MATLAB tools for Sustainable Research Software Development

27.02.2025

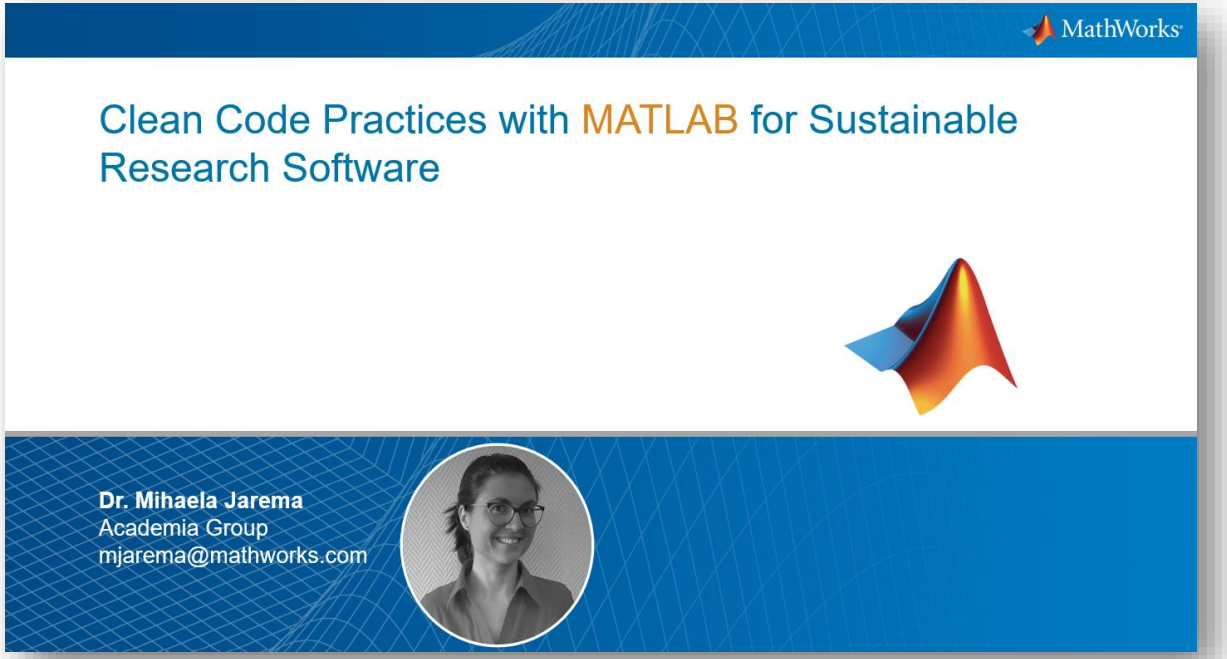
Dr. Mihaela Jarema
Academia Group
mjarema@mathworks.com

Warning: Your* findings and data can only be understood, reproduced, and built upon if the software you used in the research project is available, (re)usable, and extendable.

* *findings and data of those you collaborate with as RSEs*

Note: Today's content is designed for RSEs, researchers, and developers who have a foundational understanding of clean code practices and wish to **enhance** their **MATLAB skills for sustainable software development**.

!There is related and more detailed content for those without this foundation!



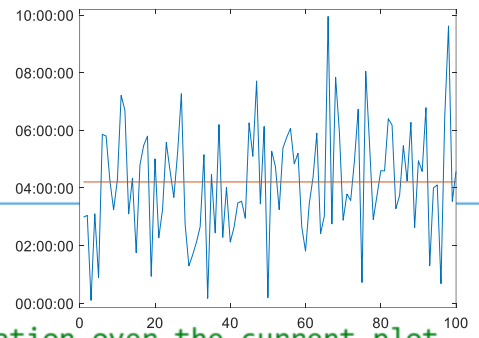
This is the *(overly simplified)* code and data we'll use to get started

- Access some data
- Explore data
- Process data
- Visualize processed data

evaluateProcessedItems.m

```
%% Access data
% Load variables from binary MATLAB file into workspace
load timestampsCN.mat entryTimestamps exitTimestamps

%% Explore data
% A MAT-file is a binary MATLAB file that stores workspace variables.
% Use whos to list all variables in the workspace, including their type and
% size
whos

%% Process data
% Compute elapsed durations between entryTimestamps and exitTimestamps
durations = exitTimestamps-entryTimestamps;
% Compute the number of durations
numberOfDurations = numel(durations);
% Compute the average duration
averageDuration = mean(durations);

%% Visualize data
% Plot all durations
figure, plot(durations)
% Add a horizontal line with the average duration over the current plot
hold on
plot([1,numberOfDurations],[averageDuration, averageDuration])
hold off
```

Name	Size	Bytes	Class
entryTimestamps	100x1	1600	datetime
exitTimestamps	100x1	1600	datetime



Our goal: Write code to analyze our data in a manner that is easy to maintain, reuse, modify, and extend (also by others).

Code Refactoring Tools

Extracting logically-related code into well-designed functions improves readability and reusability

Output arguments (results) the function returns (if any)

Name of your custom function

Input arguments the function takes (if any)

Start with keyword function

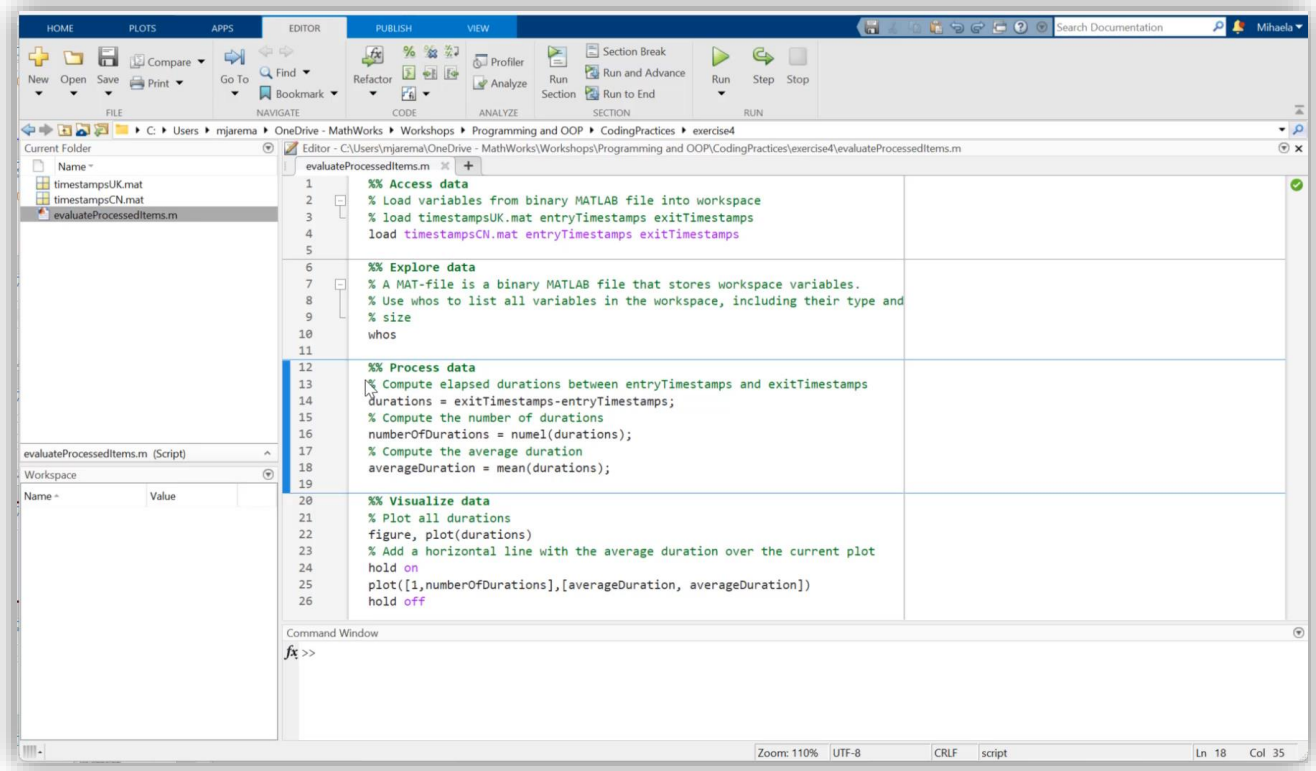
```
function [outputArg1,outputArg2] = functionName(inputArg1,inputArg2)
%functionName Summary of this function goes here
%   Detailed explanation goes here
%
% Input arguments that the function takes
%   inputArg1
%   inputArg2
%
% Output arguments that the function returns
%   outputArg1
%   outputArg2

% Sequence of statements performing computations on the function inputs
% and returning results as the function outputs
outputArg1 = inputArg1;
outputArg2 = inputArg2;
end
```

Function Body

Function end

MATLAB makes it easy to extract code into functions

- **We can refactor our code by extracting code into functions:** divide code into smaller logical units by moving fragments of logically-related code into new functions with names that explain what the code fragments do.

