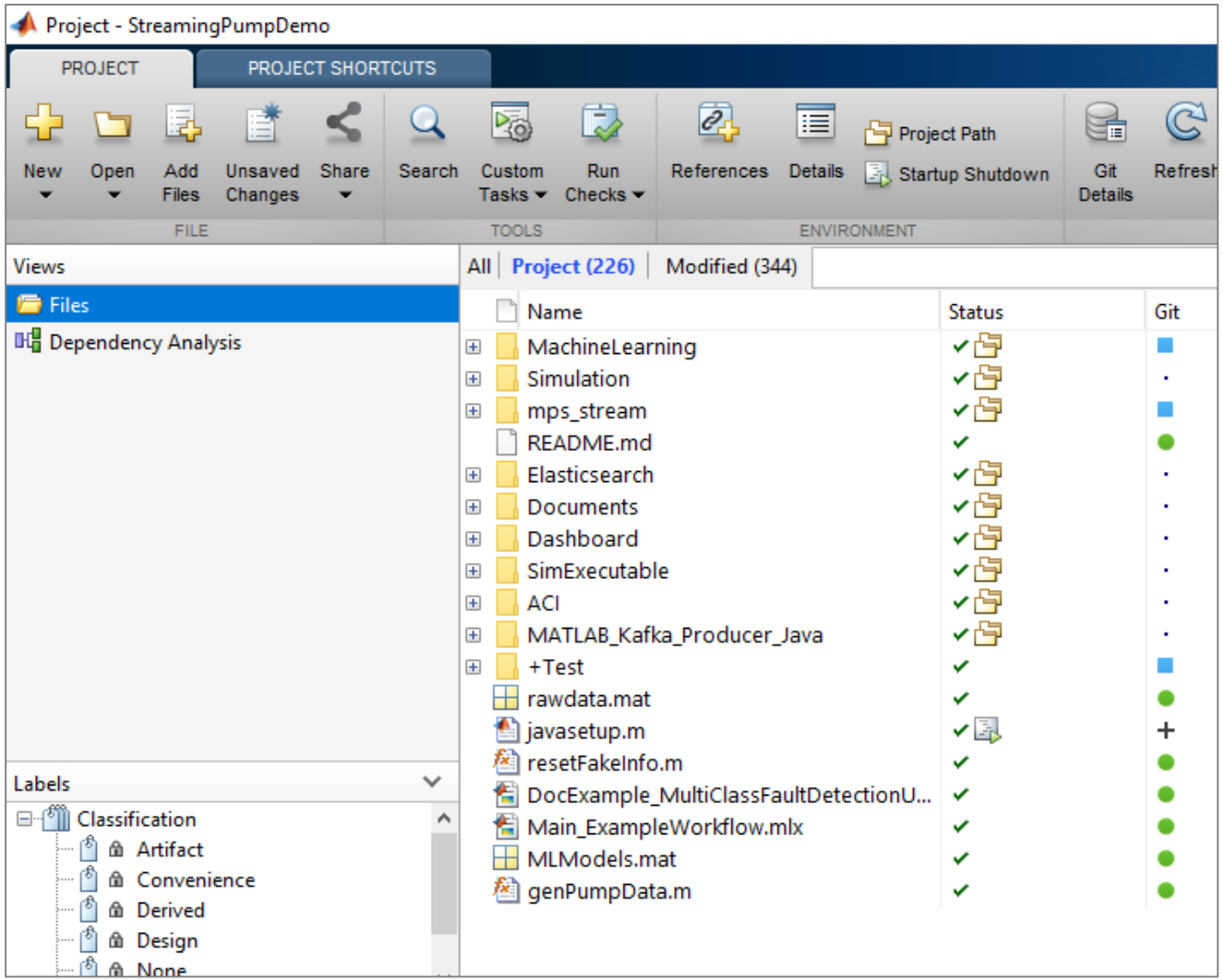
MATLAB EXPO 2019

Software Development Practices within MATLAB

Emmanuel Blanchard

How do we manage all that at MathWorks?

- Thousands of new features every year
- Over 2000 developers
- Different time zones, …



What are your software development concerns?

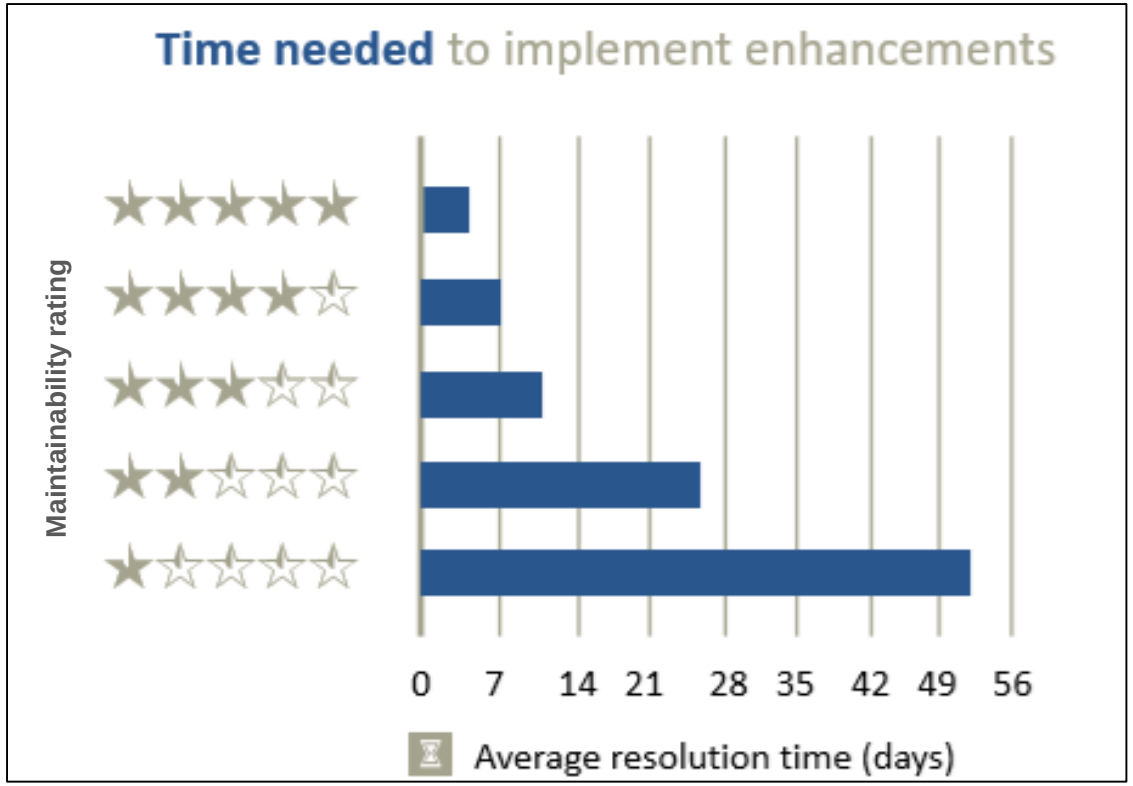
- Speed
- Development Time
- Cost
- Compatibility
- Documentation
- Reusability
- Effective Testing
- Integration
- Badly written code
- Ease of Collaboration
- Legacy Code
- Liability
- Maintainability
- Model Risk
- Robustness
- Developer Expertise
- Software Stack Complexity
- …?

Software development practices can help

Treat your software like an asset → reuse it

Developers often spend 4X the effort to maintain vs build software

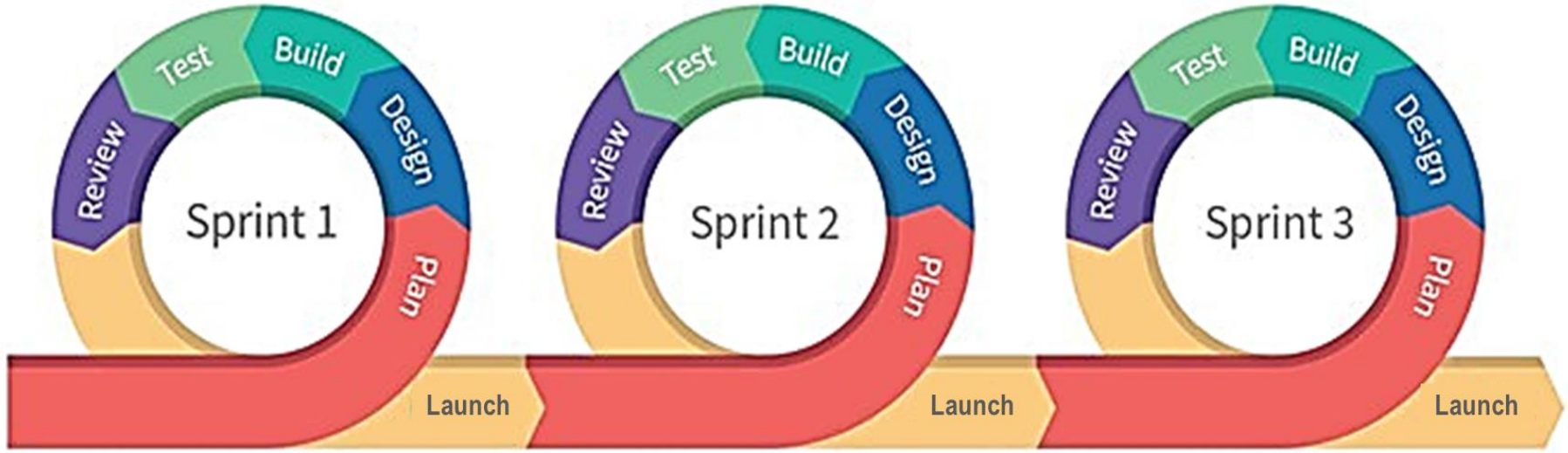
…but this doesn't need to be true!



Journal paper: "*Faster issue resolution with higher technical quality of software*", Software Quality Journal, 201100

Software development practices can help

- Software development approaches like Agile help improve code quality
- The tools and practices we discuss today support Agile development



Agenda

	Managing your code
	Tracking code changes and co-authoring workflows
	Writing better, robust, and portable code
	Testing and maintaining your code
	Summary

How do you currently manage your files and paths?

- One big folder of files?
- Many folders of files?
- Organize your code in packages?
- Manual path management?

