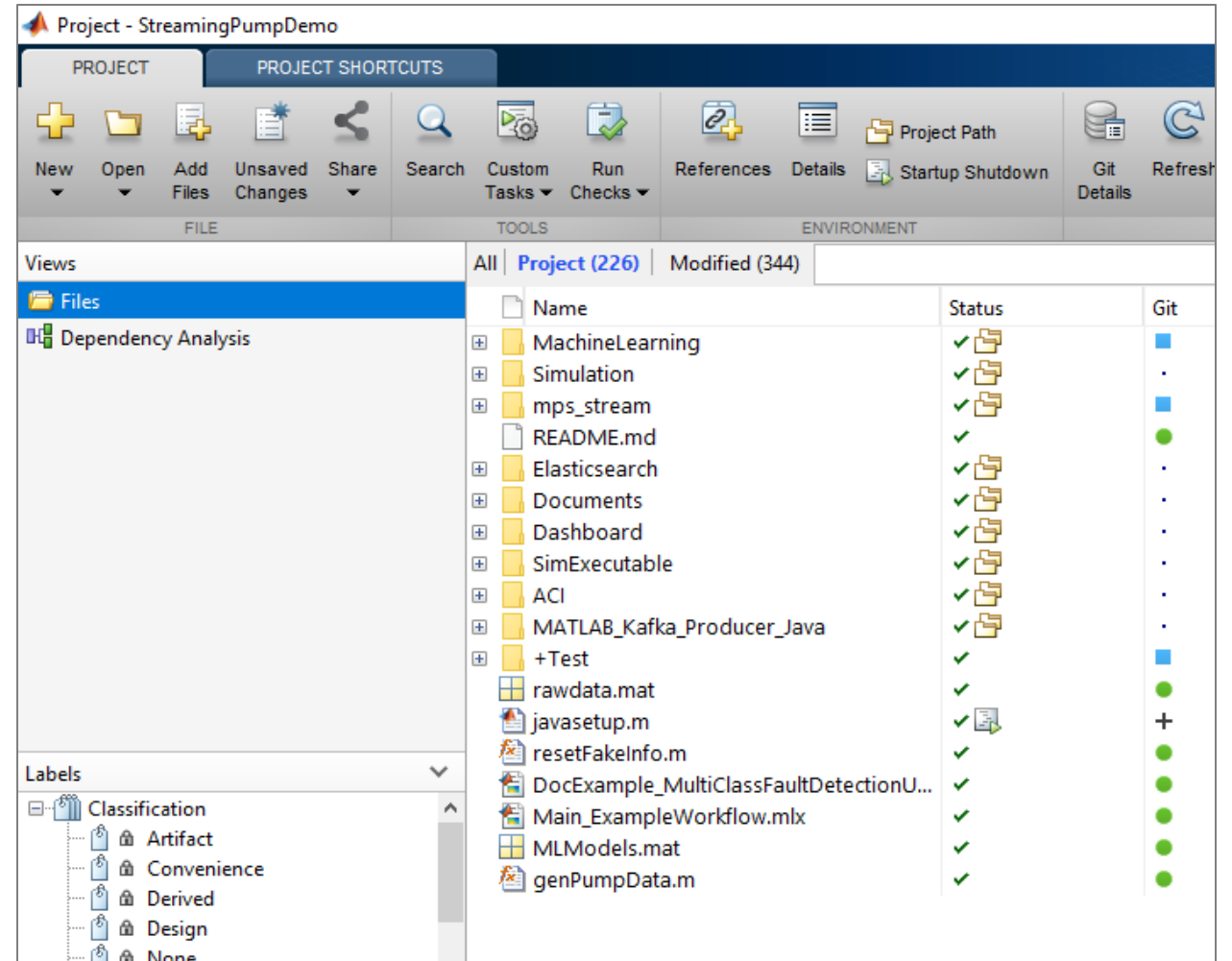
Managing Your Work with Projects

1. Create project
2. Set path and startup tasks
3. Explore dependencies
4. Label files
5. Integrate source control




Projects in MATLAB and Simulink

- Manage your files and path
- Analyze file dependencies
- Function refactoring
- Run startup & shutdown tasks
- Create project shortcuts
- Label and filter files
- Integrate source control



Agenda

	Setting up your development environment
	Managing team workflows
	Developing better code and models
	Testing and verification

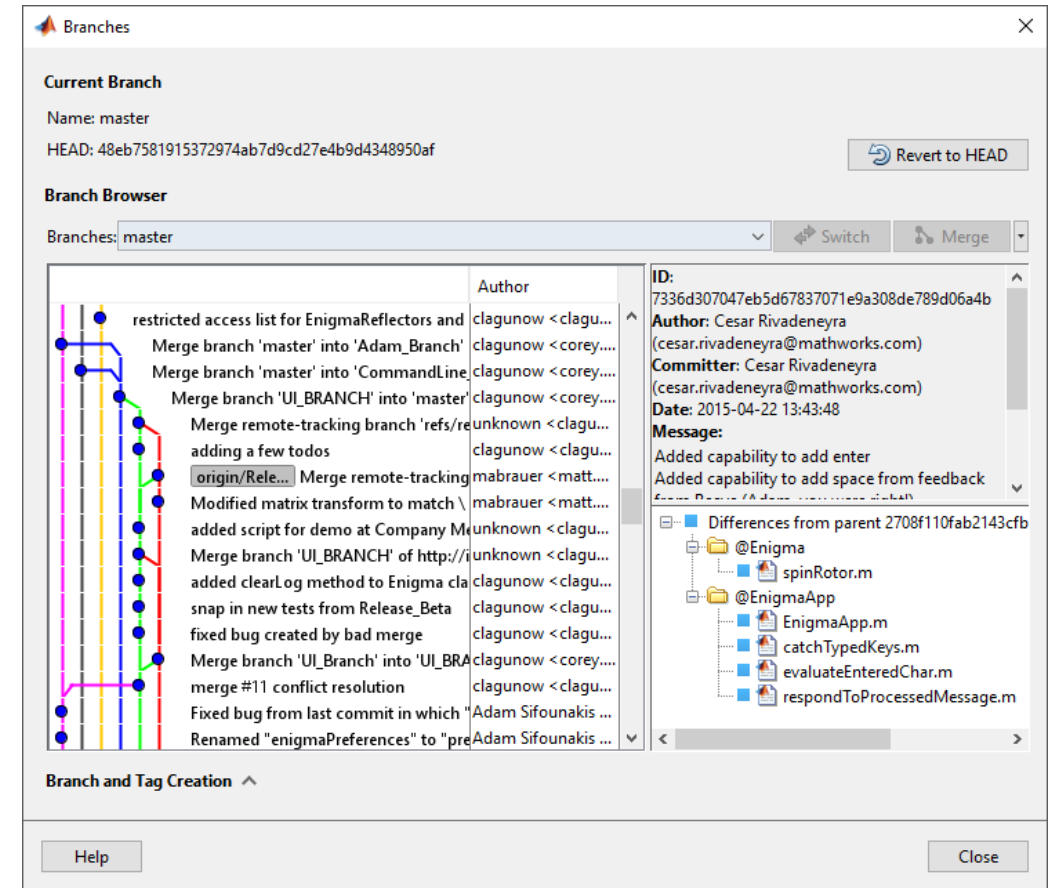
Team-Based Development Challenges

“Someone else broke my code...”

- Develop code without affecting others?
- Identify the source of development conflicts?
- Resolve development conflicts?
- ...