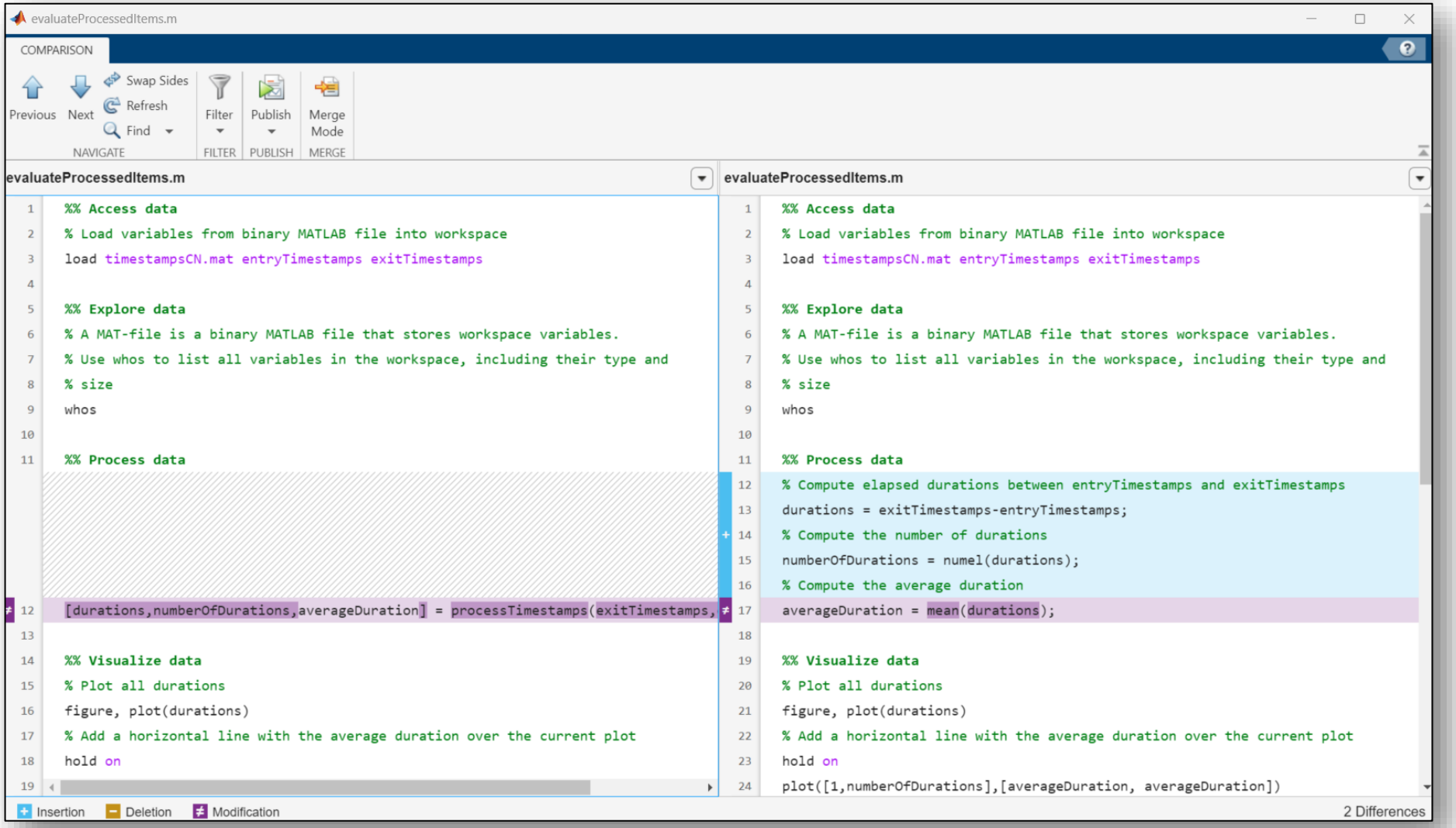


Your turn: Refactor the code to process timestamps

- Prerequisites:
  - Copy the materials via the MATLAB Drive https://drive.mathworks.com/sharing/6f91641d-8ddc-4c5d-98d1-90398fbb7197
    - click **Download Shared Folder** to save a .zip file and work with the files on your MATLAB Desktop if you have a recent MATLAB version available.
    - click **Add to my Files > Copy Folder** to work with the files in MATLAB Online. You will need a MathWorks account with the email address at your institution.
- Your turn:
  - Go to your version of **CleanCodePractices>exercise1** to refactor the code.

! *You can find a solution in* ***CleanCodePractices>exercise2***

Tip: We can use the **Comparison Tool** to display the differences between selected pairs of files



Testing Frameworks

Imagine we want to add functionality to, e.g., determine which processing durations took too long

- Let's add a function to compute which & how many processing durations took longer than a given threshold …

getLongProcessingDurations.m

```
function [numberOfDurations,isLong] = getLongProcessingDurations(durations,durationThreshold)
% items with long processing durations larger than delayThreshold
%
%   [numberOfDurations,isLong] = getLongProcessingDurations(durations,...
%   durationThreshold) returns the number of items with a processing
%   duration larger than the provided durationThreshold and a logical array
%   storing which item had a long processing duration.
isLong = durations>durationThreshold;
numberOfDurations = length(isLong);
end
```

! *You can find the function in* ***CleanCodePractices>exercise3***

Imagine we want to add functionality to, e.g., determine which processing durations took too long

- Is our newly added code correct? Let's run the function with a durationThreshold of 0 seconds *(e.g., all our 100 processing durations should take too long…).*

getLongProcessingDurations.m

```
function [numberOfDurations,isLong] = getLongProcessingDurations(durations,durationThreshold)
% items with long processing durations larger than delayThreshold
%
%   [numberOfDurations,isLong] = getLongProcessingDurations(durations,...
%   durationThreshold) returns the number of items with a processing
%   duration larger than the provided durationThreshold and a logical array
%   storing which item had a long processing duration.
isLong = durations>durationThreshold;
numberOfDurations = length(isLong);
end
```

```
>> numberOfDurations = getLongProcessingDurations(durations,seconds(0))

numberOfDurations =

   100
```

Imagine we want to add functionality to, e.g., determine which processing durations took too long.

- Is our newly added code correct? Let's run the function with a durationThreshold so large that none of our processing durations take too long …

getLongProcessingDurations.m

```
function [numberOfDurations,isLong] = getLongProcessingDurations(durations,durationThreshold)
% items with long processing durations larger than delayThreshold
%
%   [numberOfDurations,isLong] = getLongProcessingDurations(durations,...
%   durationThreshold) returns the number of items with a processing
%   duration larger than the provided durationThreshold and a logical array
%   storing which item had a long processing duration.
isLong = durations>durationThreshold;
numberOfDurations = length(isLong);
end
```

```
>> numberOfDurations = getLongProcessingDurations(durations,max(durations)+seconds(1))

numberOfDurations =

   100
```

! length *returns the number of all elements in the array instead of how many of them meet the condition* !

Imagine we want to add functionality to, e.g., determine which processing durations took too long.

- Let's change the code to fix the bug … and recheck everything …

getLongProcessingDurations.m

```
function [numberOfDurations,isLong] = getLongProcessingDurations(durations,durationThreshold)
% items with long processing durations larger than delayThreshold
%
%   [numberOfDurations,isLong] = getLongProcessingDurations(durations,...
%   durationThreshold) returns the number of items with a processing
%   duration larger than the provided durationThreshold and a logical array
%   storing which item had a long processing duration.
isLong = durations>durationThreshold;
numberOfDurations = nnz(isLong);
end
```

```
>> numberOfDurations = getLongProcessingDurations(durations,max(durations)+seconds(1))

numberOfDurations =

     0
```

*Works **now**, after bug fix!* ☺

```
>> numberOfDurations = getLongProcessingDurations(durations,seconds(0))

numberOfDurations =

   100
```

Still works after bug fix! ☺

But we had to test again manually, which is bound to become cumbersome as we add more functionality and need to test for the same (and more) things … ☹

What if we could automatize testing our code behaves as intended?

- We can use **unit tests** to test each **unit** *(e.g., function)* in our code behaves as intended.

When our code is used in this way,

```
actualNumberOfDurations = getLongProcessingDurations(durations,seconds(0));
```

a certain result/behavior

```
expectedNumberOfDurations = length(durations);
```

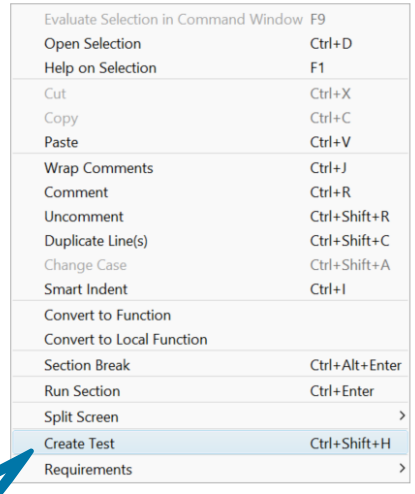
should occur.

```
assert(actualNumberOfDurations == expectedNumberOfDurations)
```

* assert throws an error if the tested condition is false.

We can autogenerate a sample test class with a sample test function directly from the Editor

```
function [numberOfDurations,isLong] = getLongProcessingDurations(durations,durationThreshold)
% items with long processing durations larger than delayThreshold
%
%   [numberOfDurations,isLong] = getLongProcessingDurations(durations,...
                                                                  essing
                                                                  logical array
```




```
% This is an autogenerated sample test for file getLongProcessingDurations.m
classdef testgetLongProcessingDurations < matlab.unittest.TestCase

    methods (Test)

        function test_getLongProcessingDurations(testCase)
            % Specify the input(s) of
            % getLongProcessingDurations

            durations = ;
            durationThreshold = ;

            % Specify the expected output(s) of
            % getLongProcessingDurations

            expected_numberOfDurations = ;
            expected_isLong = ;

            % Exercise the function getLongProcessingDurations
            [actual_numberOfDurations, actual_isLong] = getLongProcessingDurations(durations, durationThreshold);

            testCase.verifyEqual(actual_numberOfDurations, expected_numberOfDurations);
            testCase.verifyEqual(actual_isLong, expected_isLong);
        end
    end
end
```

! *You can find the correct function in* ***CleanCodePractices>exercise4****, and the function and sample test in* ***exercise5***

Your turn: Generate a sample test and fill it in to test that all processing durations take longer than 0 seconds

- Go to your version of **CleanCodePractices>exercise4**, generate a sample test, and fill it in to test that all processing durations take longer than 0 seconds.
- Use the skeleton code provided in **CleanCodePractices>exercise5** if you cannot generate a sample test.