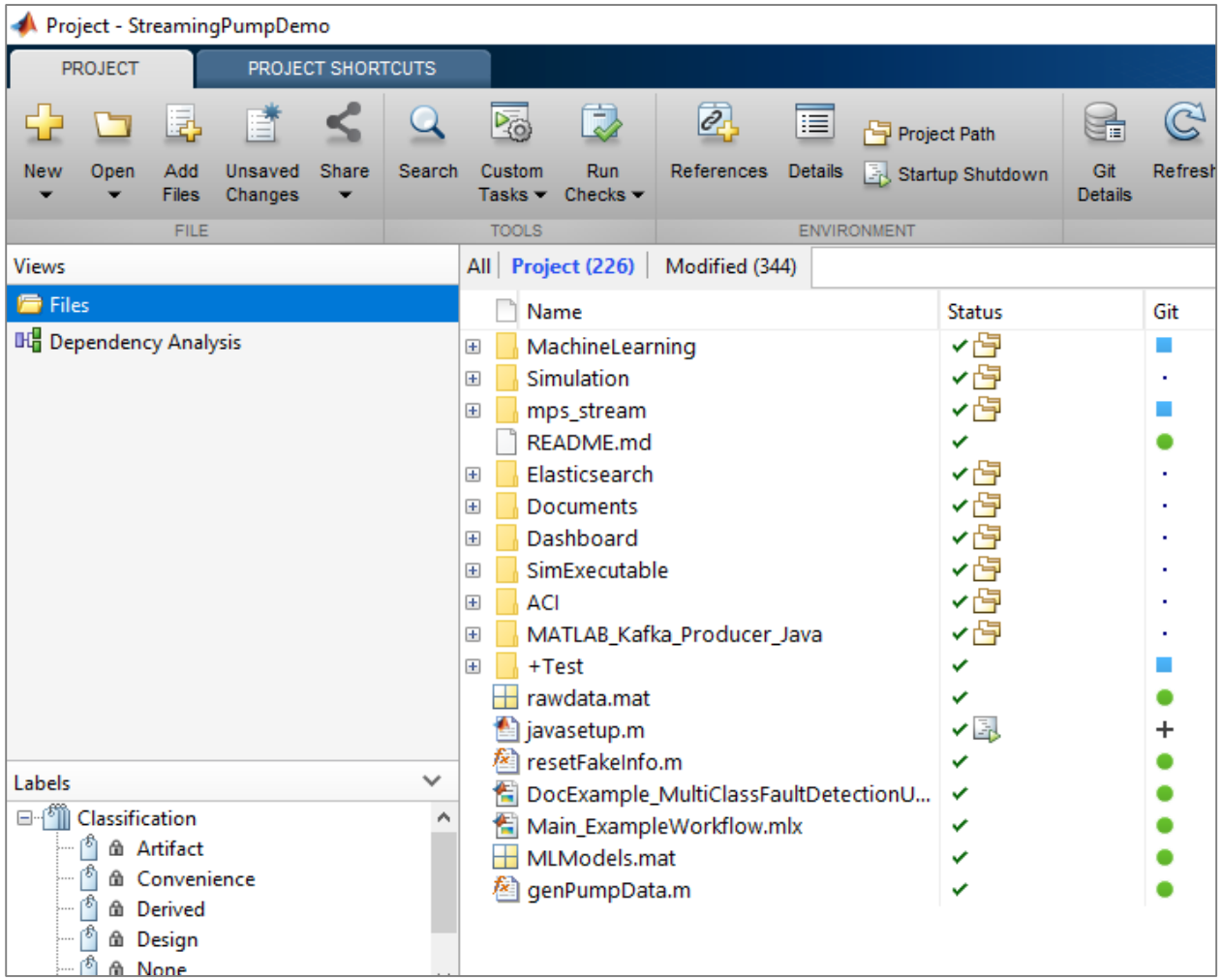
Successful collaborative development requires …

- Same source code, tests, doc, requirements, …
- Consistent, shared environment
- Integration with source control

MATLAB Path | Workspace Data | C Compiler | Apps & Toolboxes | . . .

Projects (MATLAB + Simulink Projects)

- 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

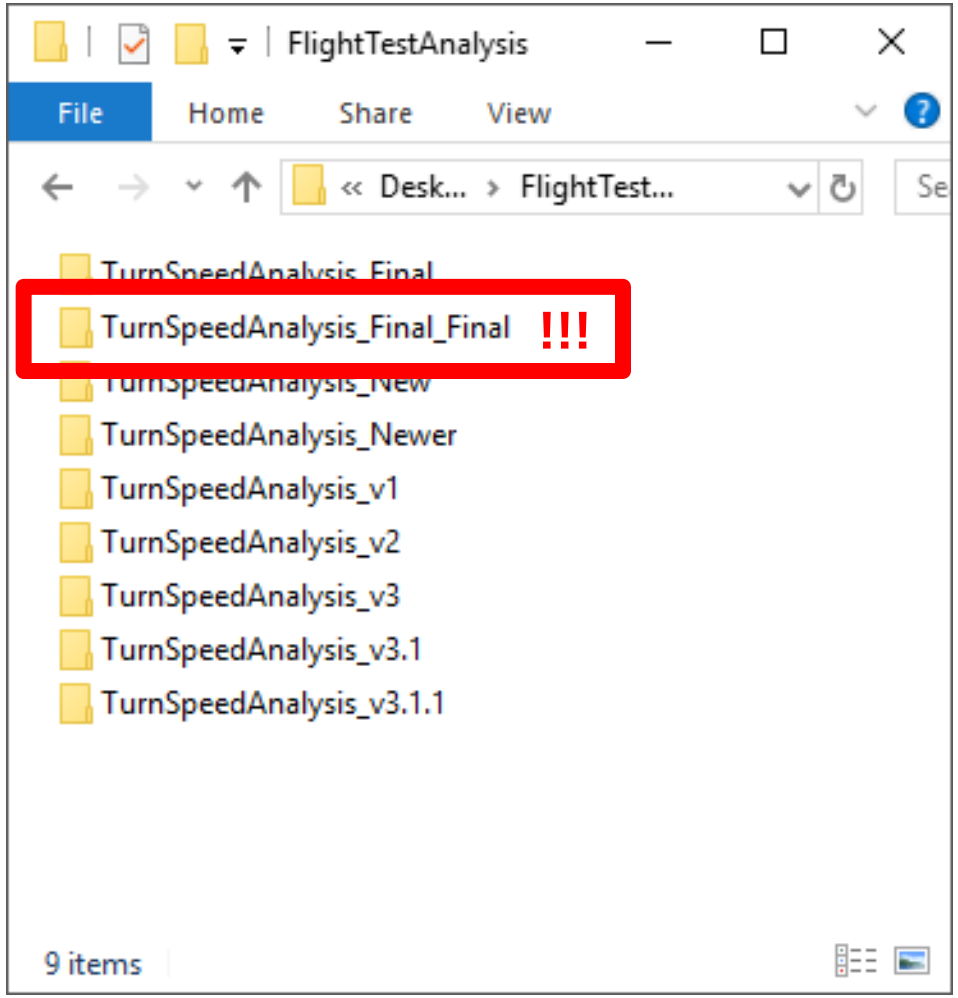
	Managing your code
	Tracking code changes and co-authoring workflows
	Writing better, robust, and portable code
	Testing and maintaining your code
	Summary

How do you keep track of and share your code as it changes?

- Do you:
  - make copies of your code?
  - e-mail yourself copies of your code?
  - keep a spreadsheet of changes?

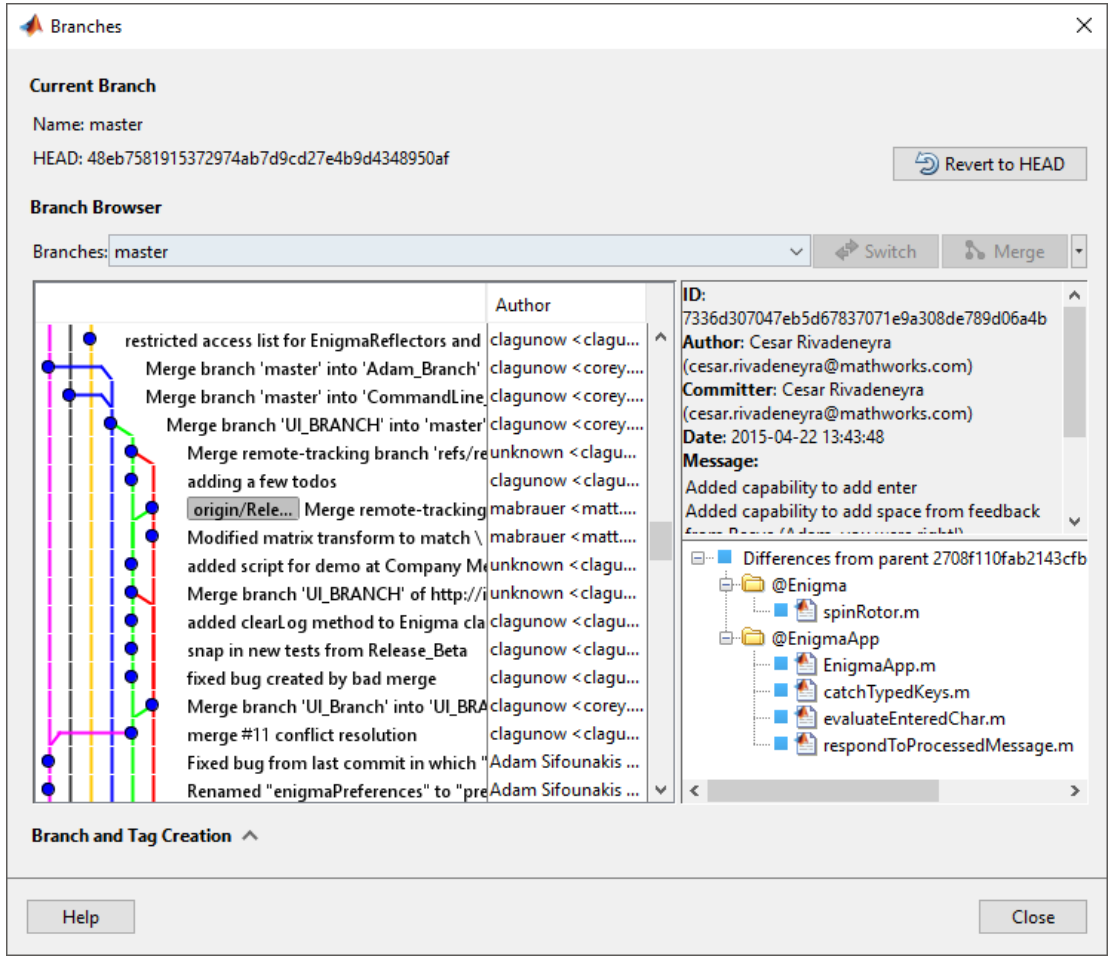
- Or do you not keep track of your changes?