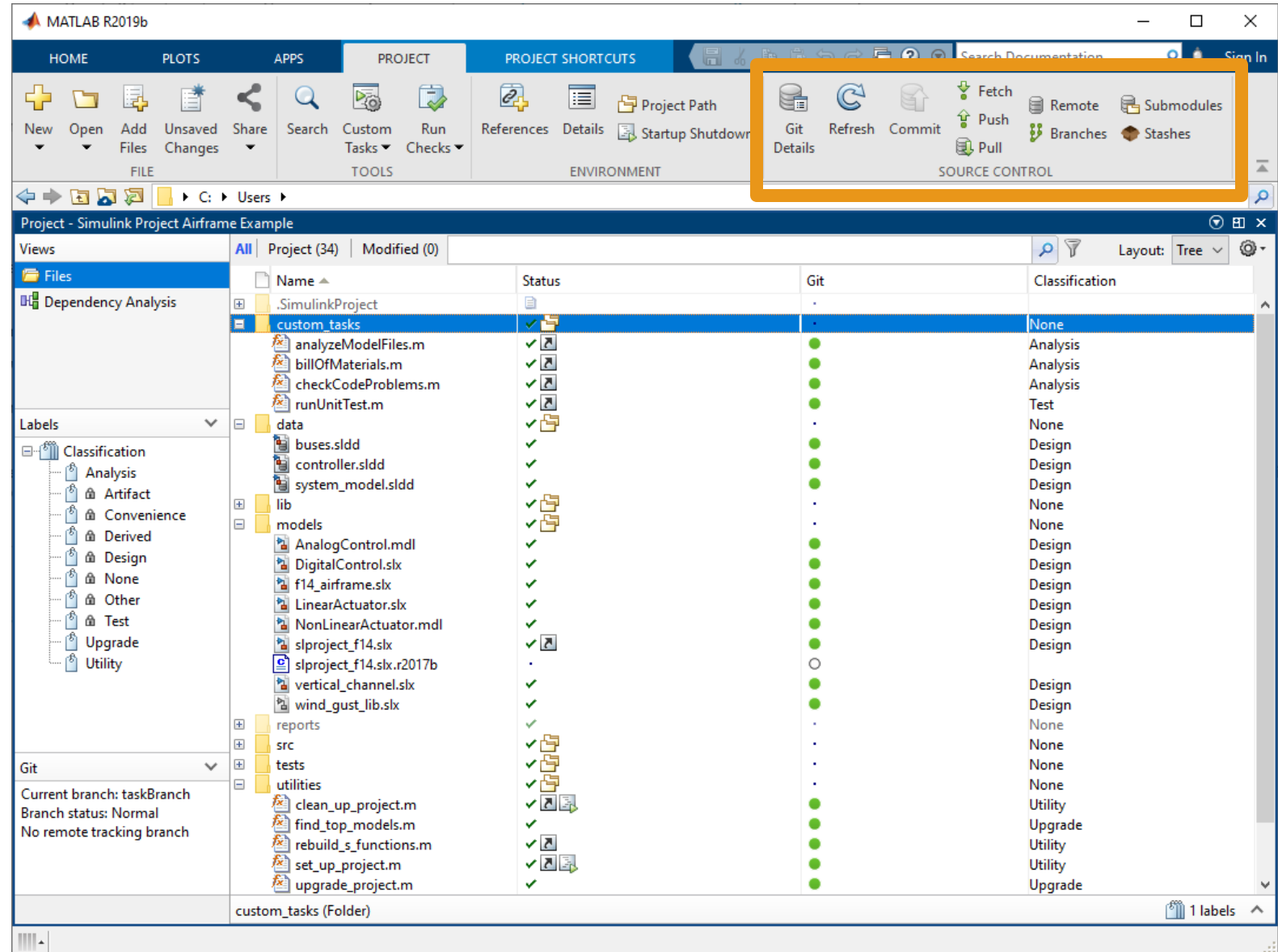
Source Control

- A system to manage changes to code, models, documents, etc.
- Benefits of source control:
 - Maintain backups, history, and ability to restore
 - Track changes and responsibility
 - Reconcile conflicting changes
 - Generate discussion
 - Save you from yourself



Source Control Integration

- Manage your code and models from within MATLAB and Simulink
- Git integrated into:
 - Projects
 - Current Folder browser
- Use Comparison Tool to view and merge changes between revisions



Comparison Tool and 3-way merge resolution

MATLAB

Comparison - Timetable_Example_Rev_521ad33e530dc8fe516d5fa2f1dad27a51bf8245.mlx vs. Timetable_Example.mlx

COMPARISON VIEW

Next Previous Refresh Find Merge Mode

Timetable_Example_Rev_521ad33e530dc8fe516d5fa2f1dad27a51bf8245.mlx vs. Timetable_Example.mlx

Timetable_Example_Rev_521ad33e530dc8fe516d5fa2f1dad27a51bf8245.mlx

Synchronizing with interpolation

synchronize combines data and sorts it, but does not interpolate or fill missing values by default. You can specify which type of interpolation to use for missing data with additional parameters.

```
8 ttLinear = synchronize(indoors,outdoors,'union','linear');
9 ttLinear(1:5,:)
```

Timetable_Example.mlx

Synchronizing with interpolation

synchronize combines data and sorts it, but does not interpolate or fill missing values by default. You can specify which type of interpolation to use for missing data with additional parameters.

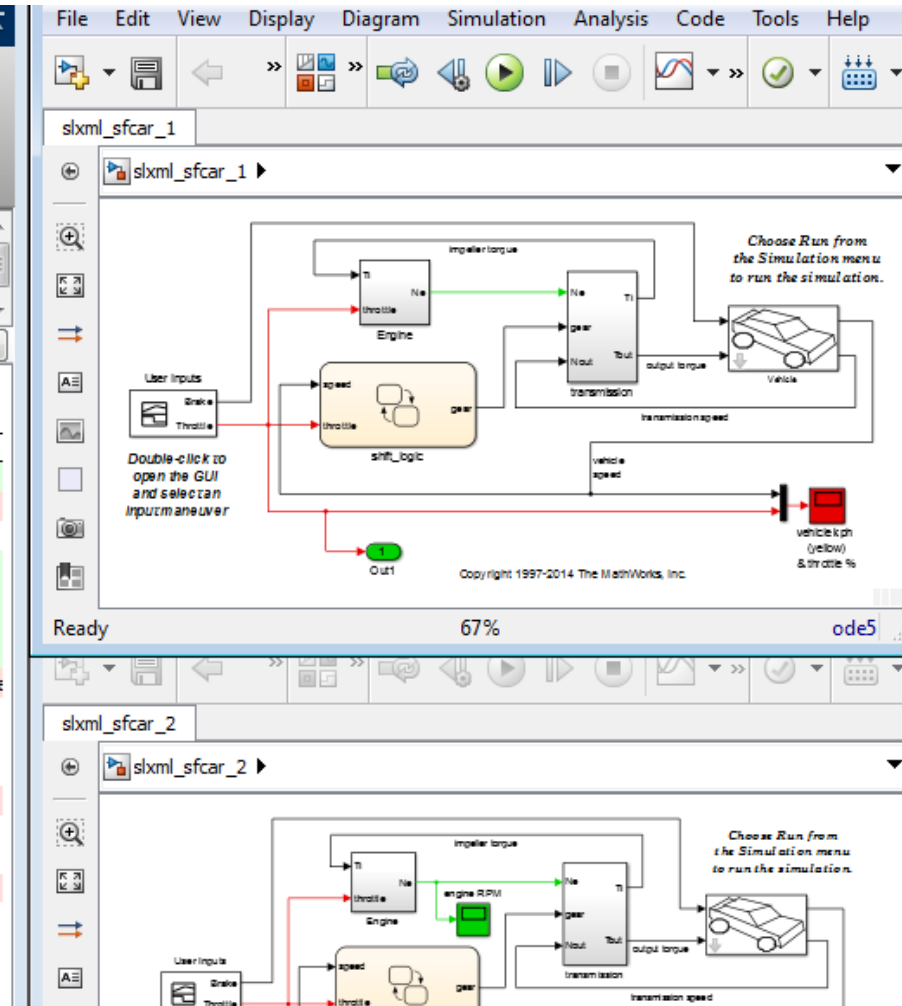
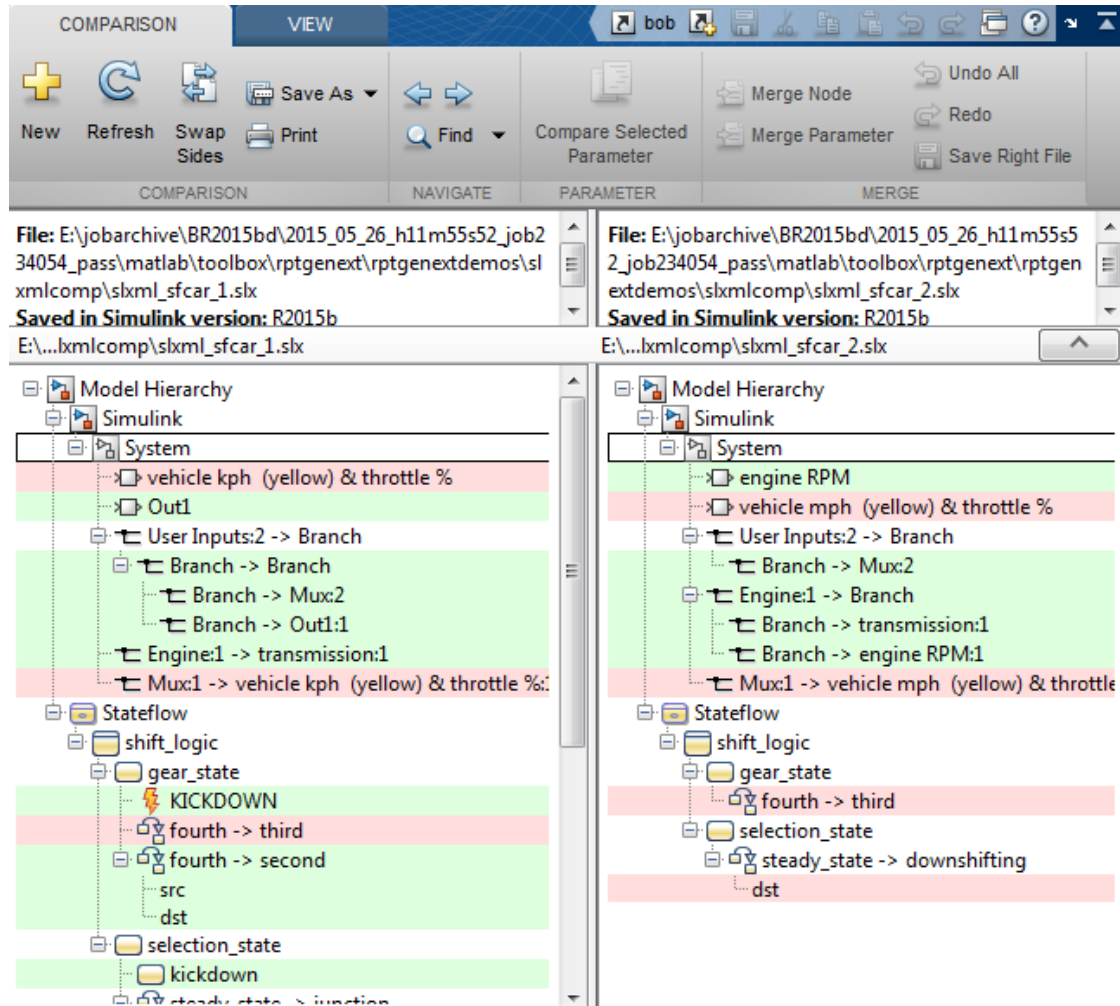
```
8 ttLinear = synchronize(indoors,outdoors,'union','spline')
9 ttLinear(1:5,:)
```