! *You can find the solution in* ***CleanCodePractices>exercise6***

We can complete the test …

```
% This is an autogenerated sample test for file getLongProcessingDurations.m
classdef testgetLongProcessingDurations < matlab.unittest.TestCase

    methods (Test)

        function test_getLongProcessingDurations(testCase)
            % Specify the input(s) of
            % getLongProcessingDurations

            durations = ;
            durationThreshold = ;

            % Specify the expected output(
            % getLongProcessingDurations

            expected_numberOfDurations = ;
            expected_isLong = ;

            % Exercise the function getLon
            [actual_numberOfDurations, act

            testCase.verifyEqual(actual_nu
            testCase.verifyEqual(actual_is
        end
    end
end
```

```
% This is an autogenerated sample test for file getLongProcessingDurations.m
classdef testGetLongProcessingDurations < matlab.unittest.TestCase

    methods (Test)

        function test_AllLong(testCase)
            % Specify the input(s) of
            % getLongProcessingDurations

            load timestampsCN.mat entryTimestamps exitTimestamps
            durations = exitTimestamps-entryTimestamps;
            durationThreshold = seconds(0);

            % Specify the expected output(s) of
            % getLongProcessingDurations

            expected_numberOfDurations = length(durations);

            % Exercise the function getLongProcessingDurations
            [actual_numberOfDurations, ~] = getLongProcessingDurations(durations, durationThreshold);

            testCase.verifyEqual(actual_numberOfDurations, expected_numberOfDurations);
        end
```

We can complete the test and add further test functions

```
% This is an autogenerated sample test for file getLongProcessingDurations.m
classdef testGetLongProcessingDurations < matlab.unittest.TestCase

    methods (Test)

        function test_allLong(testCase)
            % Specify the input(s) of
            % getLongProcessingDurations

            load timestampsCN.mat entryTimestamps exitTime
            durations = exitTimestamps-entryTimestamps;
            durationThreshold = seconds(0);

            % Specify the expected output(s) of
            % getLongProcessingDurations

            expected_numberOfDurations = length(durations)

            % Exercise the function getLongProcessingDurat
            [actual_numberOfDurations, ~] = getLongProcess

            testCase.verifyEqual(actual_numberOfDurations,
        end
```

```
function test_NoneLong(testCase)
    % Specify the input(s) of
    % getLongProcessingDurations

    load timestampsCN.mat entryTimestamps exitTimestamps
    durations = exitTimestamps-entryTimestamps;
    durationThreshold = max(durations)+seconds(1);

    % Specify the expected output(s) of
    % getLongProcessingDurations

    expected_numberOfDurations = 0;

    [actual_numberOfDurations,~] = getLongProcessingDurations(durations,...
        durationThreshold);

    % Exercise the function getLongProcessingDurations
    testCase.verifyEqual(actual_numberOfDurations, expected_numberOfDurations);
end
```

We can run tests interactively in the Editor or in the Test Browser app

EDITOR | PUBLISH | VIEW | Try the New Desktop | Search Documentation

Find | Bookmark | NAVIGATE | Refactor | CODE | Profiler | Analyze | ANALYZE | Insert Method | Insert Parameters | TEST | Run Section | Section Break | Run and Advance | Run to End | SECTION | Run Tests | Run Current Test | Step | Stop | RUN

OneDrive - MathWorks ▸ Workshops ▸ Programming and OOP ▸ testing ▸ building-robust-research-software-matlab-testing-strategies ▸ exercise2

Editor - C:\Users\mjarema\OneDrive - MathWorks\Workshops\Programming and OOP\testing\building-robust-research-software-matlab-testing-strategies\ex

testGetLongProcessingDurations.m

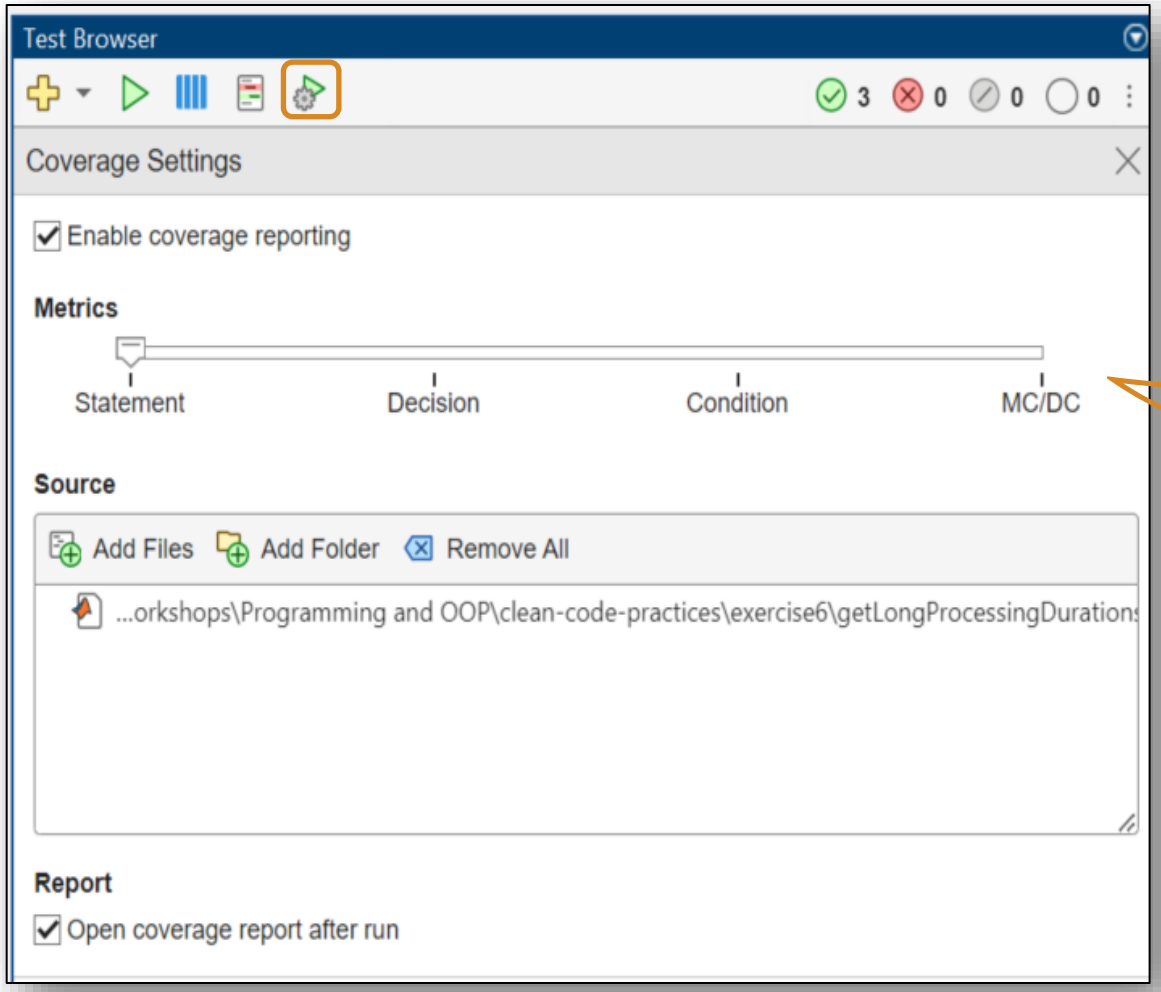
```
% This is an autogenerated sample test for file getLongProcessingDurations.m
classdef testGetLongProcessingDurations < matlab.unittest.TestCase

    methods (Test)

        function test_allLong(testCase)
            % Specify the input(s) of
            % getLongProcessingDurations

            load timestampsCN.mat entryTimestamps exitTimestamps
            durations = exitTimestamps-entryTimestamps;
            durationThreshold = seconds(0);

            % Specify the expected output(s) of
            % getLongProcessingDurations

            expected_numberOfDurations = length(durations);
```