There's a better way!



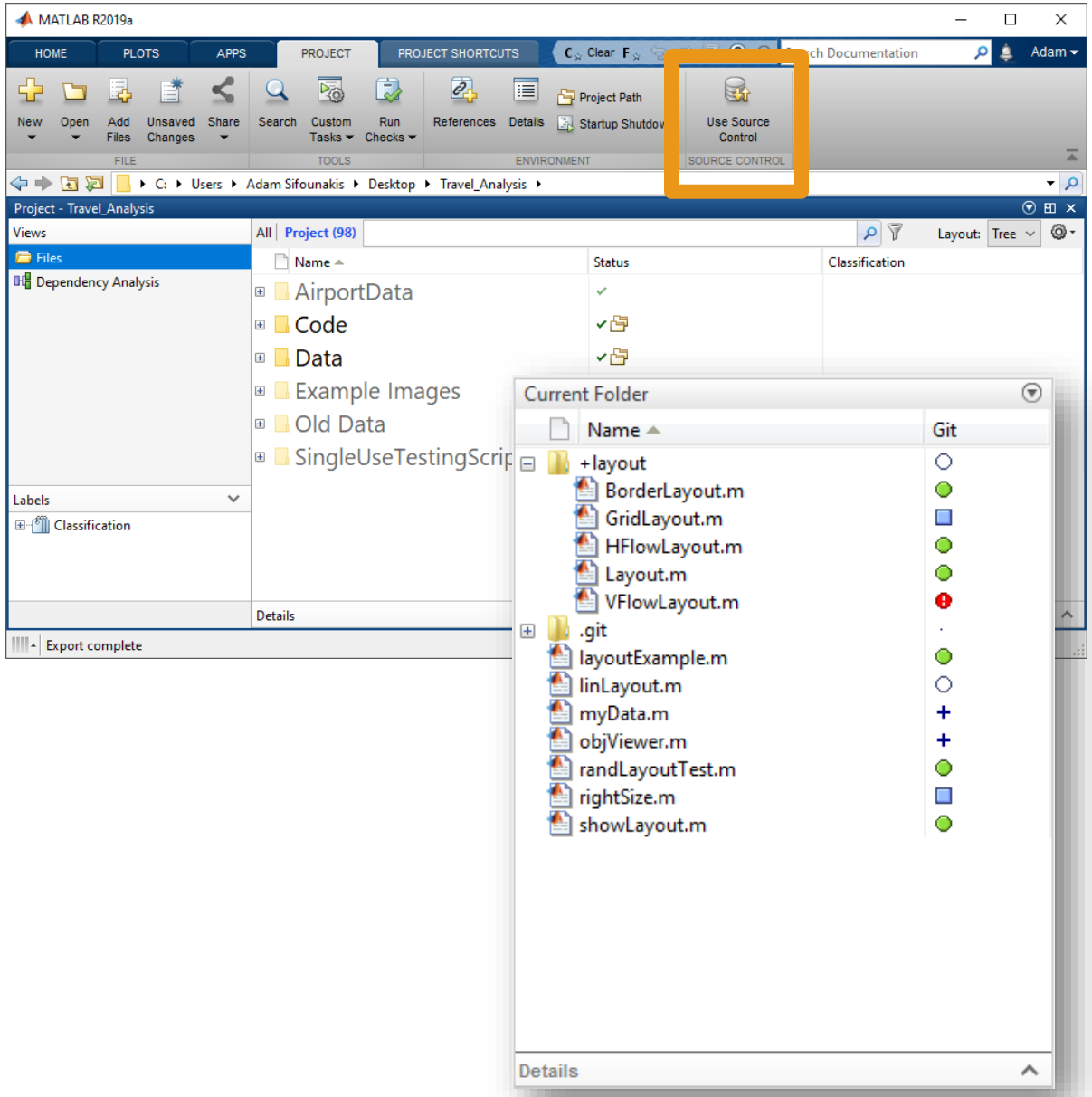
Source Control

- A system to manage changes to code, documents, etc.
- Maintain backups, history & ability to restore
- Track changes and responsibility
- Simplify reconciling conflicting changes



Source Control integration

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



Co-authoring workflows

Creating a repo:

- Initialize
- Add
- Clone

Making changes:

- Commit
- Push
- Branch
- Merge

Repo

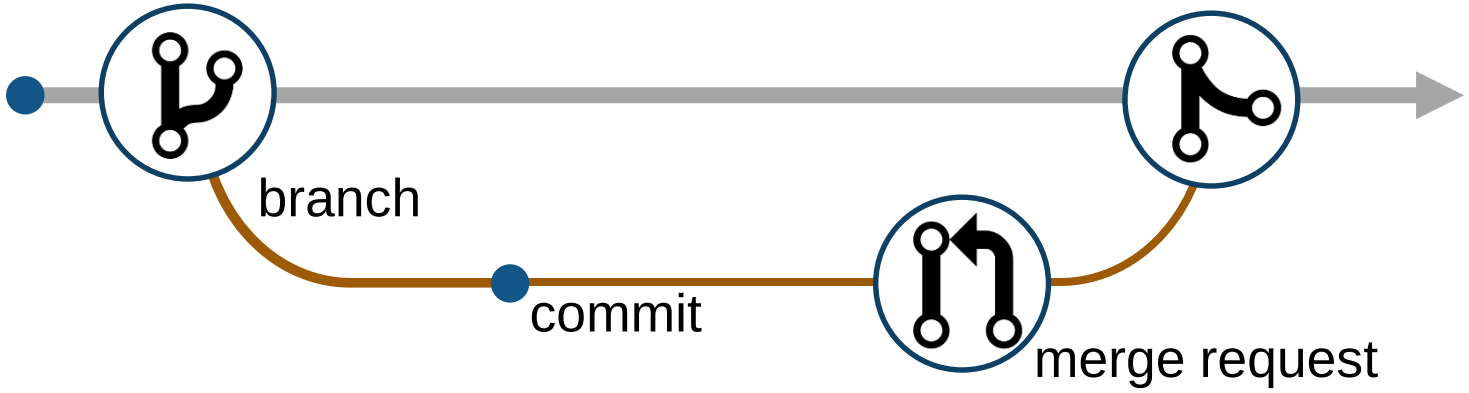
```
% Make prediction
RUL = predictRUL(mdl,data,n);
```

predictUpdateRUL.m

Repo

```
% Make prediction only when slope changes
if ~isempty(mdl.SlopeDetectionInstant)
    RUL = predictRUL(mdl,data,n);
else
    RUL = old_state.RUL;
end
```

predictUpdateRUL.m



Repo

```
% Make prediction only when slope changes
if ~isempty(mdl.SlopeDetectionInstant)
    RUL = predictRUL(mdl,data,n);
else
    RUL = old_state.RUL;
end
```

predictUpdateRUL.m

Agenda

	Managing your code
	Tracking code changes and co-authoring workflows
	Writing better, robust, and portable code
	Testing and maintaining your code
	Summary

What defines “better” code?

- Better organized?
- Smaller?
- Faster?
- More stable?
- More portable?
- Easier to maintain?
- …

YES!

Writing more robust code

```
>> y = myfunc( 1:5 )
Index exceeds matrix dimensions.

Error in mypkg1.mypkg1a.mypkg1ab.myfunc1 (line 9)
y(idx) = u(idx)*log(u_hat(idx))+(1-u(idx))*log(1-u_hat(idx));

Error in mypkg2.mypkg2a.myfunc2 (line 5)
y = mypkg1.mypkg1a.mypkg1ab.myfunc1( myVar1 .* myVar2 );

Error in mypkg3.mypkg3a.myfunc3>@(x)mypkg2.mypkg2a.myfunc2(x) (line 4)
y = arrayfun( @(x) mypkg2.mypkg2a.myfunc2( x ), myVar );

Error in mypkg3.mypkg3a.myfunc3 (line 4)
y = arrayfun( @(x) mypkg2.mypkg2a.myfunc2( x ), myVar );

Error in myfunc (line 10)
```

Writing more robust code – Validating inputs

- `validateattributes`
- `isempty, isnan, isfinite, …`
- `narginchk`
- `inputParser`
- Property validation for classes

```
1  function y = myfunc( x )
2
3  % Validate inputs
4  validateattributes(x, 'double', {'size', [1 3], 'increasing'});
5
```