+ Insertion - Deletion ≠ Modification

5 Differences

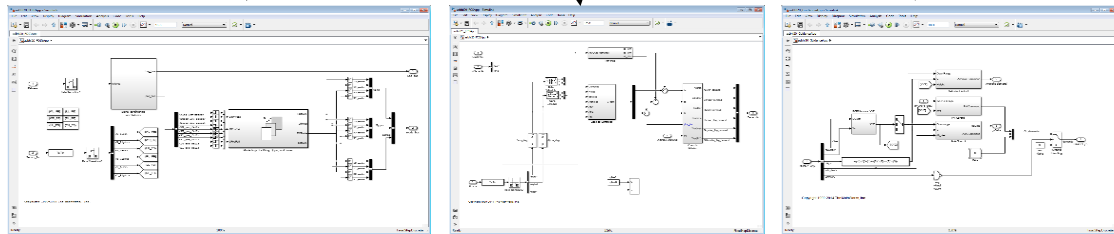
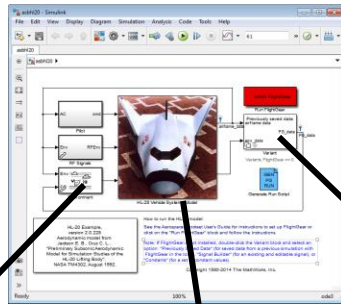
Comparison Tool and 3-way merge resolution

Simulink



Managing Complexity with Model and Project References

Model Reference



Project Reference


A screenshot of the 'Simulink Project - ConVeh' interface. The 'References' pane on the left lists several sub-projects: 'AutoTransmission', 'Auto Transmission', 'Brake System', 'Electrical', 'Climate Control', 'Radar', and 'Engine'. The 'AutoTransmission' entry is highlighted with an orange box. The right pane shows a file list for the 'Auto Transmission' project, including 'Actuator.mdl', 'Actuator.mdl.r201...', 'AnalogControl.mdl', 'Analog Control.mdl', and 'AutomaticTransmi...'. Below the file list, the 'Actuator.mdl (Simulink Model)' details are shown, including its name, checkpoint, description, and project root.

Name	Status	SVN	Revision
branches	✓	●	1
data	✓	●	7
lib	✓	●	7
models	✓	●	8
Actuator.mdl	✓	■	7
Actuator.mdl.r201...	•	•	•
AnalogControl.mdl	✓	●	8
Analog Control.mdl	✓	●	7
AutomaticTransmi...	✓	■	7

Actuator.mdl (Simulink Model)

Name: AutoTransmission
Checkpoint: August 7, 2018 3:11:50 PM
Description:
Project root: H:\Documents\MATLAB\Automotive Projects\Auto Transmissi

Agenda

	Setting up your development environment
	Managing team workflows
	Developing better code and models
	Testing and verification

What defines a “better” design?

- Faster?
- More memory efficient?
- Better organized?
- More stable?
- More portable?
- Easier to maintain?
- ...

YES!



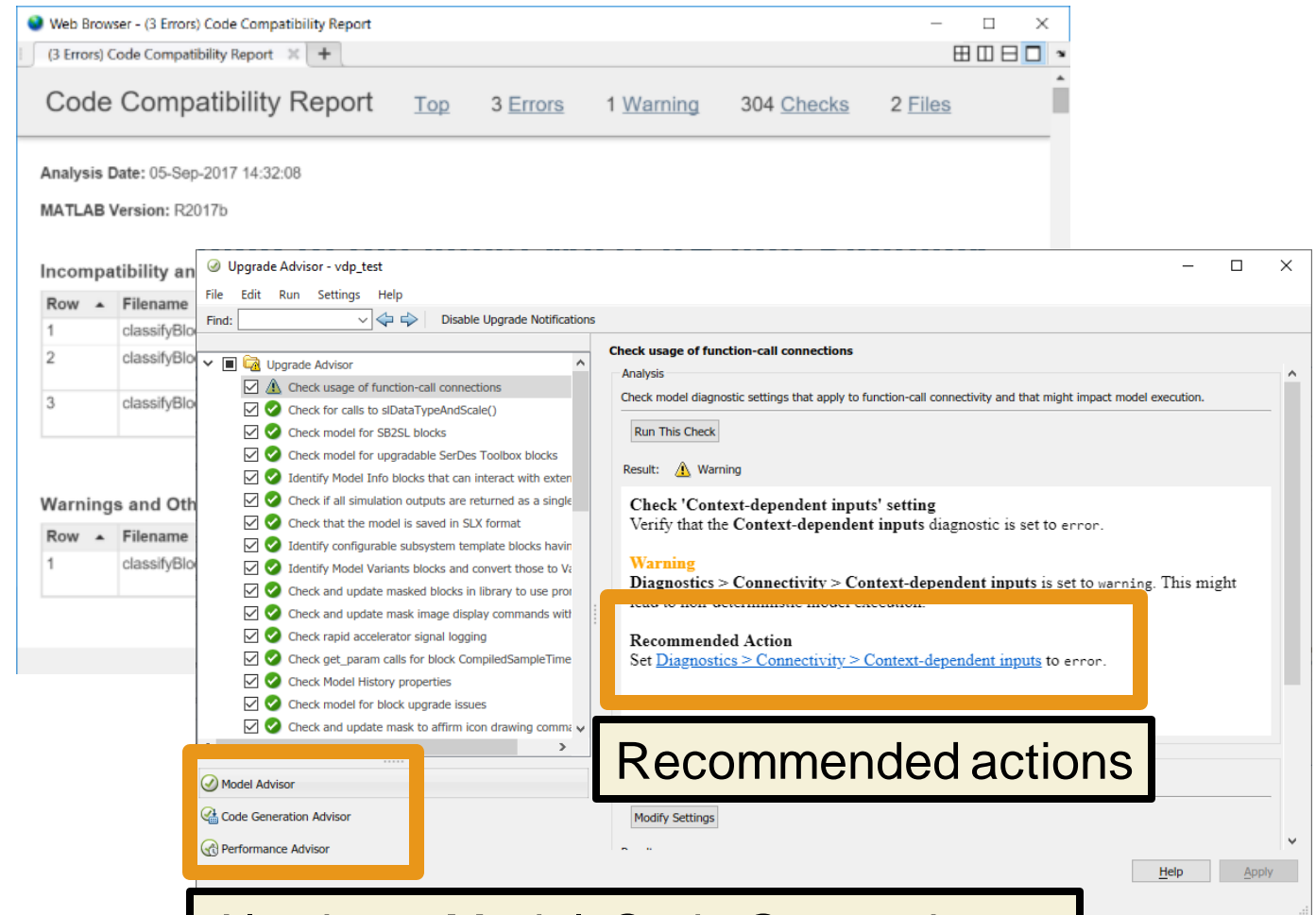
Developing robust software systems in MATLAB and Simulink

- Writing better and faster code
- Reduce complexity with refactoring
- Integrating with other languages and tools
- Sharing and reuse

Upgrading to the Latest Version of MATLAB and Simulink

- Code Compatibility Report


- Upgrade Advisor



Simplify Function Argument Validation and Error Checking

```
% Error check required input arguments
if nargin < 1
    error("rectangle requires width and height values");
elseif ~isnumeric(width) || ~isscalar(width)
    error("width must be a scalar numeric value")
elseif ~isnumeric(height) || ~isscalar(height)
    error("height must be a scalar numeric value")
end

% Process optional inputs xStart and yStart
xStart = 0;
if nargin > 2 && isnumeric(varargin{1}) && isscalar(varargin{1})
    xStart = varargin{1};
end
yStart = 0;
if nargin > 3 && isnumeric(varargin{2}) && isscalar(varargin{2})
    yStart = varargin{2};
end
```