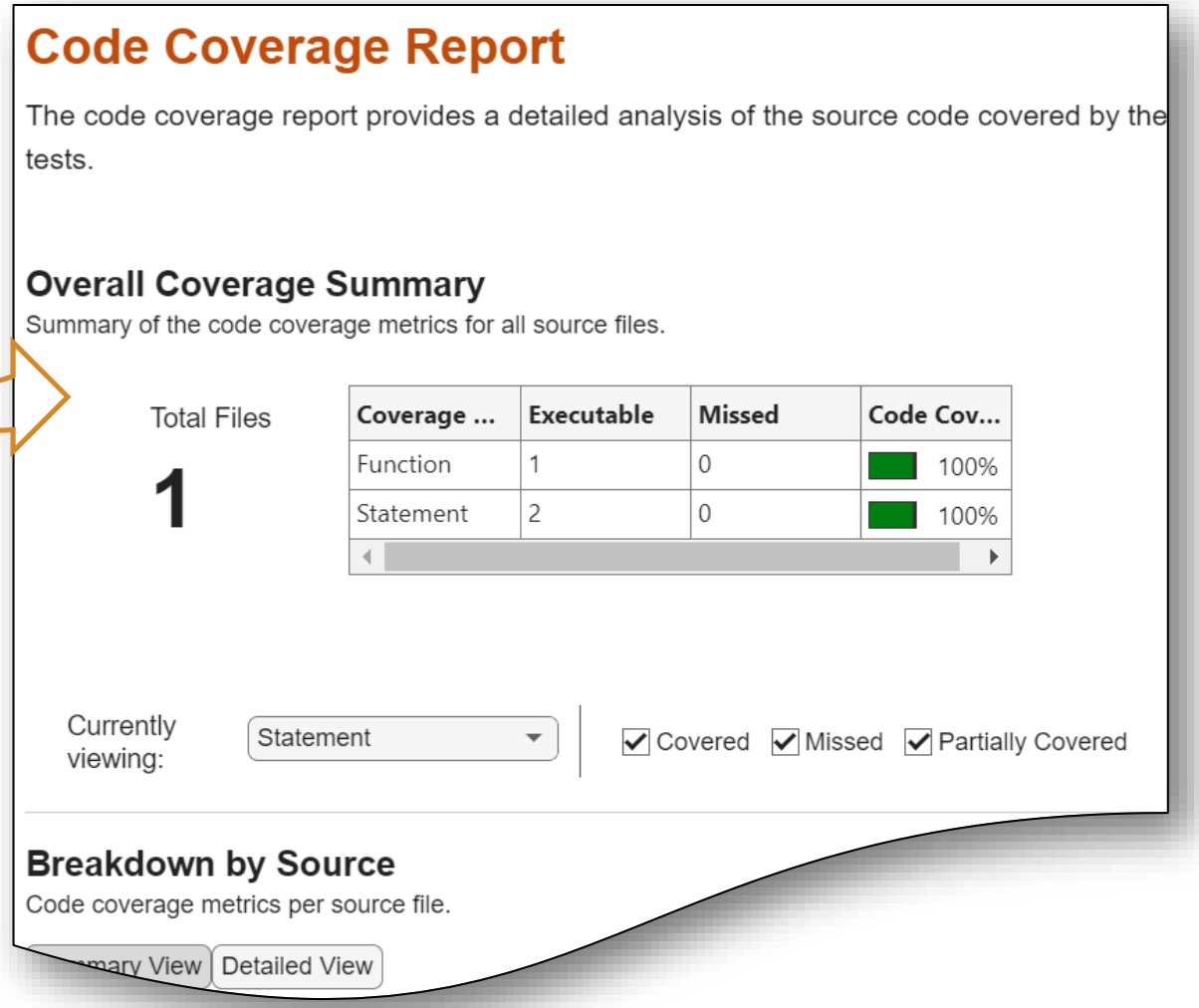
Test Browser

3 0 0 0

- testGetLongProcessingDurations
  - test_AllLong
  - test_NoneLong
  - test_MedianLong

With the Test Browser we can also interactively collect code coverage





Tip: We can extract setup/teardown code in test fixtures and avoid duplicated code

```
% This is an autogenerated sample test for file getLongProcessingDurations
classdef testGetLongProcessingDurations < matlab.unittest.TestCase

    properties
        durations
    end

    methods(TestClassSetup)

        function durationsSetup(testCase)
            % Set up shared state for all tests.
            load timestampsCN.mat entryTimestamps exitTimestamps
            testCase.durations = exitTimestamps-entryTimestamps;

            % Tear down with testCase.addTeardown.
            testCase.addTeardown(@clear,"entryTimestamps exitTimestamps");
        end
    end

    methods (Test)

        function test_AllLong(testCase)
            % Specify the input(s) of
            % getLongProcessingDurations

            durationThreshold = seconds(0);

            % Specify the expected output(s) of
            % getLongProcessingDurations

            expected_numberOfDurations = length(testCase.durations);
```

Your turn: Add test fixtures for all test functions

- Go to your version of **CleanCodePractices>exercise6** and add test fixtures.

! *You can find the solution in* ***CleanCodePractices>exercise7***

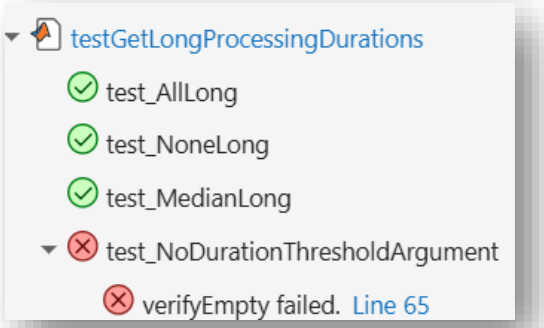
Imagine we'd like to call our function getLongProcessingDurations without a durationThreshold input argument …

- Let's add a test that should pass if we can run the function without a durationThreshold input argument and get no errors …

```
function test_NoDurationThresholdArgument(testCase)
    actualException = [];
    try
        getLongProcessingDurations(testCase.durations);
    catch actualException
    end
    testCase.verifyEmpty(actualException);
end
```

! *You can find the test in* ***CleanCodePractices>exercise8***

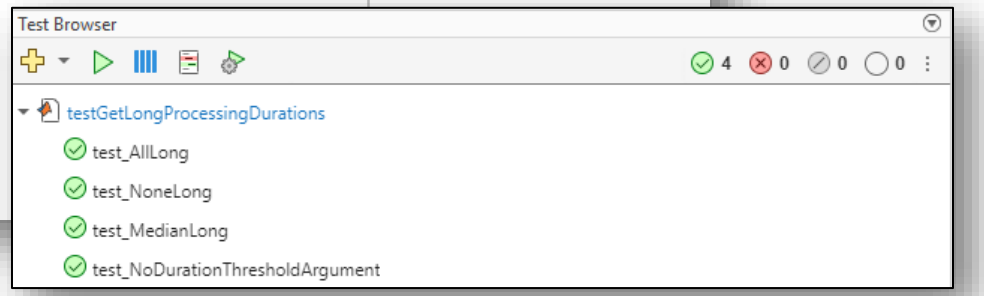
- The new test should fail *(because we have yet to change the behavior of our function …).*



Imagine we'd like to call our function getLongProcessingDurations without a durationThreshold input argument …

- We could add function code to pass the failed test by checking the number of input arguments and computing a durationThreshold if none is inputted.

```
function [numberOfDurations,isLong] = getLongProcessingDurations(durations,durationThreshold)
% items with long processing durations larger than delayThreshold
%
%   [numberofDurations,isLong] = getLongProcessingDurations(durations,...
%   durationThreshold) returns the number of items with a processing
%   duration larger than the provided durationThreshold and a logical array
%   storing which item had a long processing duration.
if nargin == 1
    durationThreshold = std(durations)*3;
end
isLong = durations>durationThreshold;
numberOfDurations = nnz(isLong);
end
```



But writing code in the function body to verify input arguments are valid before running function code could make our code harder to read and maintain …

Imagine we'd like to call our function getLongProcessingDurations without a durationThreshold input argument …

- We can use **function argument validation** to declare specific restrictions on input arguments without writing code in the body of the function to check these arguments.

```
arguments
    argument (dim1, dim2, ...)  ClassName  {fcn1, fcn2, ...} = defaultValue
end
```

Size | Class | Validator Functions | Default value

Imagine we'd like to call our function getLongProcessingDurations without a durationThreshold input argument…

- We can use **function argument validation** to declare specific restrictions on input arguments without writing code in the body of the function to check these arguments.

```
function [numberOfDurations,isLong] = getLongProcessingDurations(durations,durationThreshold)
% items with long processing durations larger than delayThreshold
%
%   [numberOfDurations,isLong] = getLongProcessingDurations(durations,...
%   durationThreshold) returns the number of items with a processing
%   duration larger than the provided durationThreshold and a logical array
%   storing which item had a long processing duration.
%
%   [numberOfDurations,isLong] = getLongProcessingDurations(durations)
%   uses a default value for the durationThreshold equal to three times the
%   standard deviation of the durations.
arguments
    durations (:,1) duration
    durationThreshold (1,1) duration = std(durations)*3
end

isLong = durations>durationThreshold;
numberOfDurations = nnz(isLong);
end
```

Organizing and Collaborating – Project Management

How can we organize our code and data so everybody can easily continue where we left off?

- We shouldn't assume others have the same hierarchical folder structure or the same operating system, but generate path and file names so they are valid for others and on other platforms:

```
>> f = fullfile("myfolder","mysubfolder","myfile")
```

Windows

```
f =

    'myfolder\mysubfolder\myfile.m'
```

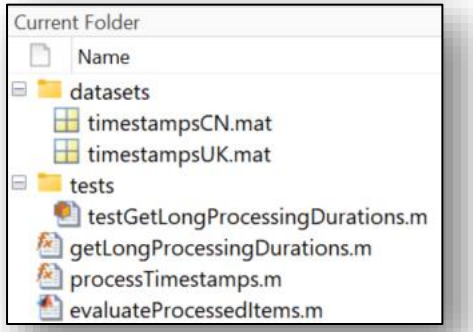
Linux

```
f =
'myfolder/mysubfolder/myfile.m'
```

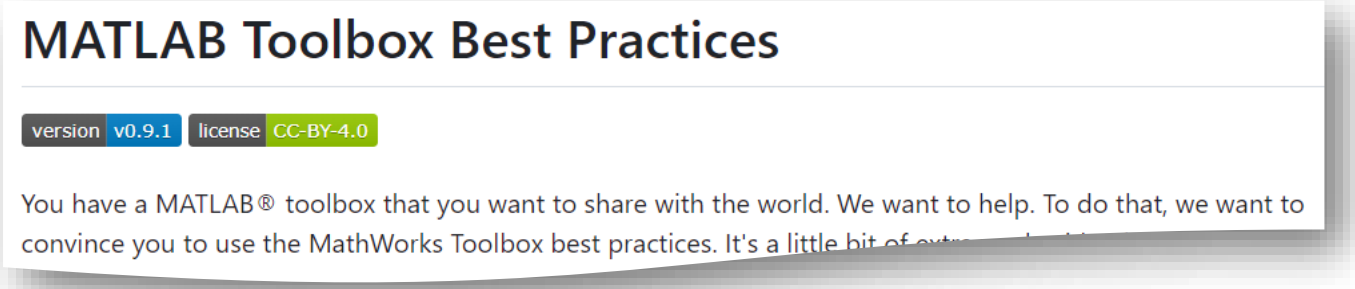
```
>> fun_dir = "toolbox" + filesep + "matlab" + filesep + "ionfun"

fun_dir =

    "toolbox\matlab\ionfun"
```

How can we organize our code and data so everybody can easily continue where we left off?