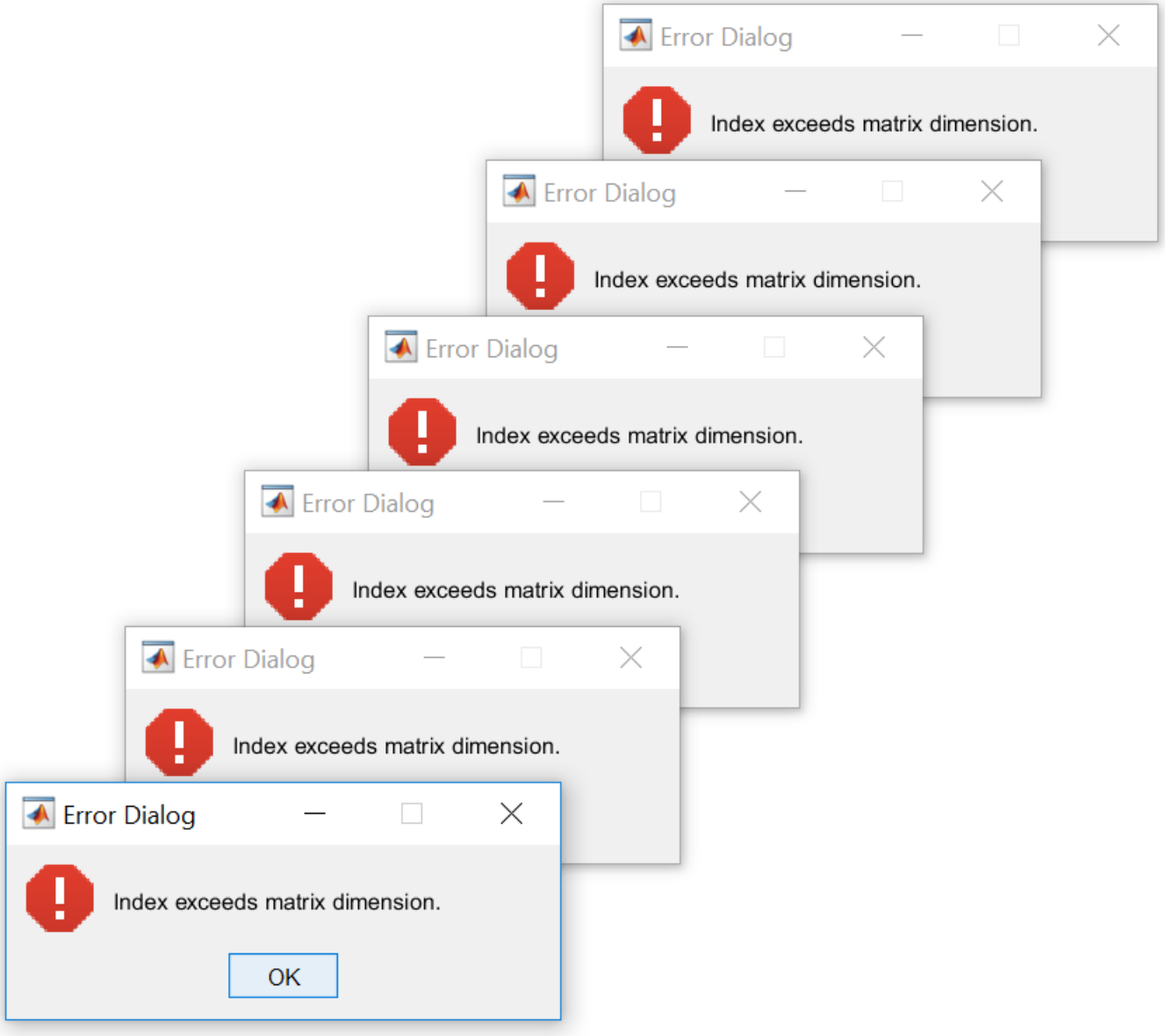
```
>> myfunc( 1:5 )
Error using myfunc (line 4)
Expected input to be of size 1x3, but it is of size 1x5.

>> myfunc( [2 3 1] )
Error using myfunc (line 4)
Expected input to be increasing valued.
```

```
classdef ValidatorFunction
   properties
      Data(:,1) double {mustBePositive, mustBeFinite} = [1 2 3]
      Interp {mustBeMember(Interp,{'linear','spline'})} = 'linear'
   end
end
```

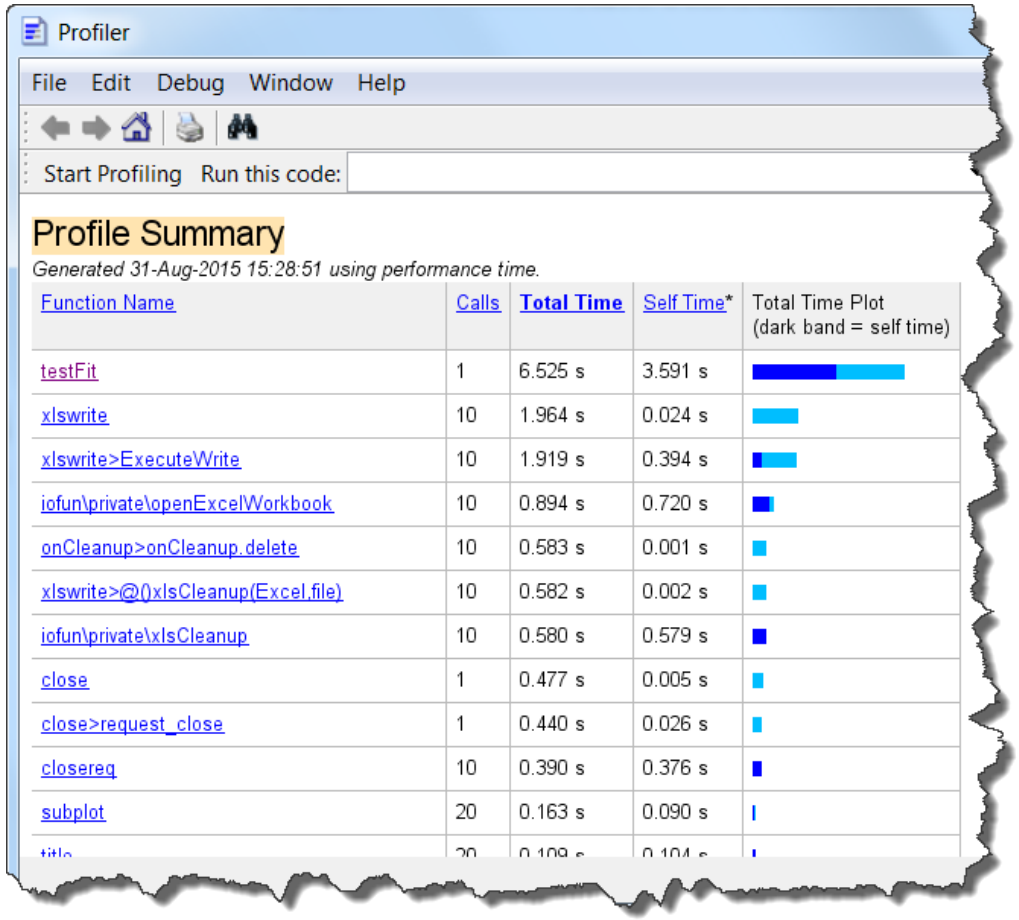
Writing more robust code – Handling errors more elegantly

- `error` and `warning`
  - Use identifiers
- `Mexception`
- `try/catch`
- `errordlg` and `warndlg`



Writing faster code – MATLAB Profiler

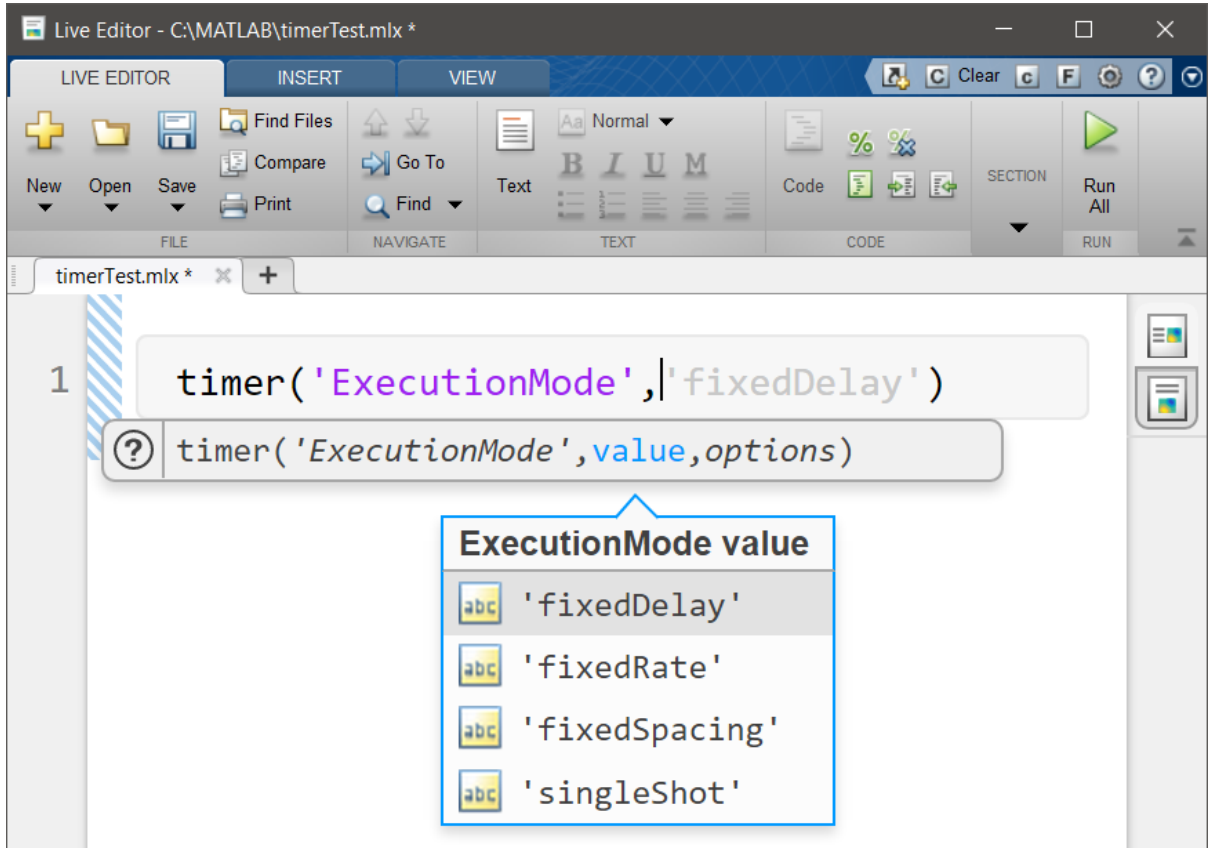
- Total number of function calls
- Time per function call
- Highlights largest code bottlenecks
- Statement coverage of code



Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
testFit	1	6.525 s	3.591 s	
xlswrite	10	1.964 s	0.024 s	
xlswrite>ExecuteWrite	10	1.919 s	0.394 s	
iofun\private\openExcelWorkbook	10	0.894 s	0.720 s	
onCleanup>onCleanup.delete	10	0.583 s	0.001 s	
xlswrite>@()xlsCleanup(Excel,file)	10	0.582 s	0.002 s	
iofun\private\xlsCleanup	10	0.580 s	0.579 s	
close	1	0.477 s	0.005 s	
close>request_close	1	0.440 s	0.026 s	
closereq	10	0.390 s	0.376 s	
subplot	20	0.163 s	0.090 s	
title	20	0.109 s	0.104 s	

Writing code faster – Programming aids in the Live Editor

- Automatically closed parentheses, loops, and conditional blocks
- Context-aware coding guides
  - Automatically suggest function names variables, or file names
  - List available Name/Value pairs



Writing code faster – Quickly and safely refactoring code

- Live Editor shortcuts to refactor blocks of code into functions

Calculate my answer:

```
3  z1 = x+y;
4  z2 = x-y;
5  z3 = y-x;
6  z4 = x*y;
7  zSum = z1 + z2 + z3 + z4;
```