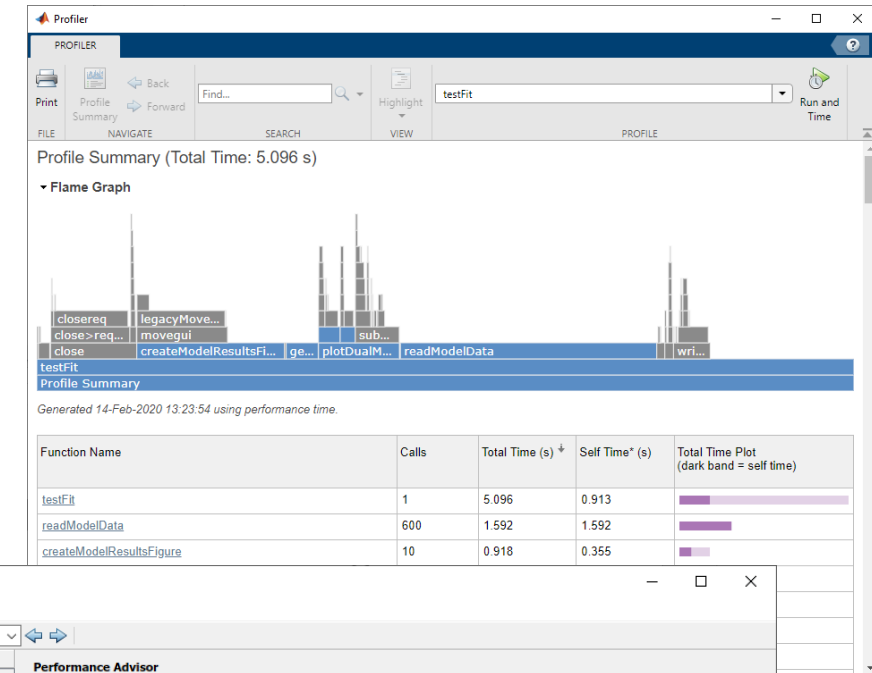


```
arguments
    width (1,1) double {mustBeNumeric}
    height (1,1) double {mustBeNumeric}
    xStart (1,1) double {mustBeNumeric} = 0; % optional
    yStart (1,1) double {mustBeNumeric} = 0; % optional
end
```

Improving Code and Model Performance

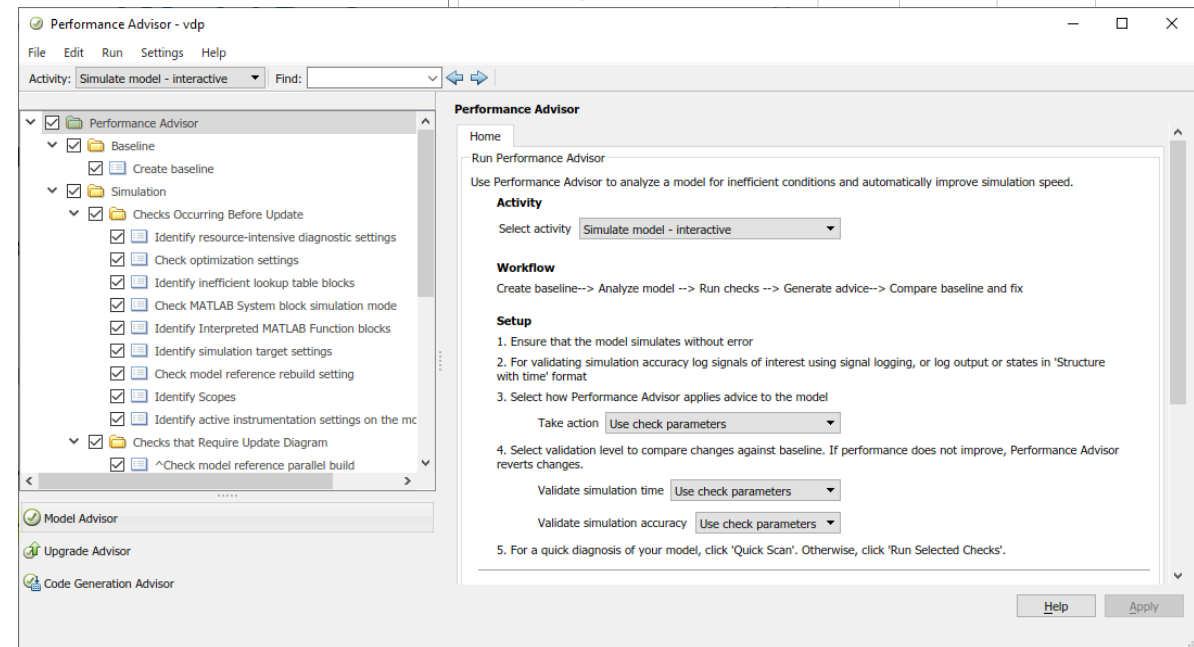
■ MATLAB Profiler

- Flame graph to highlight the largest code bottlenecks
- Total number of function calls
- Time per function call
- Statement coverage of code



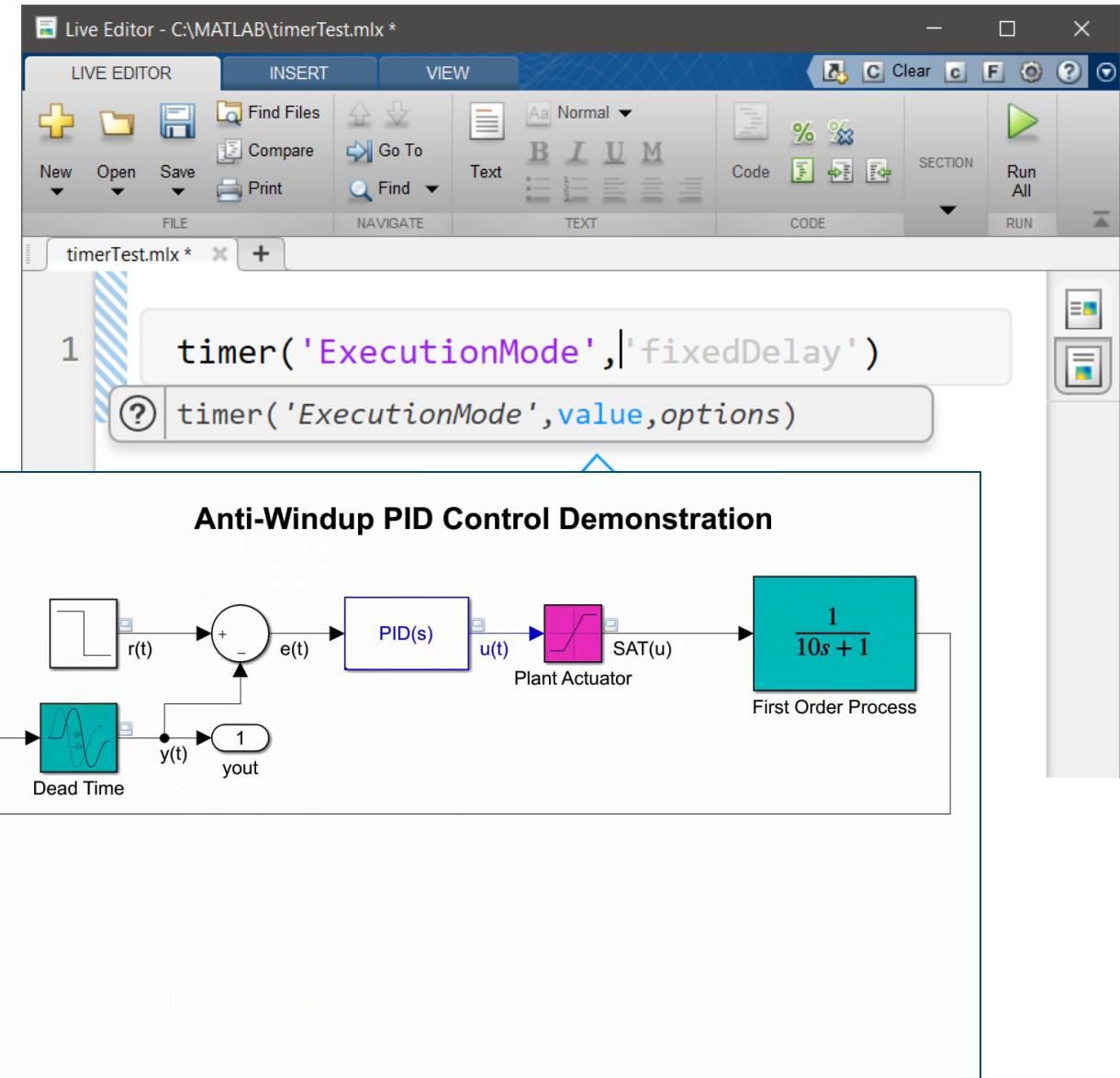
■ Performance Advisor in Simulink

- Create baselines to compare against
- Review recommendations and automatically apply changes



Speed up Your Development

- Context-aware coding guides
 - Automatically suggest functions, variables, files, and Name-Value pairs
- Model layout tools
 - Automatically clean up messy and complex models



Quickly and Safely Refactoring – MATLAB Code

- Break down large, complex codes and models into reusable and easier to maintain components

The image illustrates the process of refactoring MATLAB code. On the left, a script in the Live Editor contains the following code:

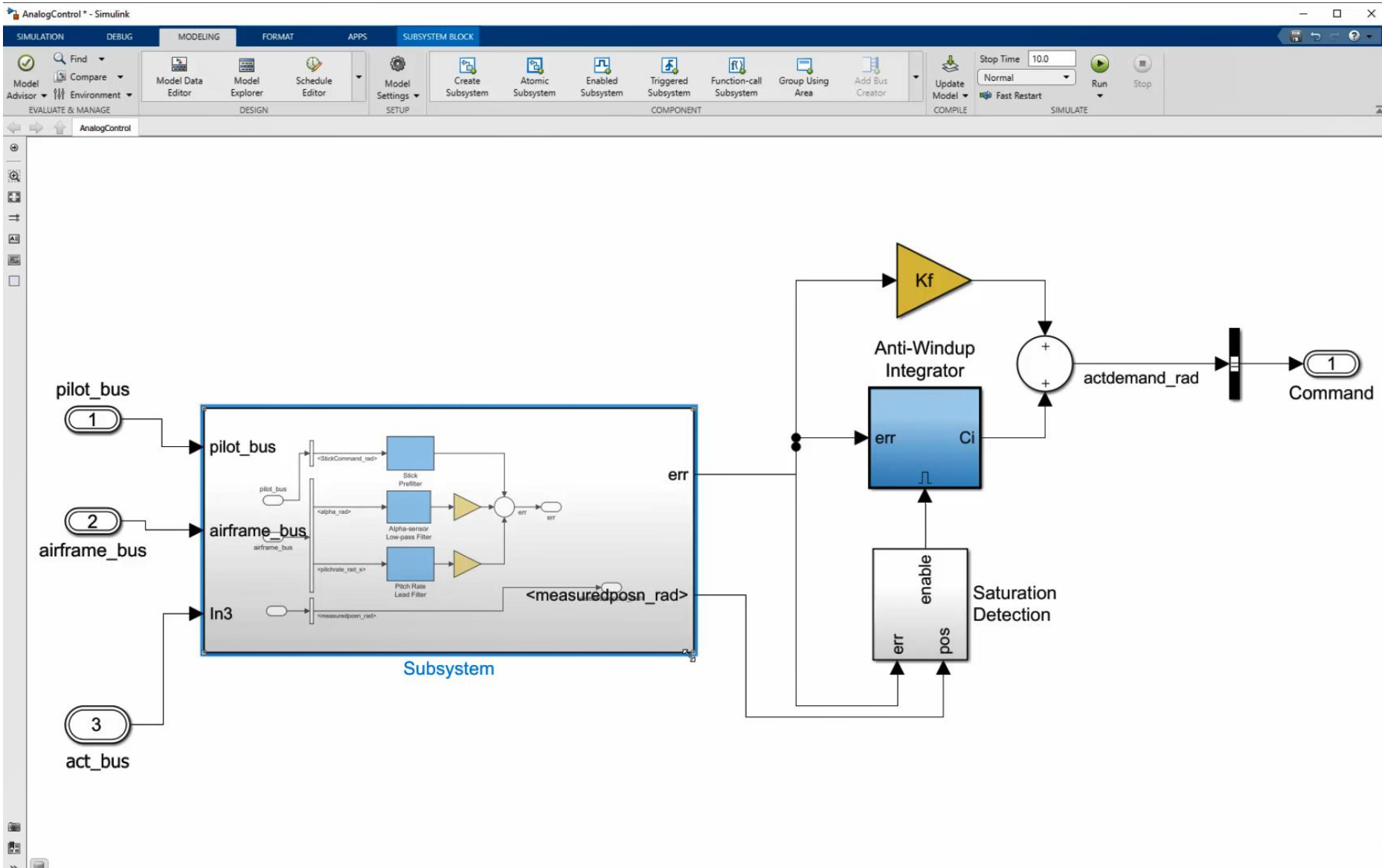
```
3 z1 = x+y;  
4 z2 = x-y;  
5 z3 = y-x;  
6 z4 = x*y;  
7 zSum = z1 + z2 + z3 + z4;  
  
8 disp(z3)  
9 disp(zSum)
```

The code is divided into two sections: "Calculate my answer:" (lines 3-7) and "Display answers of interest:" (lines 8-9). A context menu is open over the first section, with the "Convert to Function" option highlighted. An orange arrow points from this option to a second window on the right, which shows the resulting function file:

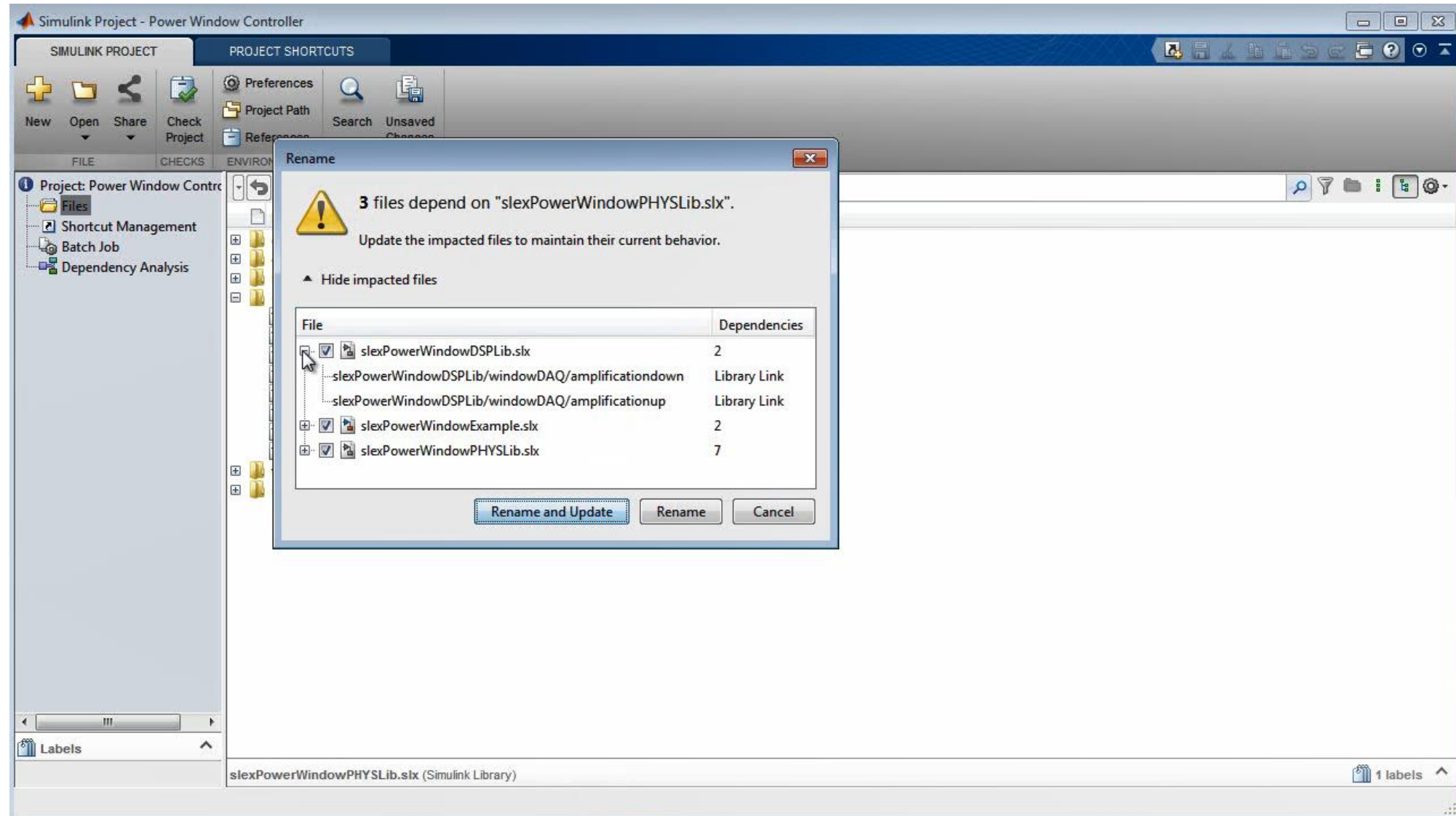
```
1 function [z3, zSum] = myMathFunction(x, y)  
2     z1 = x+y;  
3     z2 = x-y;  
4     z3 = y-x;  
5     z4 = x*y;  
6     zSum = z1 + z2 + z3 + z4;  
7 end
```

In the function file, the function signature `[z3, zSum] = myMathFunction(x, y)` and the variable `z3` in the assignment `z3 = y-x;` are highlighted with orange boxes.