- We should structure our project code so that it is easy to maintain and build on, e.g., extract datasets and tests in their own folders …



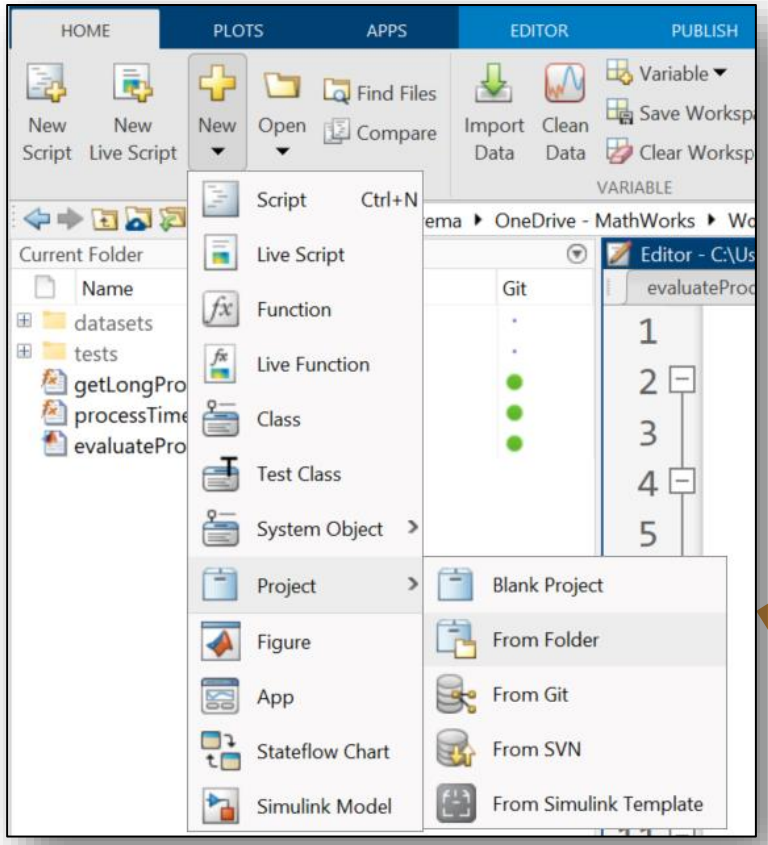
- For more complex projects, we can pick a general standard to follow, e.g.,

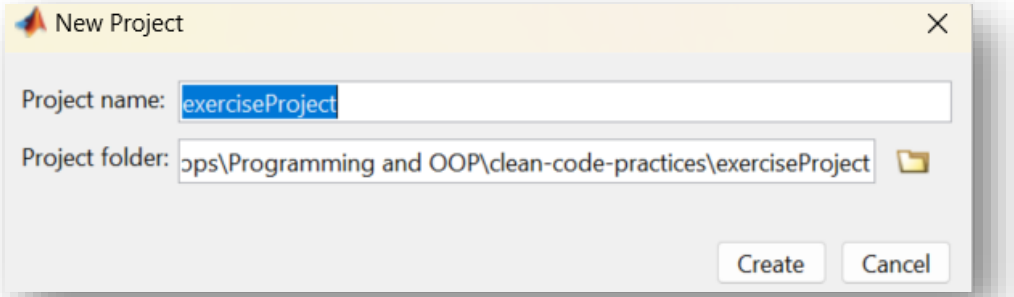


github.com/mathworks/toolboxdesign

How can we organize our code and data so everybody can easily continue where we left off?

We can create **MATLAB Projects** from an existing folder to automatically set up the working environment for anyone using the project …

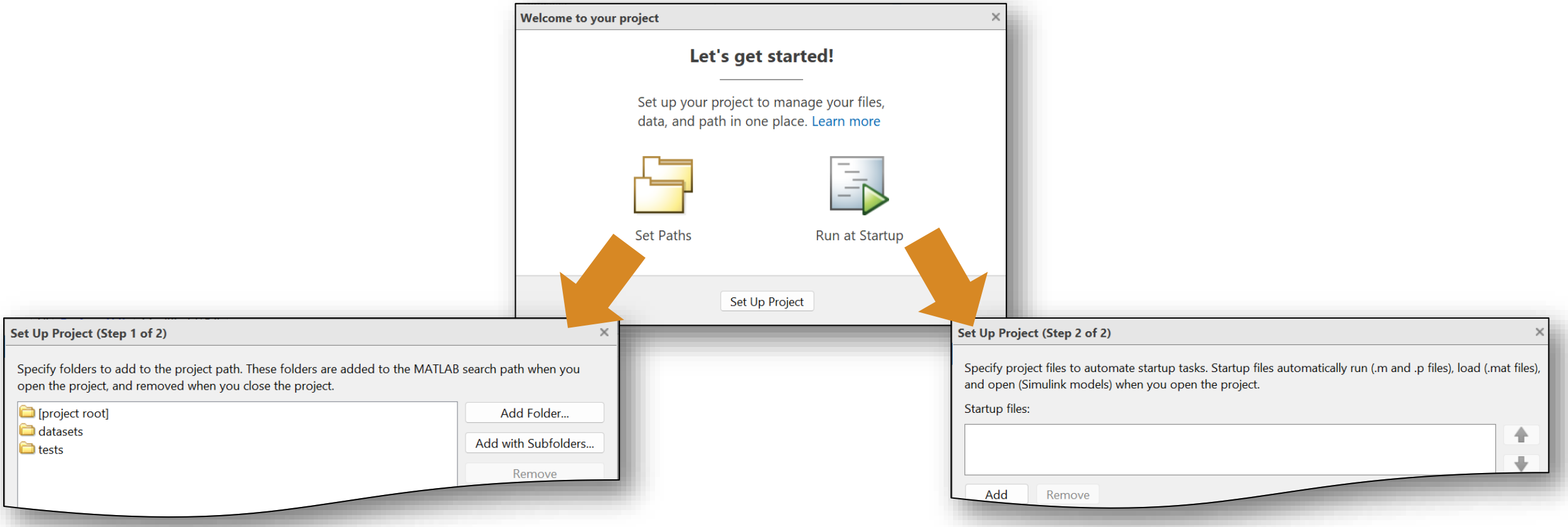




MATLAB Projects

We can organize our work with MATLAB Projects so that it is easy to maintain and build on for us and others using our code …

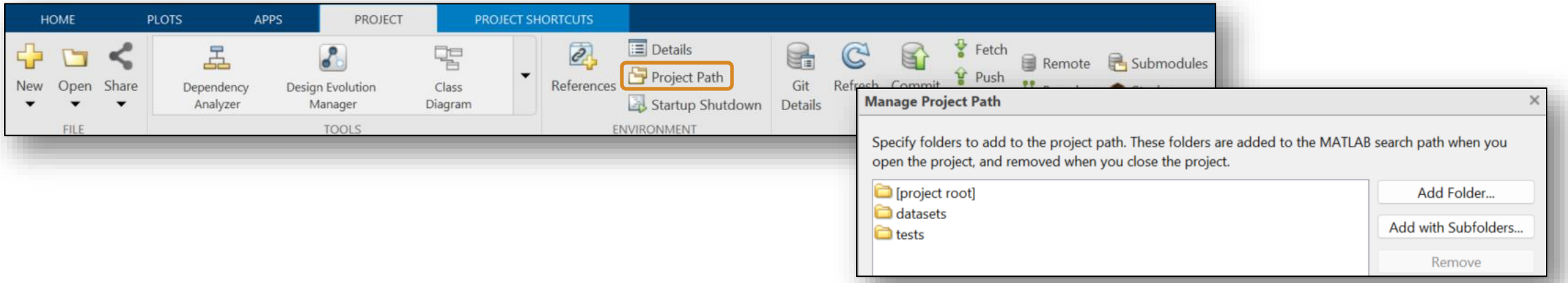
We can add any required folders to the project path, so any user can access the files within these folders without having to set up anything.



We can organize our work with MATLAB Projects so that it is easy to maintain and build on for us and others using our code …

A MATLAB Project manages all our MATLAB files, data files, etc. in one place:

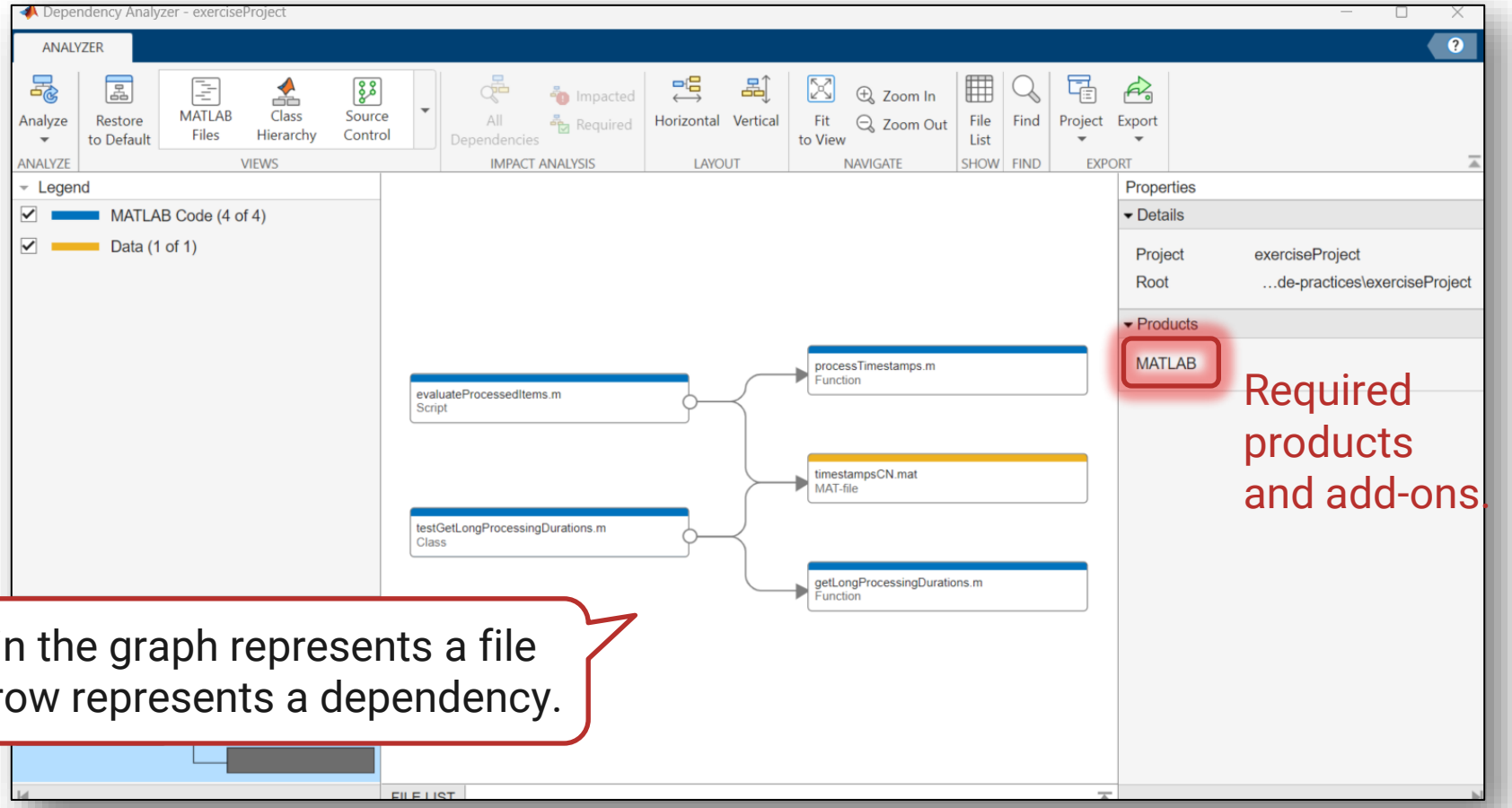
- All relevant project folders have already been added to the project path, so we and all users can access the files within them without having to set up anything. The project path is modifiable later on as well.