Display answers of interest:

```
8  disp(z3)
9  disp(zSum)
```

Evaluate Selection in Command Window	F9
Open Selection	Ctrl+D
Help on Selection	F1
Copy Output	
Copy All Output	
Cut	Ctrl+X
Copy	Ctrl+C
Paste	Ctrl+V
Comment	Ctrl+R
Uncomment	Ctrl+T
Convert Between Code and Text	Ctrl+E
Change Case	Ctrl+Shift+A
Smart Indent	Ctrl+I
Convert to Function	
Convert to Local Function	
Insert Section Break	Ctrl+Alt+Enter
Run Section	Ctrl+Enter
Clear All Output	

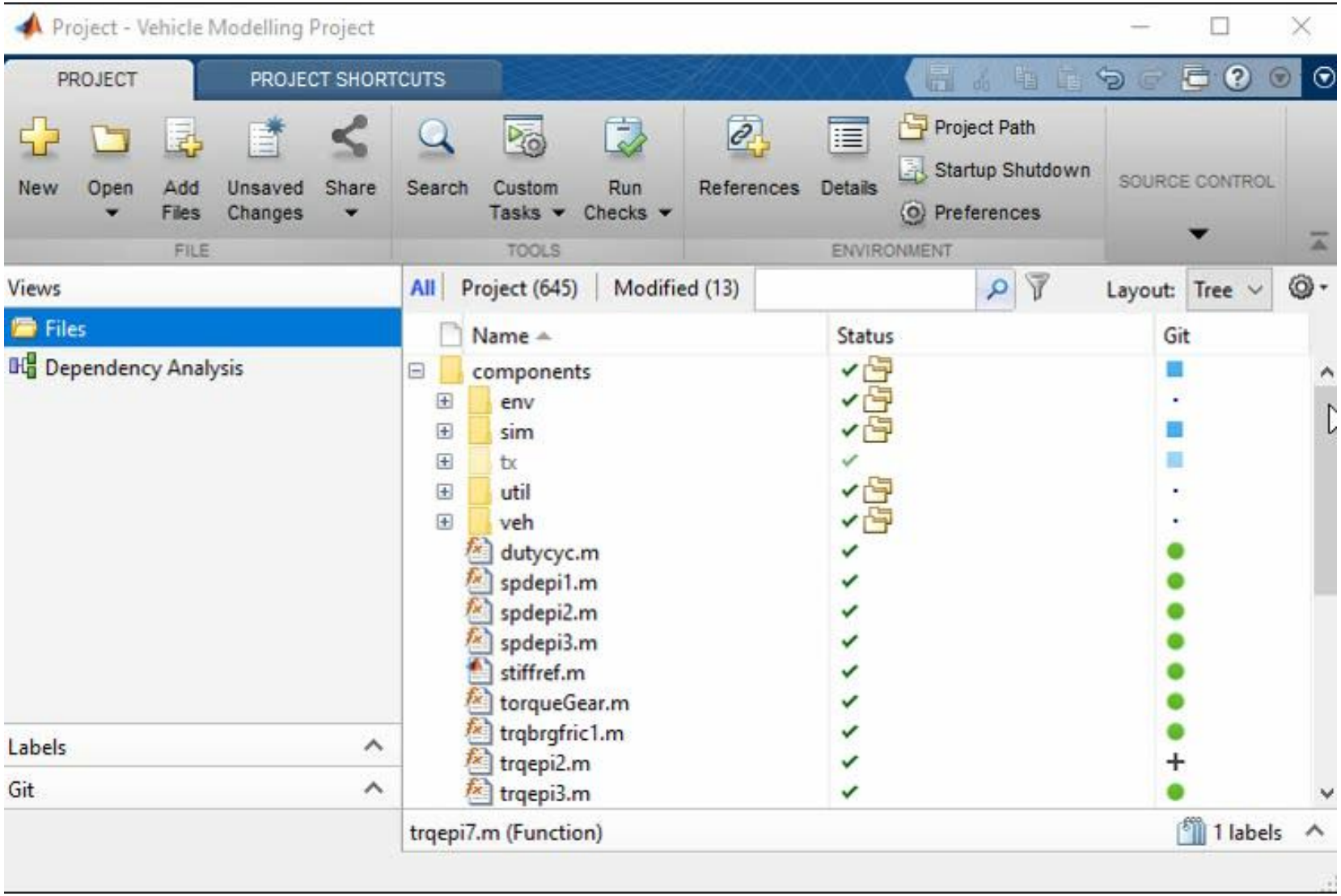
Live Editor - C:\MATLAB\TestFolder\myMathFunction.mlx

MainScript.mlx * | myMathFunction.mlx

```
1  function [z3, zSum] = myMathFunctio (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
```

Writing code faster – Quickly and safely refactoring code

- Function refactoring across files in Projects



Simple code quality and complexity assessment – `checkcode`

- Analyze all warnings and errors in a code

```
>> checkcode standardizeEmployeeInfo
L 13 (C 14-24): The value assigned here to 'maxDatetime' appears to be unused. Consider replacing it by ~.
L 80 (C 1-27): The value assigned to variable 'emailsInUsernameFormatParts' might be unused.
L 116 (C 1-17): The value assigned to variable 'validEmployeeData' might be unused.
L 118 (C 1-28): The value assigned to variable 'emailsInFirstLastFormatParts' might be unused.
```

- McCabe Cyclomatic Complexity
  - Measures complexity based on the number of linearly independent paths through a code

```
>> checkcode -cyc standardizeEmployeeInfo
L 1 (C 14-36): The McCabe cyclomatic complexity of 'standardizeEmployeeInfo' is 13.
```

Writing more portable code – Code that runs everywhere

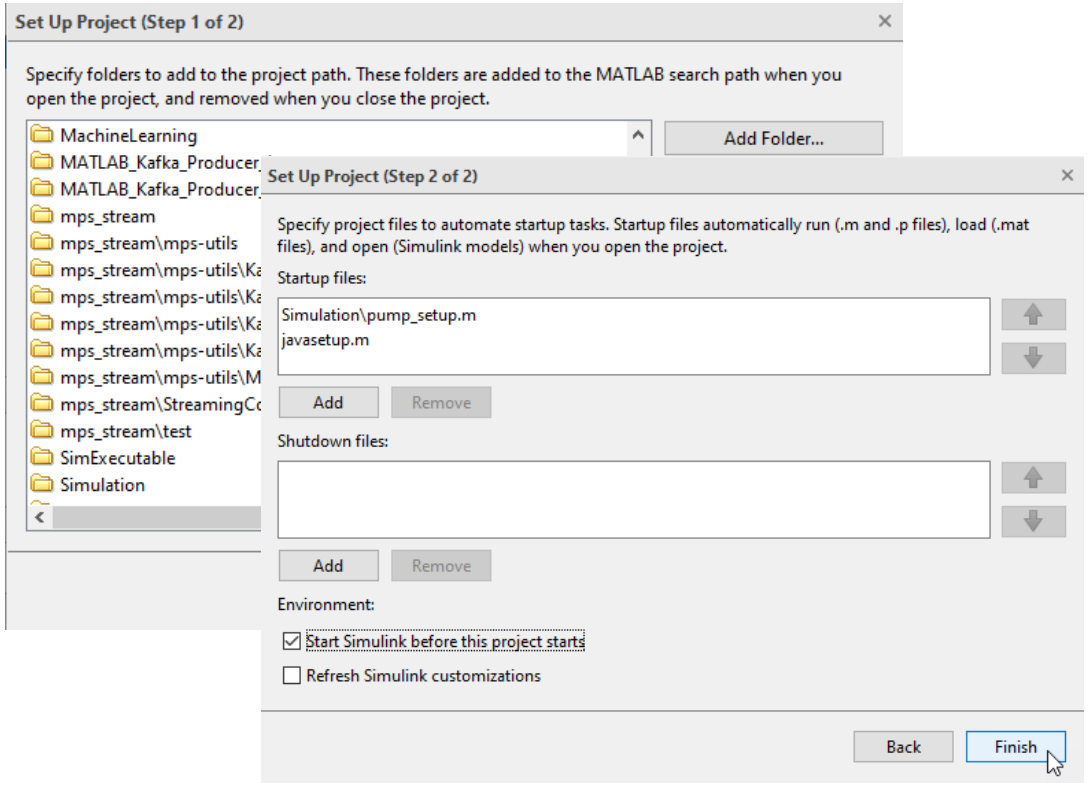
- Operating System-aware code
  - `fullfile`
  - `ispc, ismac, isunix`

```
>> fullfile("..","data","2019","April")
```

Windows: `"..\data\2019\April"`

Mac/Linux: `"../data/2019/April"`

- More reliable portability with Projects
  - Consistent path management
  - Automated startup/shutdown procedures
  - Built-in file dependency analysis

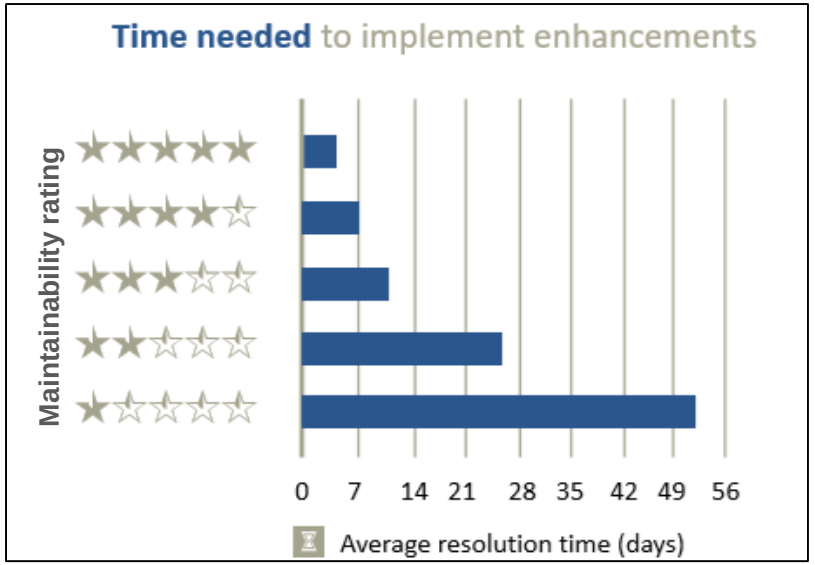


Agenda

	Managing your code
	Tracking code changes and co-authoring workflows
	Writing better, robust, and portable code
	Testing and maintaining your code
	Summary