Quickly and Safely Refactoring – Simulink Models

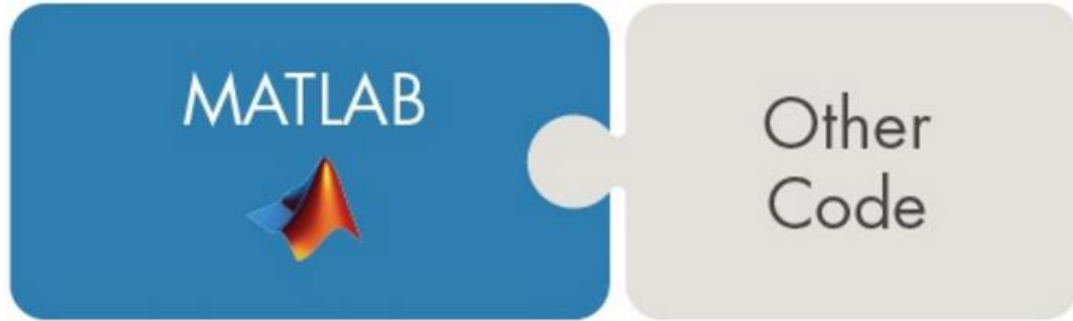


Quickly and Safely Refactoring – Function and Model Names



Integrating with other languages

Calling Libraries Written in Another Language



- Java
- Python
- C/C++
- Fortran
- COM components and ActiveX® controls
- RESTful, HTTP, and WSDL web services

Calling MATLAB from Another Language



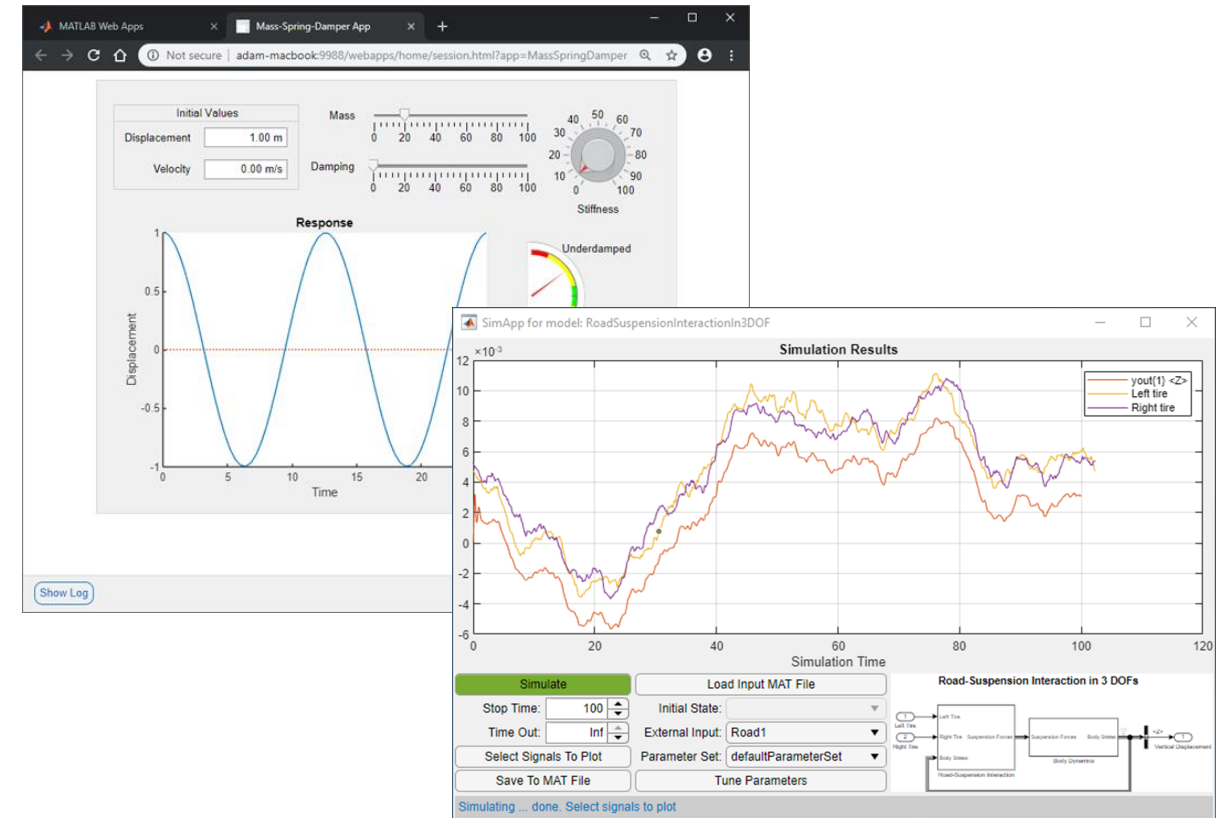
- Java
- Python
- C/C++
- Fortran
- COM Automation server

Integrating with other languages



Sharing your work

- Co-authors and development teams
 - Projects
- End-user with MATLAB and Simulink
 - Toolbox or App
- End-user without MATLAB and Simulink
 - Standalone and web applications
 - Language-specific libraries
 - Generated standalone code
 - Microservice APIs



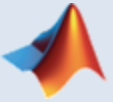
MATLAB Compiler, Simulink Compiler

MATLAB Compiler SDK

Embedded Coder, HDL Coder, PLC Coder, GPU Coder, ...

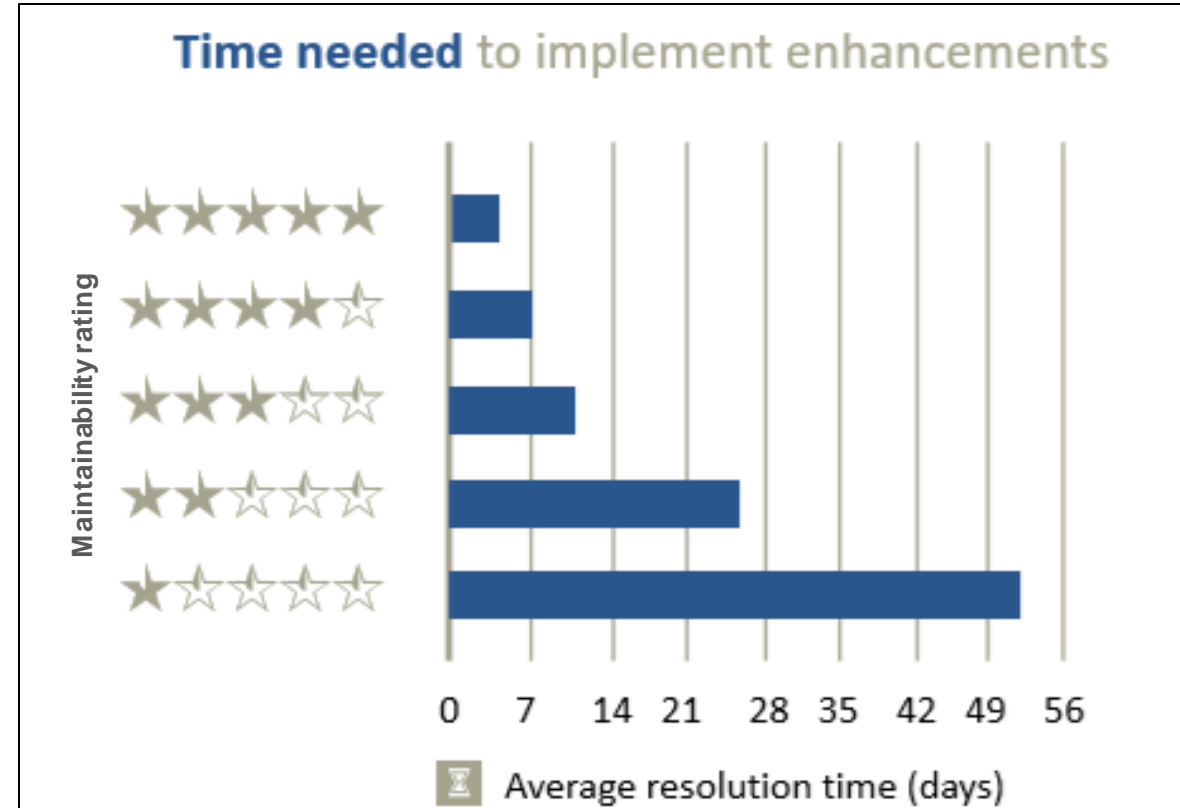
MATLAB Production Server

Agenda

	Setting up your development environment
	Managing team workflows
	Developing better code and models
	Testing and verification

Software Maintenance – The hidden cost of development

- How do you ensure code and models don't break over time?
- How do you keep new features from breaking existing features?
- How do you maintain confidence that your system is working as expected?
- How do you ensure that your software is future-proof?