- MATLAB adds the folders to the search path whenever we open the project and removes them from the path when we close the project, so we‘re always good to go!

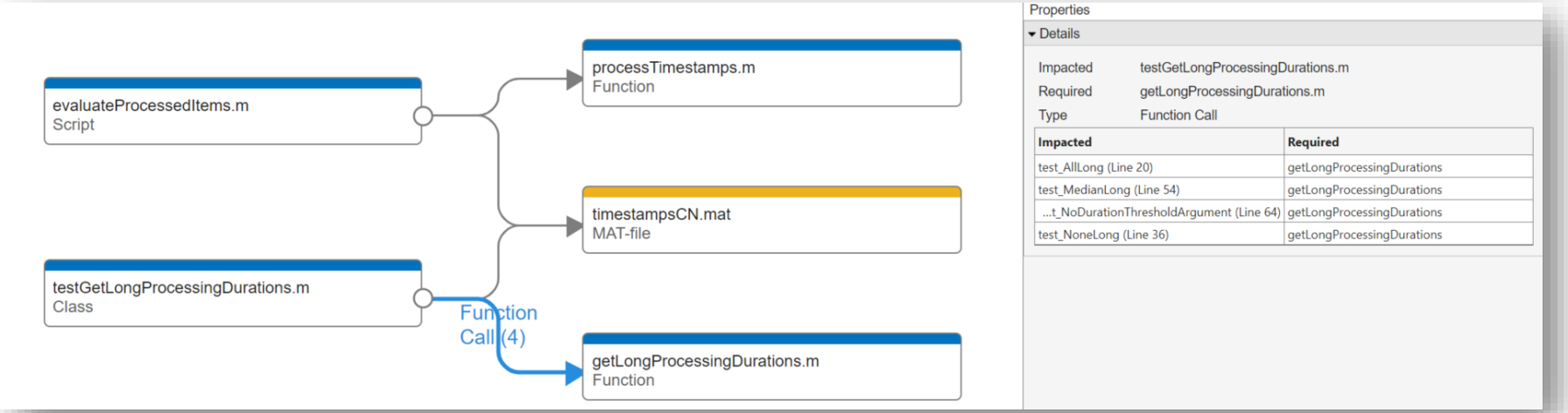
We can organize our work with MATLAB Projects so that it is easy to maintain and build on for us and others …

We can also use the **Dependency Analyzer** to see the project structure, dependencies, and how files relate to each other:



We can organize our work with MATLAB Projects so that it is easy to maintain and build on for us and others …

File dependencies help us understand better the impact of a code change and identify the tests we need to run to validate the change:



Select dependency arrow to see:

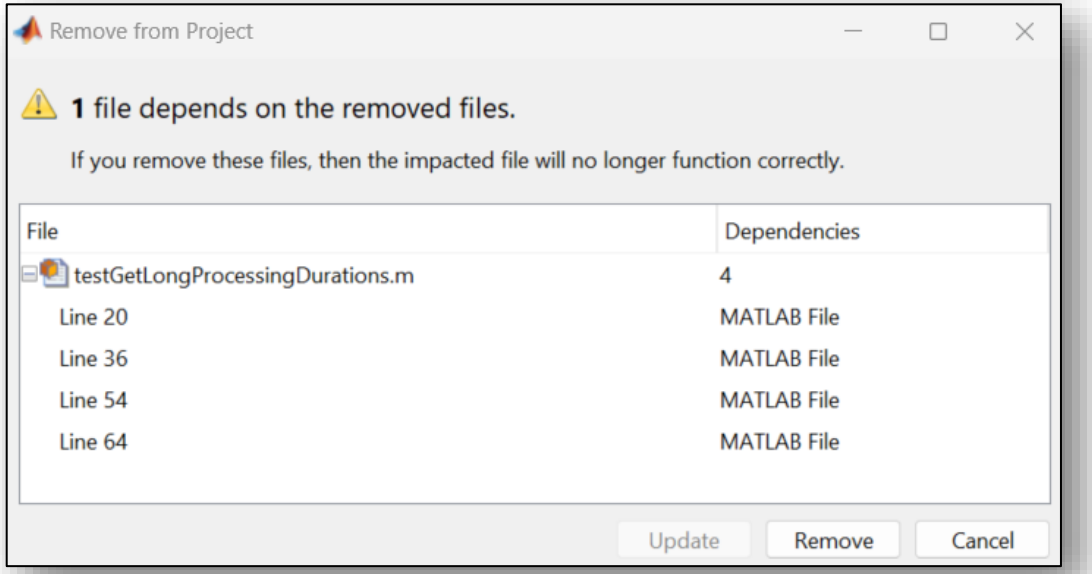
- dependency type (e.g., Function Call)
- where dependency is introduced (e.g., Line 20, etc.)

We can organize our work with MATLAB Projects so that it is easy to maintain and build on for us and others …

File dependencies help us understand better the impact of a code change and identify the tests we need to run to validate the change:

Say we refactor our code and think we can safely remove a file. When we **rename**, delete, or remove files or folders in a project, the project runs a dependency analysis to check for effects on other files:

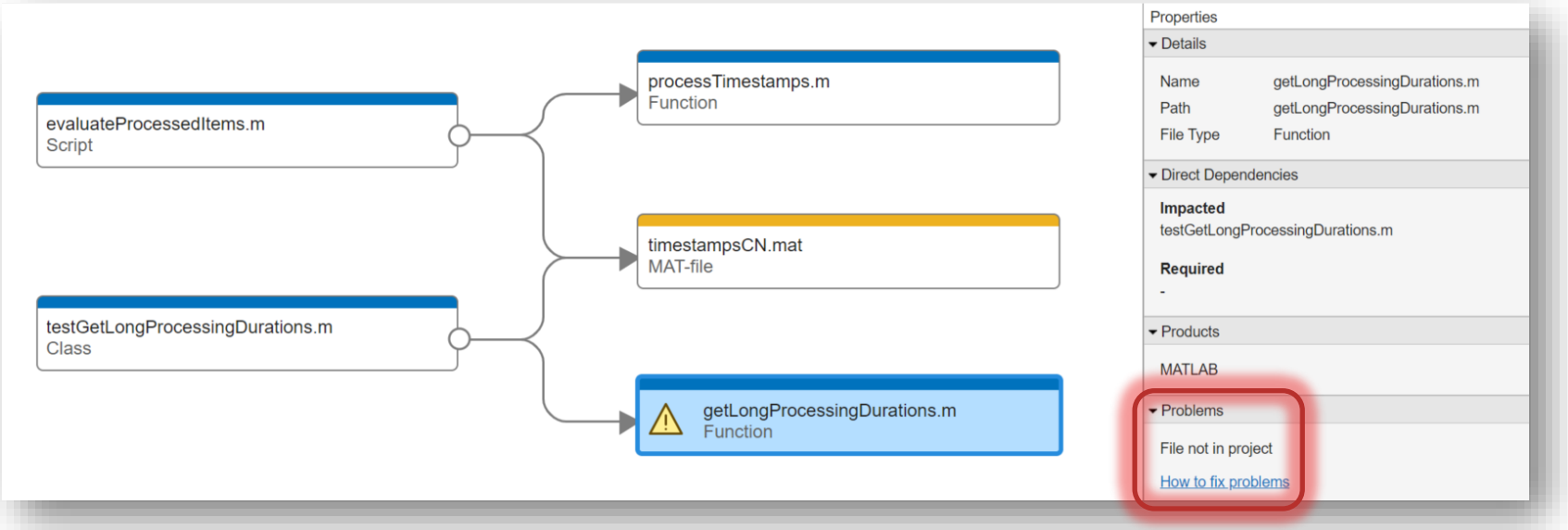
! *When **renaming** files, the project offers to automatically update references to the file, which prevents errors that result from changing names or paths manually and overlooking or mistyping the name.*



We can organize our work with MATLAB Projects so that it is easy to maintain and build on for us and others …

File dependencies help us understand better the impact of a code change and identify the tests we need to run to validate the change:

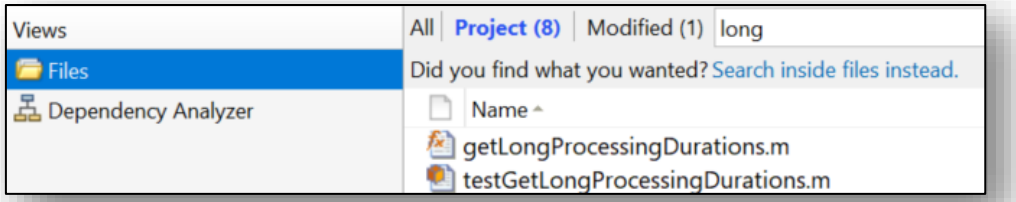
Say we refactor our code and remove a file (despite the warnings). The Dependency Analyzer automatically updates the graph and the **Problems** section:



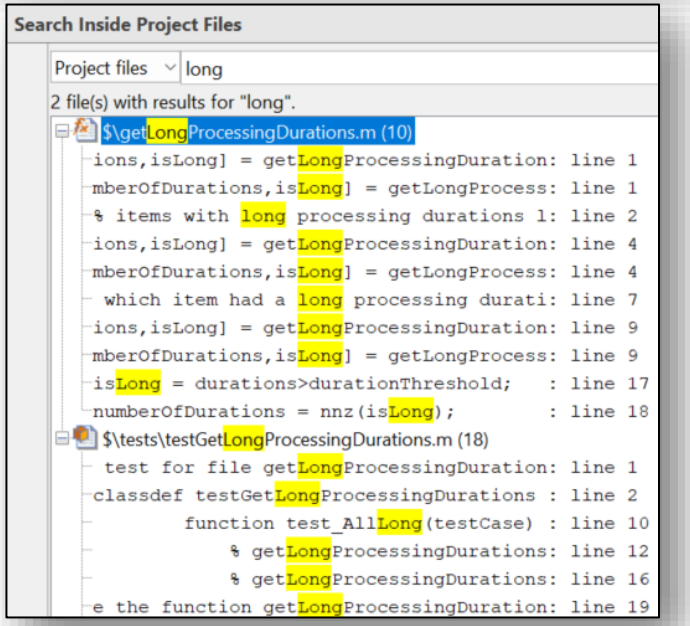
We can organize our work with MATLAB Projects so that it is easy to maintain and build on for us and others …

MATLAB Projects can help in many other ways:

- We can **search** for project files with a specific name:



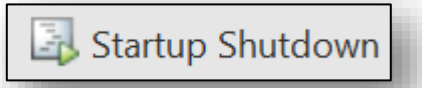
or content:



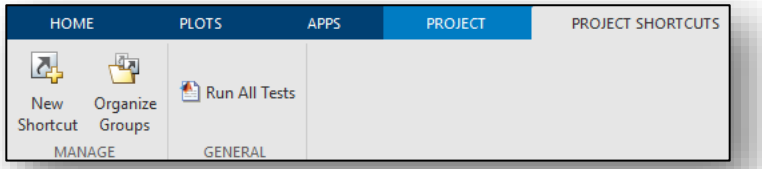
We can organize our work with MATLAB Projects so that it is easy to maintain and build on for us and others …

MATLAB Projects can help in many other ways:

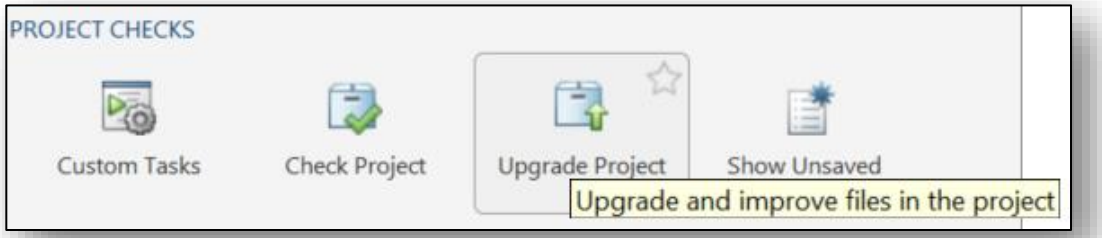
- We can specify **startup/shutdown** files to run when we open/close our project.



- We can use **shortcuts** to load data, run frequently used files/tests, etc.



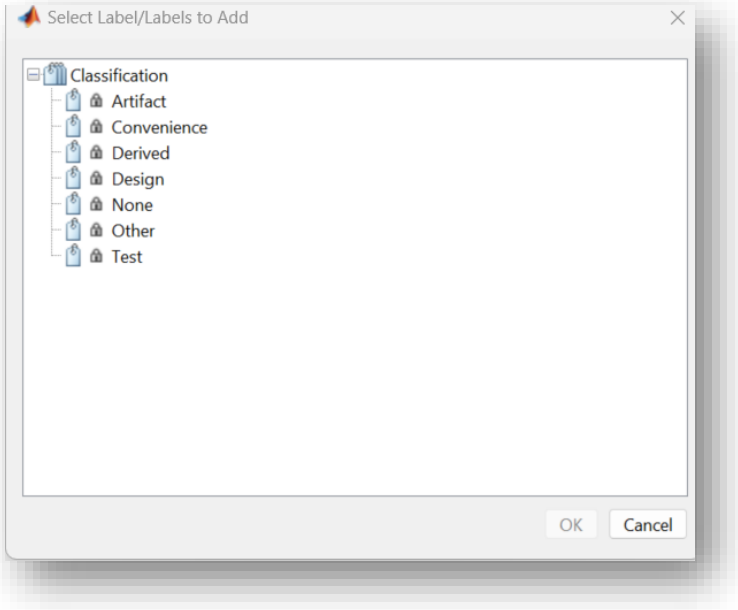
- We can **run checks**, e.g., to check for compatibility issues or upgrade our project to the current MATLAB release.



We can organize our work with MATLAB Projects so that it is easy to maintain and build on for us and others …

MATLAB Projects can help in many other ways:

- We can use one or more labels to organize project files and communicate information to project users:

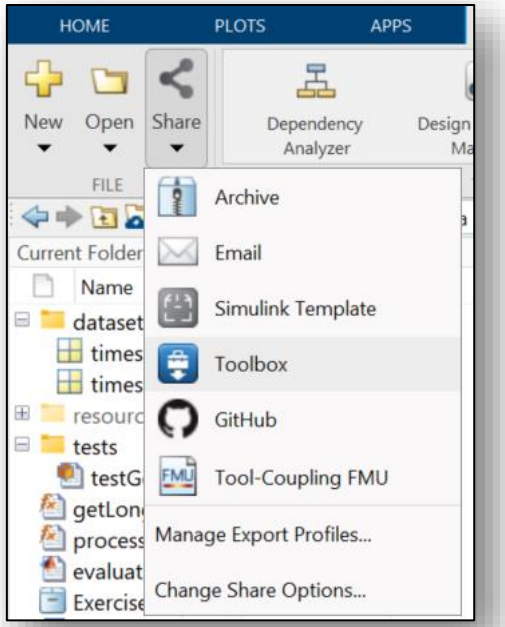


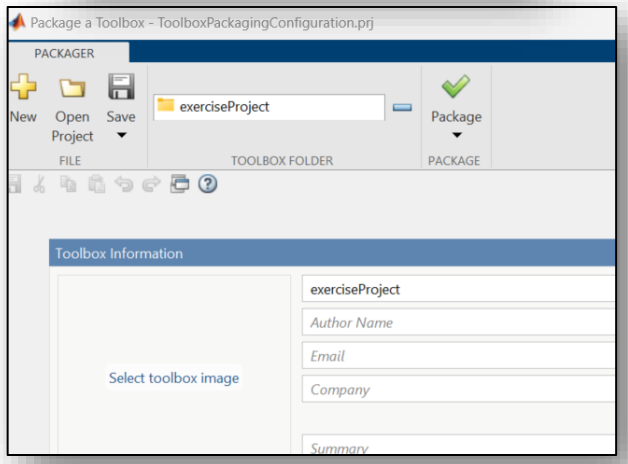
- Class-based unit tests are automatically associated with a Test label!

We can organize our work with MATLAB Projects so that it is easy to maintain and share with others …

When we are ready to share, we can use source control to collaborate, but there are also other ways:

- We can make our project publicly available on **GitHub**.
- Suppose we‘re sharing with somebody who doesn‘t have access to a source control tool, we can share our project as an **archive**.
- Or we can create a **toolbox** to share, this also puts everything into one file.





Your turn: Create a MATLAB project from an existing folder

- Go to your version of **CleanCodePractices>exerciseProjectDraft** and create a project from the folder.

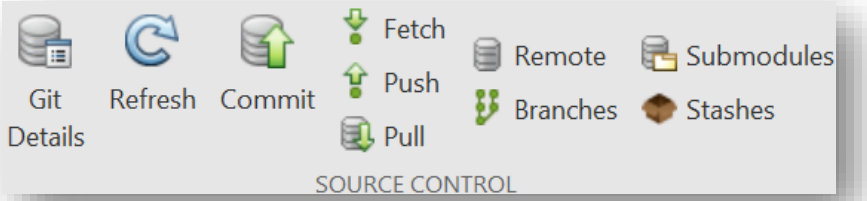
! *You can find the solution in* ***CleanCodePractices>exerciseProject***

Organizing and Collaborating – Version Control

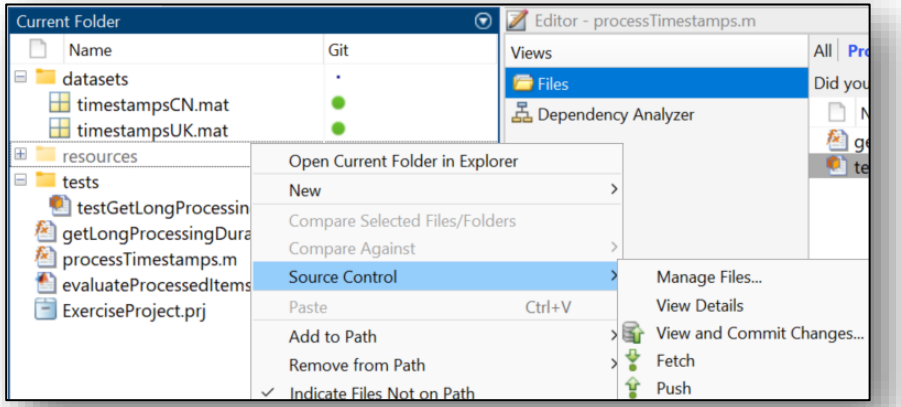
How do we keep track of our code as it changes?