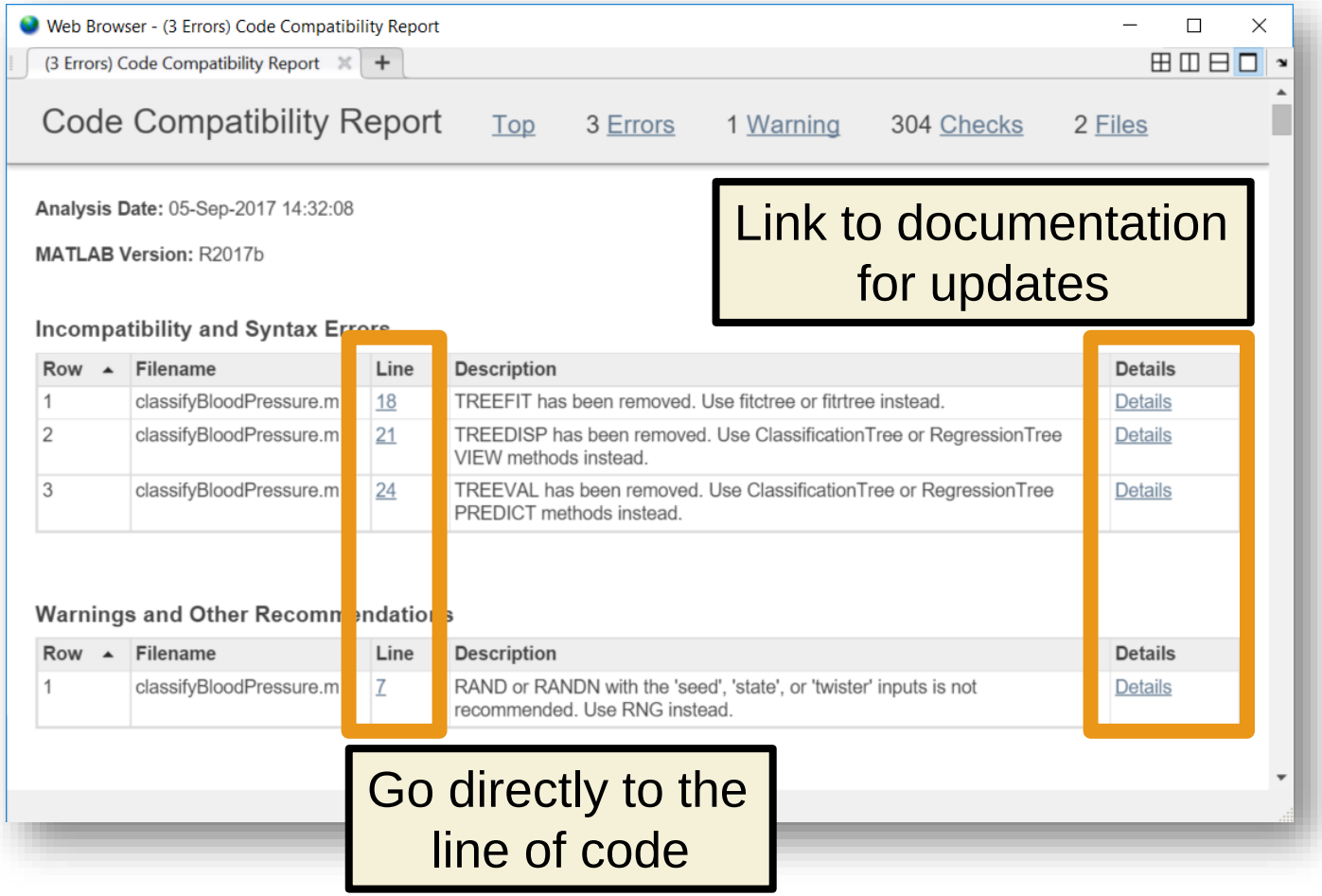
Code Maintenance – The hidden cost of development

- How do you ensure code doesn't break over time?
- How do you keep new features from breaking existing features?
- How do you maintain confidence that your code is working as expected?



Upgrading to the latest MATLAB – Code Compatibility Report

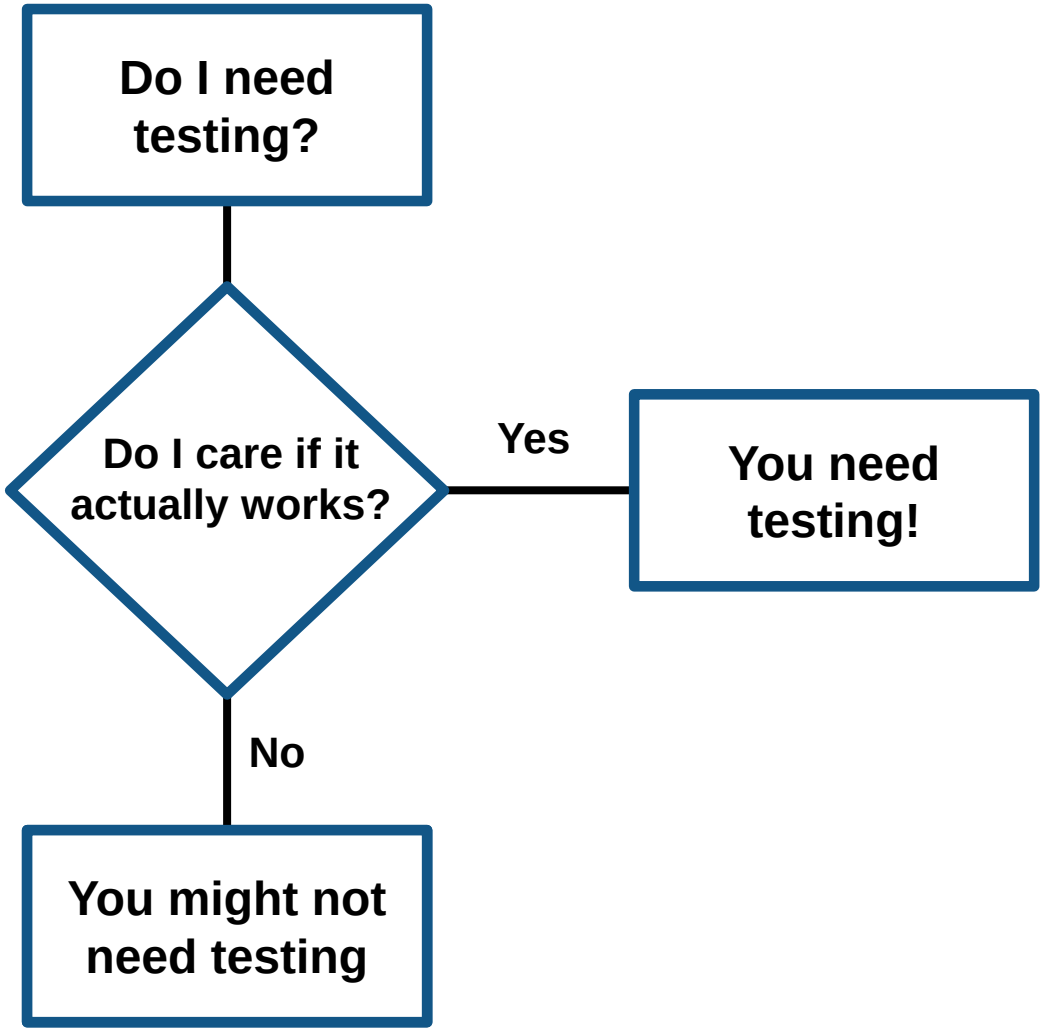
- Tool to help upgrade code to latest and greatest MATLAB
- Identifies potential compatibility issues
- Hundreds of checks for incompatibilities, errors, and warnings



Test early, test often, test automatically

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

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



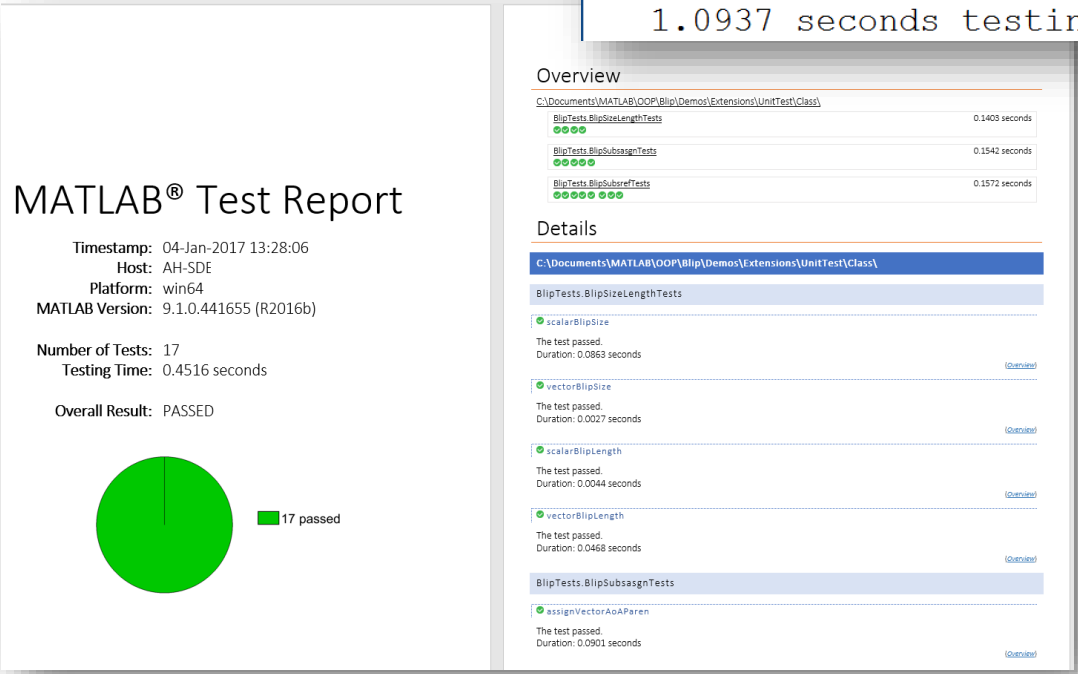
Testing Frameworks

Test your code early and often

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

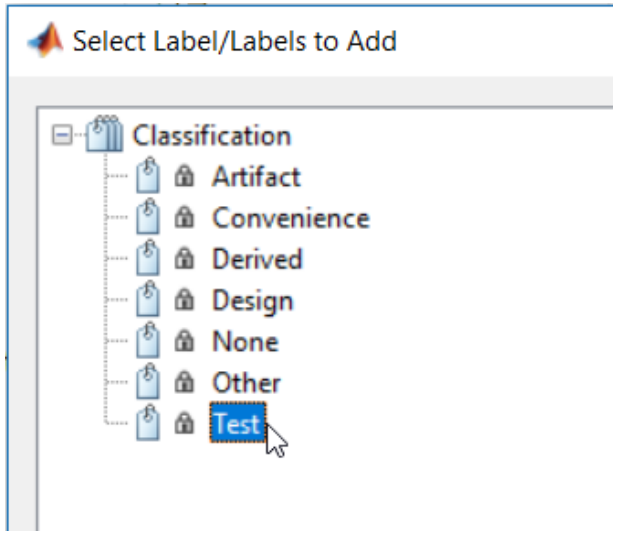
```
results =
  1×17 TestResult array with properties:

    Name
    Passed
    Failed
    Incomplete
    Duration
    Details
Totals:
   17 Passed, 0 Failed, 0 Incomplete.
   1.0937 seconds testing time.
```



Testing Frameworks – Flexible development

- Script-based test
- Function-based test
- Class-based test
- Test integration with Projects



test_Predictions.mlx

Test Pump Fault Model

This includes unit tests for the predictions

Test: Model type

Load the models and ensure they are the right types.