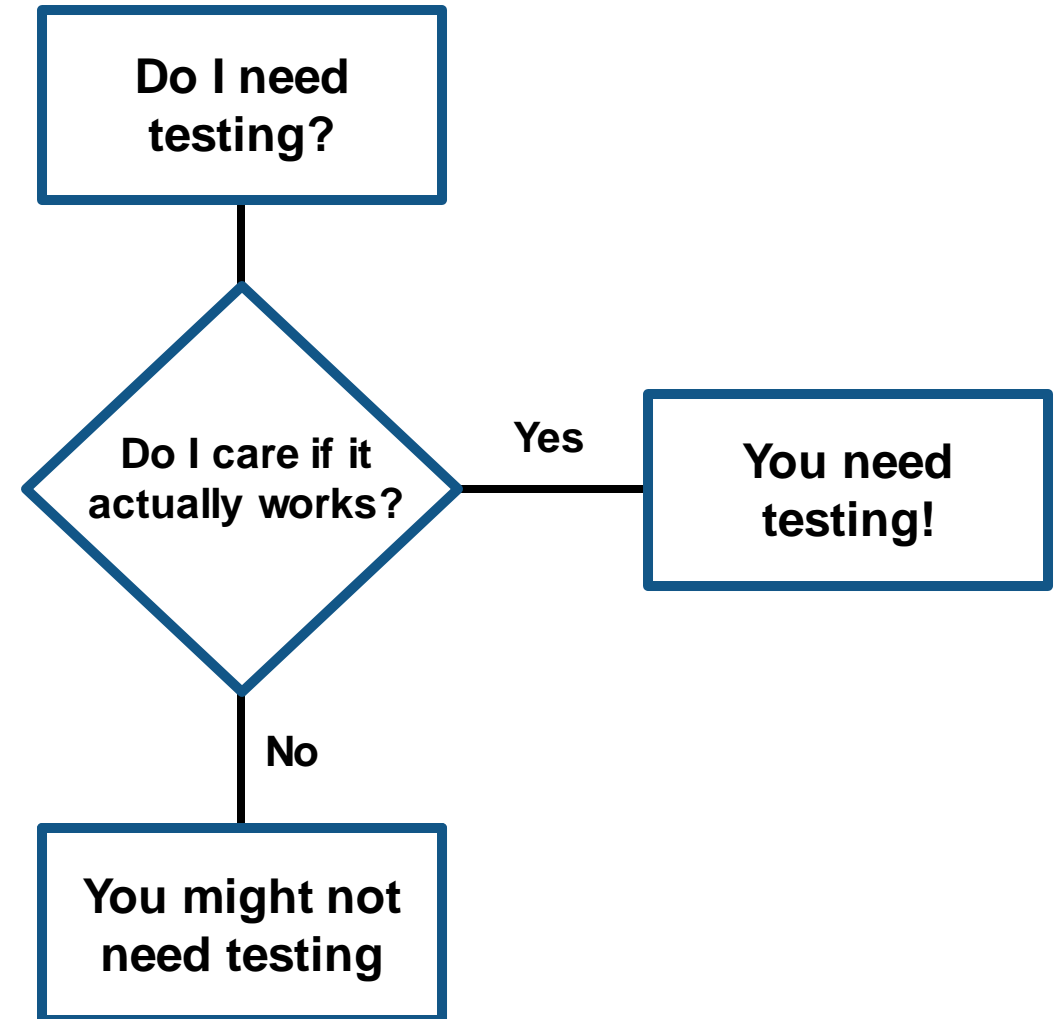
Journal paper: “Faster issue resolution with higher technical quality of software”, Software Quality Journal, 2011

Test early, test often, test automatically

- Reduce risk of software breaking
- Catch problems early
- Improve quality
- Document expected behaviour

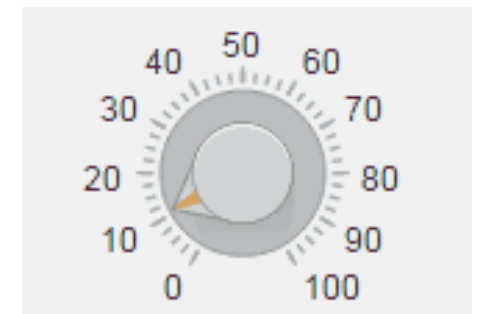
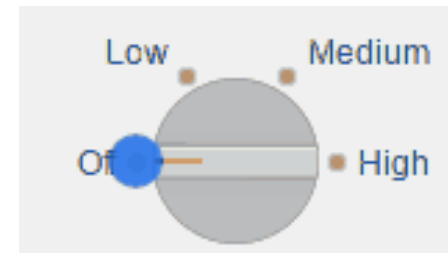
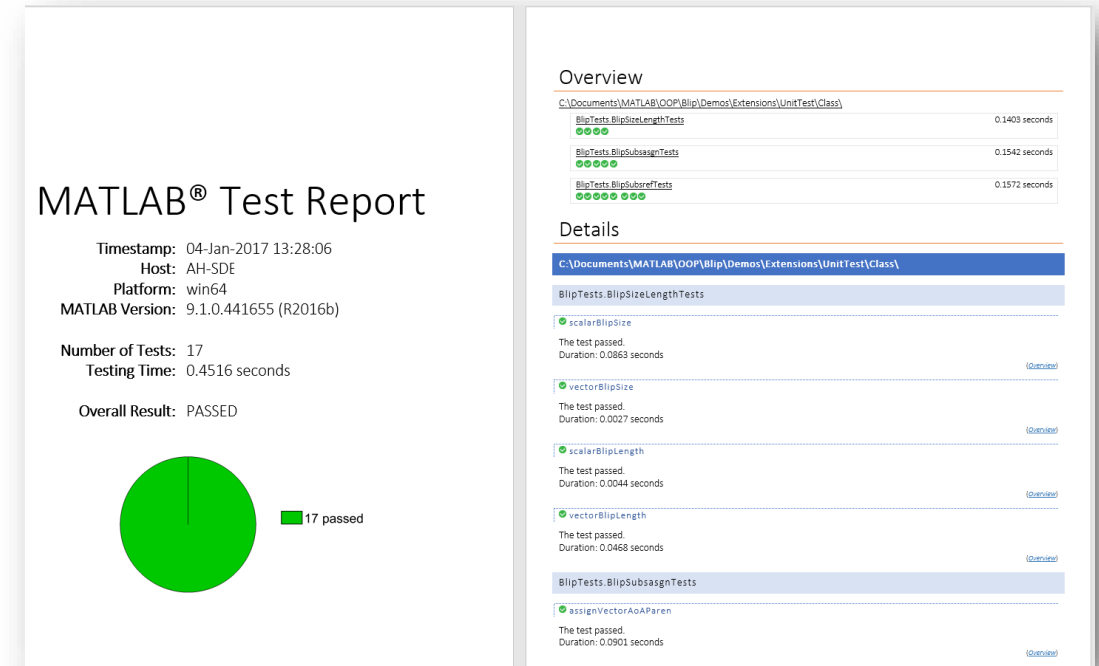


Credit: <http://geek-and-poke.com/>

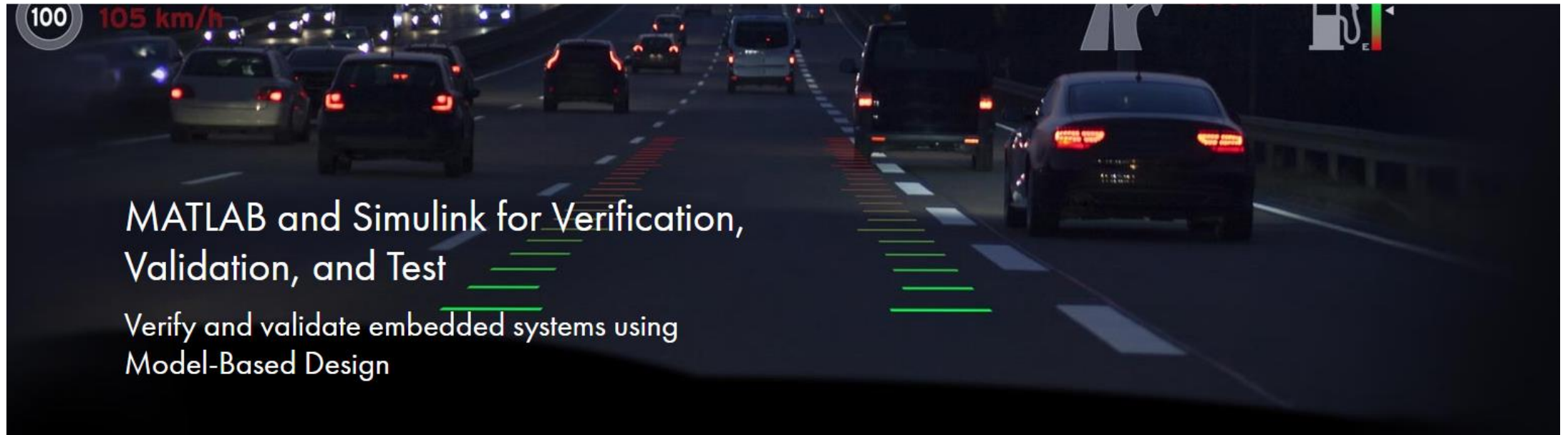


MATLAB Testing Frameworks

- MATLAB Unit Testing Framework
- Performance Testing Framework
- Mocking Framework
- App Testing Framework



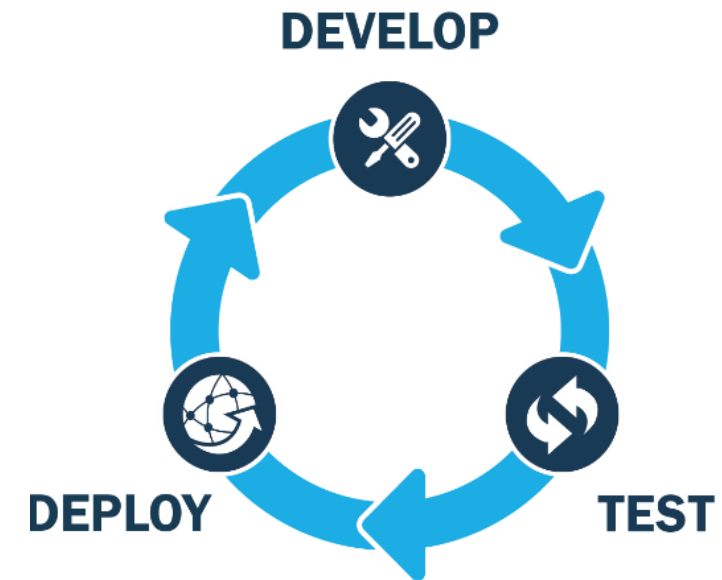
Verification and Validation in Simulink



- **Trace requirements** to architecture, design, tests, and code
- **Verify your design** meets requirements and is free of critical run-time errors
- **Check compliance** and measure quality of models and code
- **Generate test cases** automatically to increase test coverage
- **Produce reports** and artifacts, and certify to standards (such as [DO-178](#) and [ISO 26262](#)).

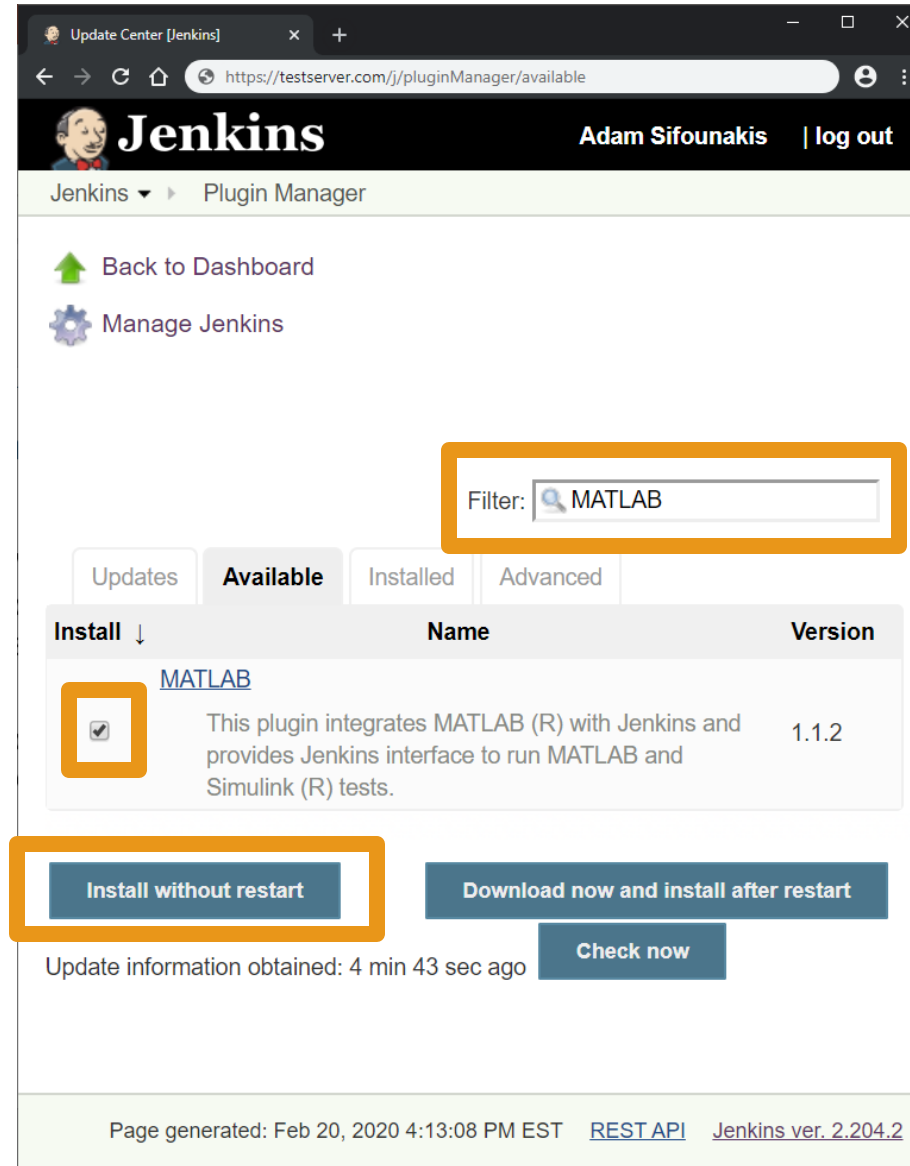
Automated Testing with Continuous Integration (CI)

- A system to automate the building, testing, integration, and deployment of code as it is being developed and maintained
- Popular CI systems: Jenkins, Travis, CircleCI , Azure DevOps, and others...
- Benefits:
 - Detect integration bugs early
 - Allow you to stop bugs from being accepted
 - Track and report testing history
 - Flexible testing schedules and triggers



MATLAB Plugin for Jenkins

- Install MATLAB Plugin for Jenkins directly from the Jenkins Plugin Manager
- Easily connect and configure MATLAB with Jenkins
- Schedule automatic code and model testing
 - MATLAB Unit Test Framework
 - Simulink Test



The screenshot shows the Jenkins Plugin Manager interface. The browser address bar indicates the URL `https://testserver.com/j/pluginManager/available`. The Jenkins logo and user name "Adam Sifounakis" are visible at the top. The "Plugin Manager" section is active, showing a "Filter" box with "MATLAB" entered. Below the filter, the "Available" tab is selected, displaying a table of plugins. The "MATLAB" plugin is listed with version 1.1.2 and a description: "This plugin integrates MATLAB (R) with Jenkins and provides Jenkins interface to run MATLAB and Simulink (R) tests." The "Install without restart" button is highlighted with an orange box. Other buttons include "Back to Dashboard", "Manage Jenkins", "Updates", "Installed", "Advanced", "Download now and install after restart", and "Check now". The footer shows the page generation time and version information.

Update Center [Jenkins] x +
https://testserver.com/j/pluginManager/available

Jenkins Adam Sifounakis | log out

Jenkins ▾ Plugin Manager

Back to Dashboard
Manage Jenkins

Filter:

Updates Available Installed Advanced

Install ↓	Name	Version
<input checked="" type="checkbox"/>	MATLAB This plugin integrates MATLAB (R) with Jenkins and provides Jenkins interface to run MATLAB and Simulink (R) tests.	1.1.2

Update information obtained: 4 min 43 sec ago

Page generated: Feb 20, 2020 4:13:08 PM EST [REST API](#) [Jenkins ver. 2.204.2](#)



Testing Reports in Jenkins



- View testing results
- View code coverage
- View testing reports

Test Result

1 failures (+1)

3 tests (±0)
Took 1.6 sec.
[add description](#)

All Failed Tests

Test Name
[designTest.testOvershoot](#)

Stack Trace

```
=====
Verification failed in designTest/te

-----
Framework Diagnostic:

verifyLessThan failed.
--> The value must be less than

-----
Actual Value:
0.082943282378937
Maximum Value (Exclusive):
0.0100000000000000
-----
Stack Information:

In /Users/Shared/Jenkins/Home/jc
Damper/workspace/tests/designTest.m
=====
```

File Coverage summary

Name	Classes	Lines	Conditionals
simulateSystem.m	100% 1/1	90% 9/10	100% 0/0

Coverage Breakdown by Class

[simulateSystem](#) 90%

Source

```
simulateSystem.m
1 function [x, t] = simulateSystem
2
3 2 springMassDamperDesign; % Create
4
5 2 if ~isstruct(design) || ~all(isfi
6 0 error('simulateSystem:Invalid
7     'The design should be a s
8 end
9
10 % Design variables
11 2 c = design.c;
12 2 k = design.k;
13
14 % Constant variables
15 2 z0 = [-0.1; 0]; % Initial Positi
16 2 m = 1500; % Mass
17
18 2 odefun = @(t,z) [0 1; -k/m -c/m]*
19 2 [t, z] = ode45(odefun, [0, 3], z0
20
21 % The first column is the positio
22 2 x = z(:, 1);
```

Project Mass-Spring-Damper

[add description](#)
[Disable Project](#)

Coverage Report
 Workspace
 Recent Changes

Permalinks

- Last build (#89). 1 mo 24 days ago
- Last stable build (#88). 1 mo 29 days ago
- Last successful build (#88). 1 mo 29 days ago
- Last failed build (#89). 1 mo 24 days ago
- Last unstable build (#52). 2 mo 7 days ago
- Last unsuccessful build (#89). 1 mo 24 days ago
- Last completed build (#89). 1 mo 24 days ago

TAP Tests

Code Coverage

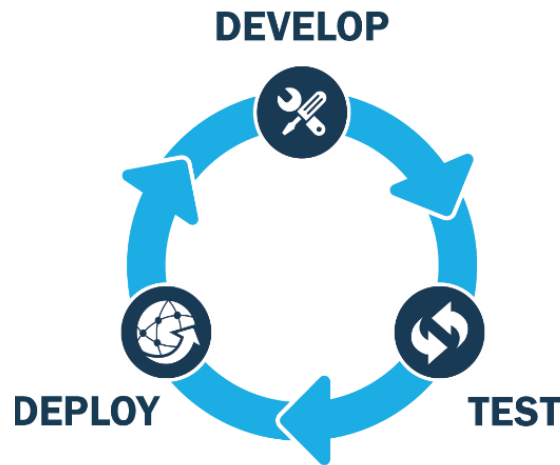
Packages 100% Files 100% Classes 100% Lines 93% Conditionals

Test Result Trend

Summary

Summary

- MATLAB and Simulink can take you all the way from idea to production
- Save time and effort with good software and modeling practices
- Projects bulletproof your collaborative development workflows



MATLAB and Simulink

are the **easiest** and
most **productive** environments
for **engineers** and **scientists**