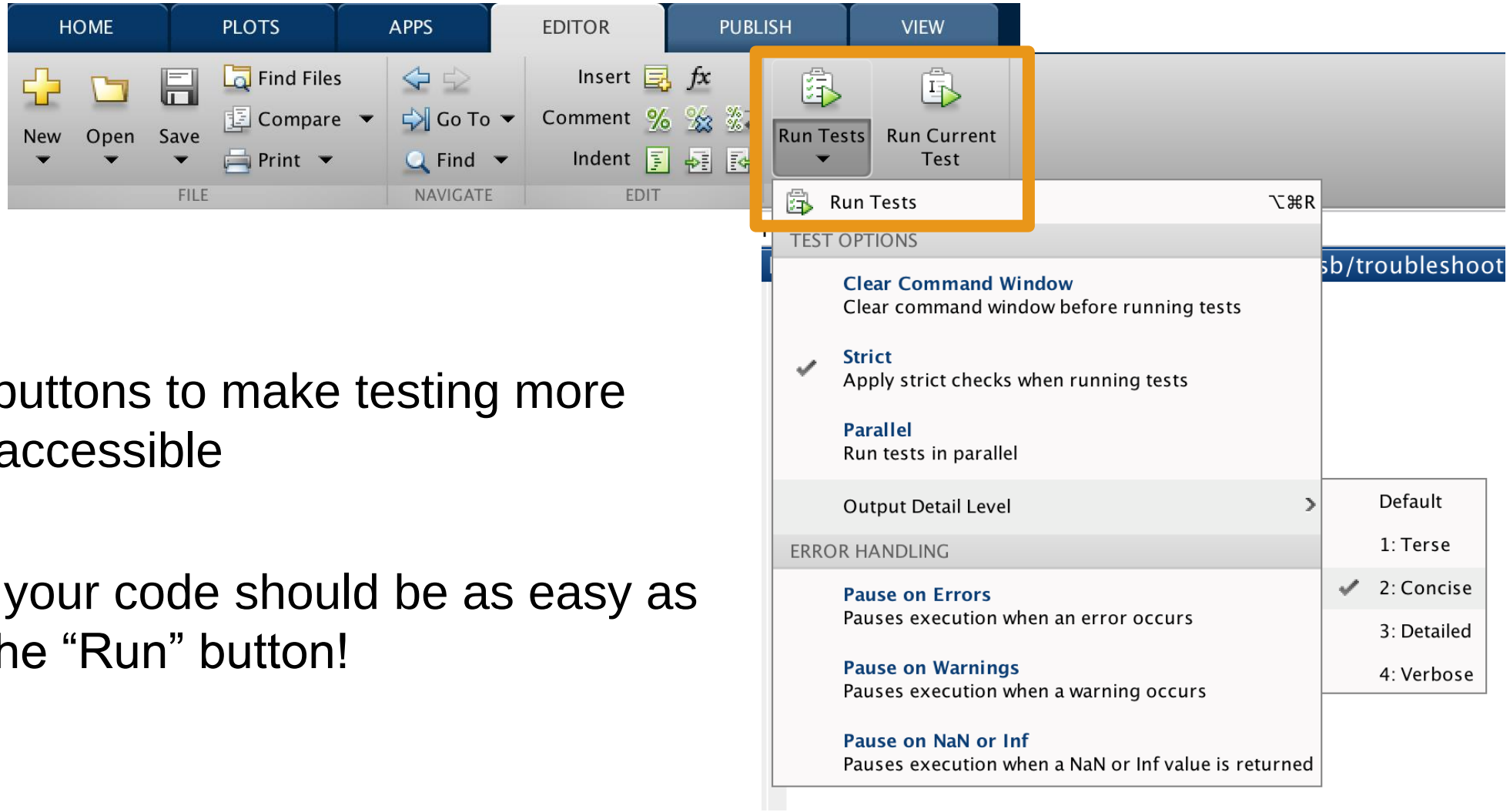
```
load MLModels trainedModel
mdl = trainedModel.ClassificationEnsemble;
assert(isa(mdl,'classreg.learning.classif.CompactClassificationEnsemble'),...
    'Model is not a CompactClassificationEnsemble.')
```

Test: Prediction

Ensure a prediction is returned from the model using predictFcn.

```
load MLModels trainedModel
load MLData data
FaultType = trainedModel.predictFcn(data);
assert(length(FaultType) == height(data))
assert(iscategorical(FaultType))
```

Testing Frameworks – Easily customize and run existing tests

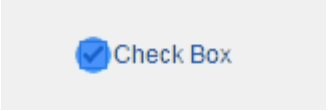


- Added buttons to make testing more readily accessible
- Testing your code should be as easy as hitting the “Run” button!

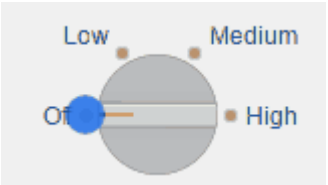
Testing Frameworks – App Testing Framework

- Verify app behavior with tests that programmatically perform gestures on a UI component

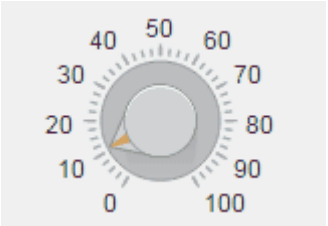
```
testCase.press(myApp.checkbox)
```



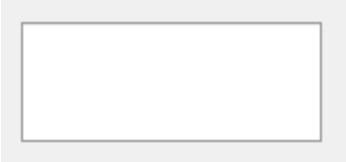

```
testCase.choose(myApp.discreteKnob, "Medium")
```




```
testCase.drag(myApp.continuousKnob, 10, 90)
```




```
testCase.type(myApp.editfield, myTextVar)
```

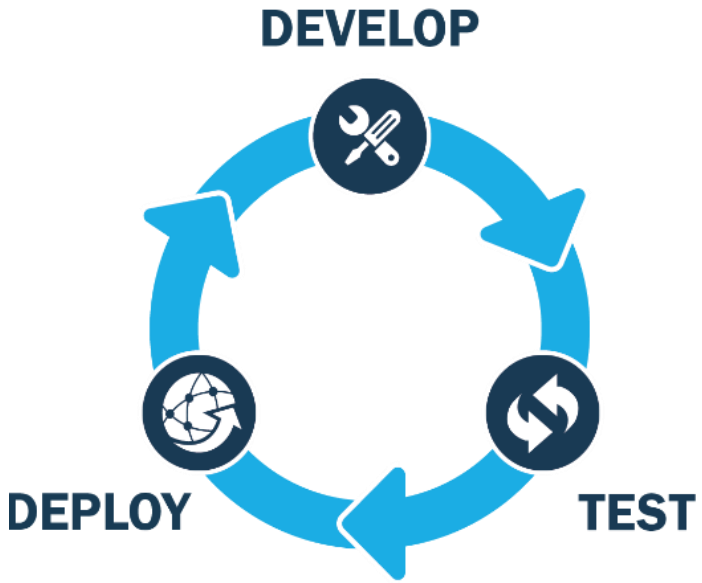


Automated Testing – Continuous Integration (CI)

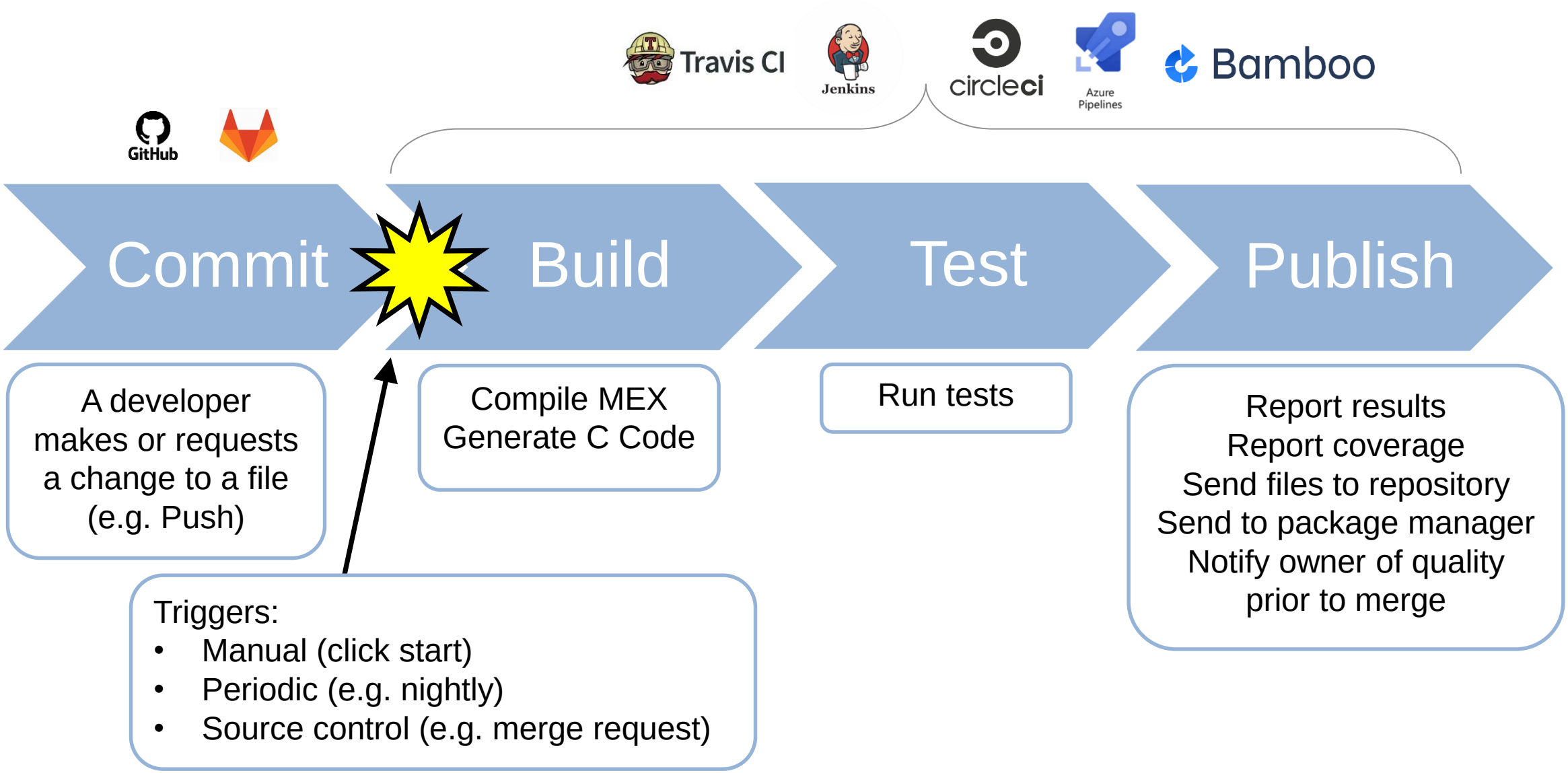
- A system to automate the building, testing, integration, and deployment of code as it is being developed and maintained

- Popular CI systems: Jenkins, Travis, CircleCI , Bamboo, 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



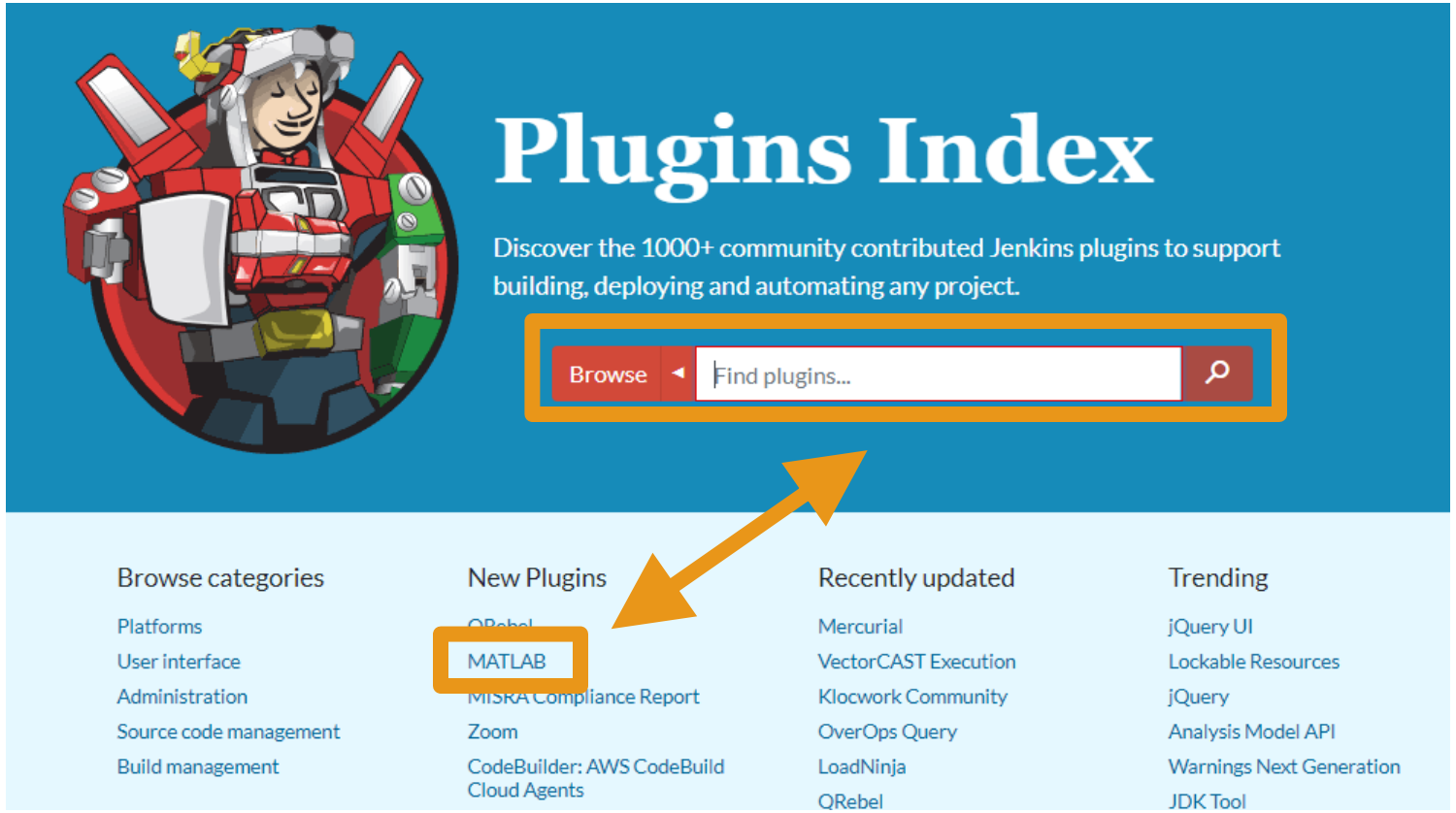
Automated Testing – Continuous Integration workflow



Automated Testing – Jenkins plugin

- Easily connect and configure MATLAB with Jenkins

- Schedule automatic code execution and testing:
  - based on time of day
  - whenever new code changes are committed



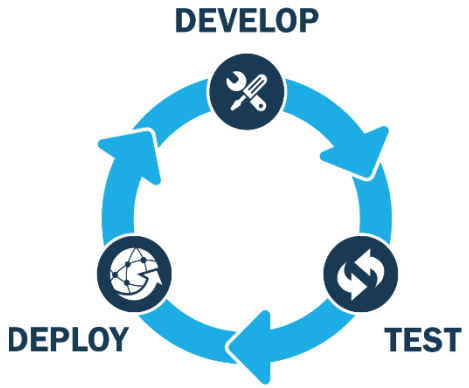
Automated Testing – Jenkins plugin – Testing reports

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

Agenda

	Managing your code
	Tracking code changes and co-authoring workflows
	Writing better, robust, and portable code
	Testing and maintaining your code
	Summary

Summary



- Good software development practices save you:
  - time
  - money
  - effort
  - frustration
- MATLAB makes good software development practices easy and automated
  - Projects
  - Source control
  - MATLAB Profiler
  - MATLAB Code Analyzer
  - Interactive programming aids
  - Code Compatibility Report
  - MATLAB Testing Frameworks
  - And more!
- We’re adding more software development tools and features every release!

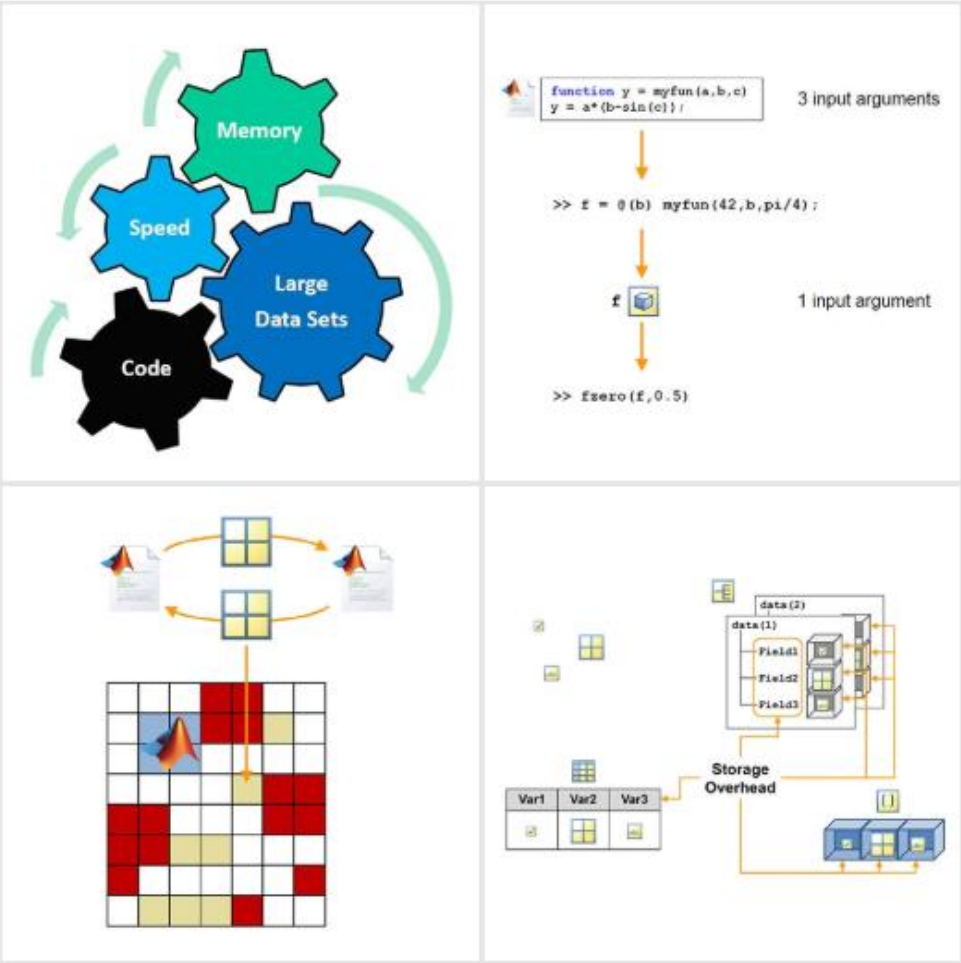
Training (Self – paced or instructor led) / Consulting

MATLAB Programming Techniques

This two-day course provides hands-on experience using the features in the MATLAB® language to write efficient, robust, and well-organized code. These concepts form the foundation for writing full applications, developing algorithms, and extending built-in MATLAB capabilities. Details of performance optimization, as well as tools for writing, debugging, and profiling code are covered. Topics include:

- Utilizing development tools
- Verifying application behavior
- Creating robust applications
- Structuring code
- Structuring data
- Managing data efficiently
- Creating a toolbox

See detailed course outline



Dates	Location	Language	Price	Register
On Demand	Self-Paced 180 days of full access from the day of purchase	English	AUD 350	