We can use **source control** instead of maintaining several copies of our code versions:

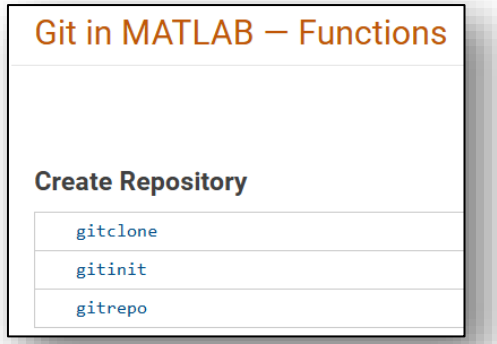
Use source control with Projects:



Use source control in Current Folder:



Use Git in MATLAB:

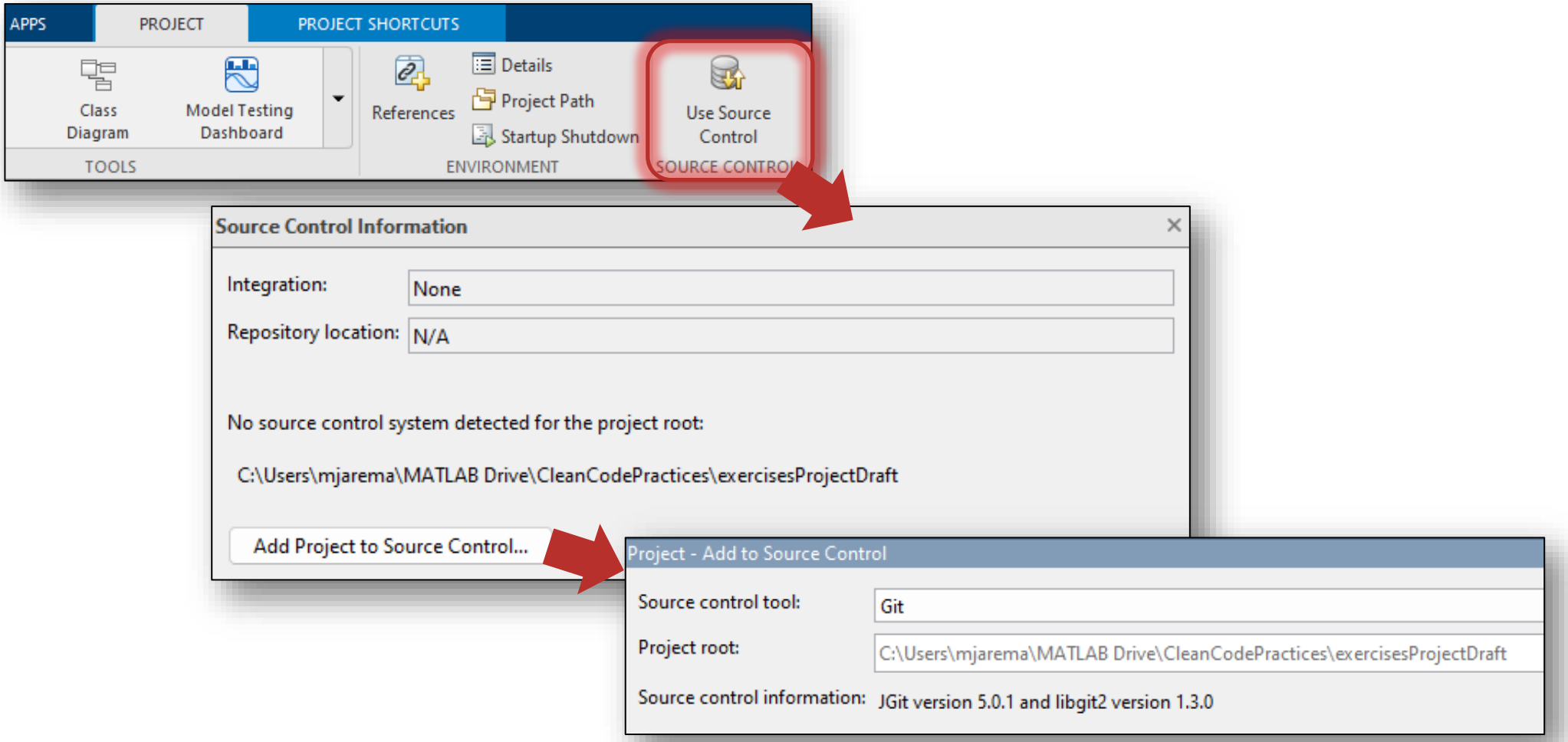


Use source control in Command Window:

```
>> !git status
On branch rse
Your branch is ahead of 'origin/rse' by 3 commits.
  (use "git push" to publish your local commits)

Changes to be committed:
```

How do we keep track of our project code as it changes?

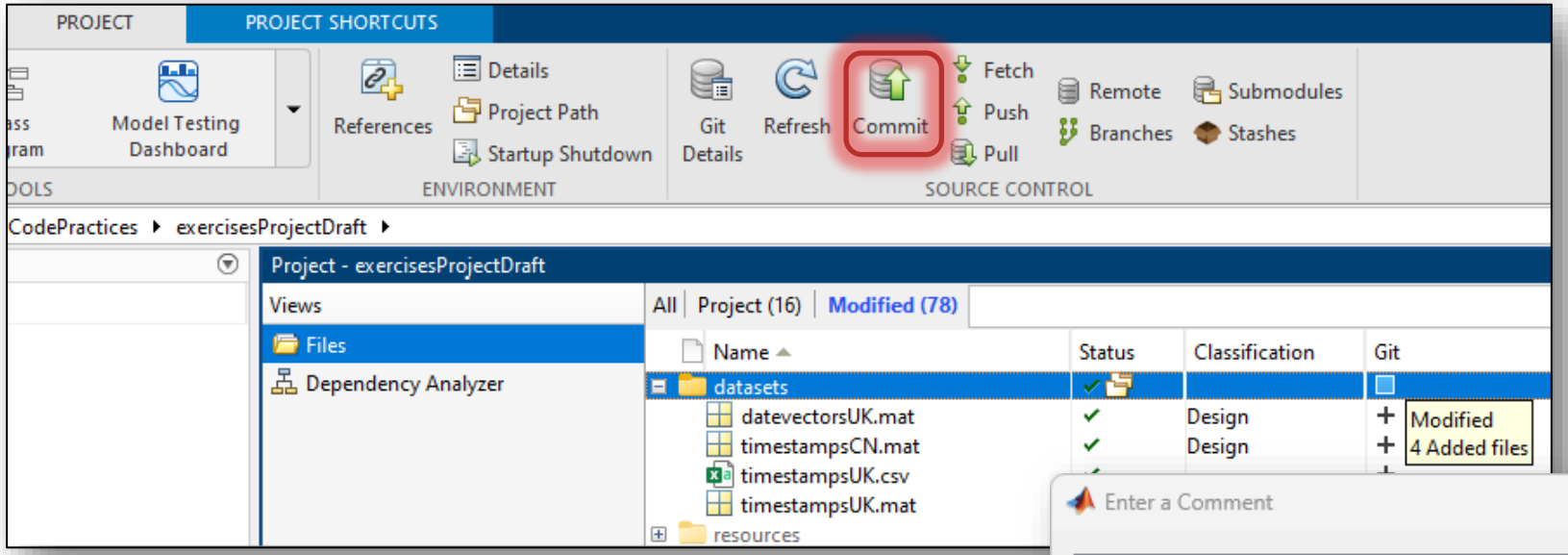
We can use projects to work with files under source control*:

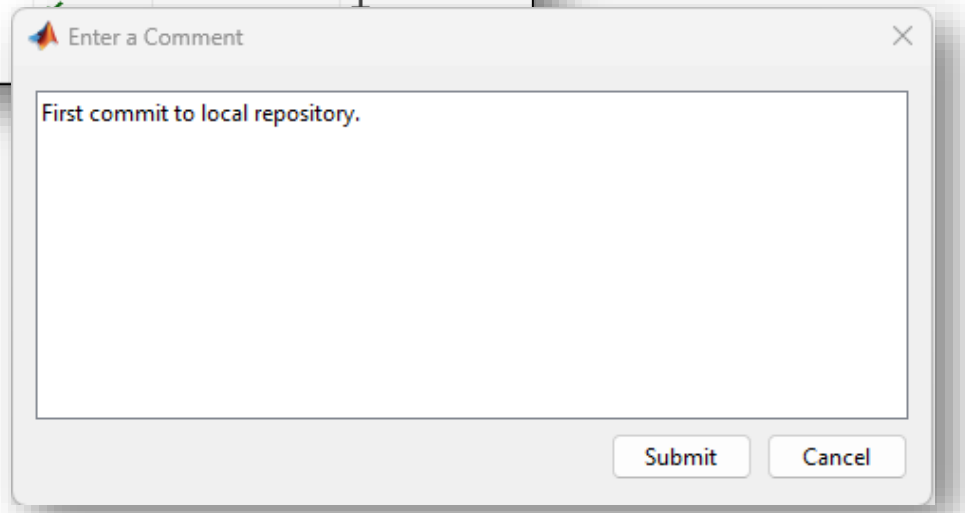


*In this case, **local** source control, ready to store **locally** a history of changes we make to our repository!

How do we keep track of our project code as it changes?

We can use projects to work with files under source control and perform source control operations, such as **committing*** added project files:





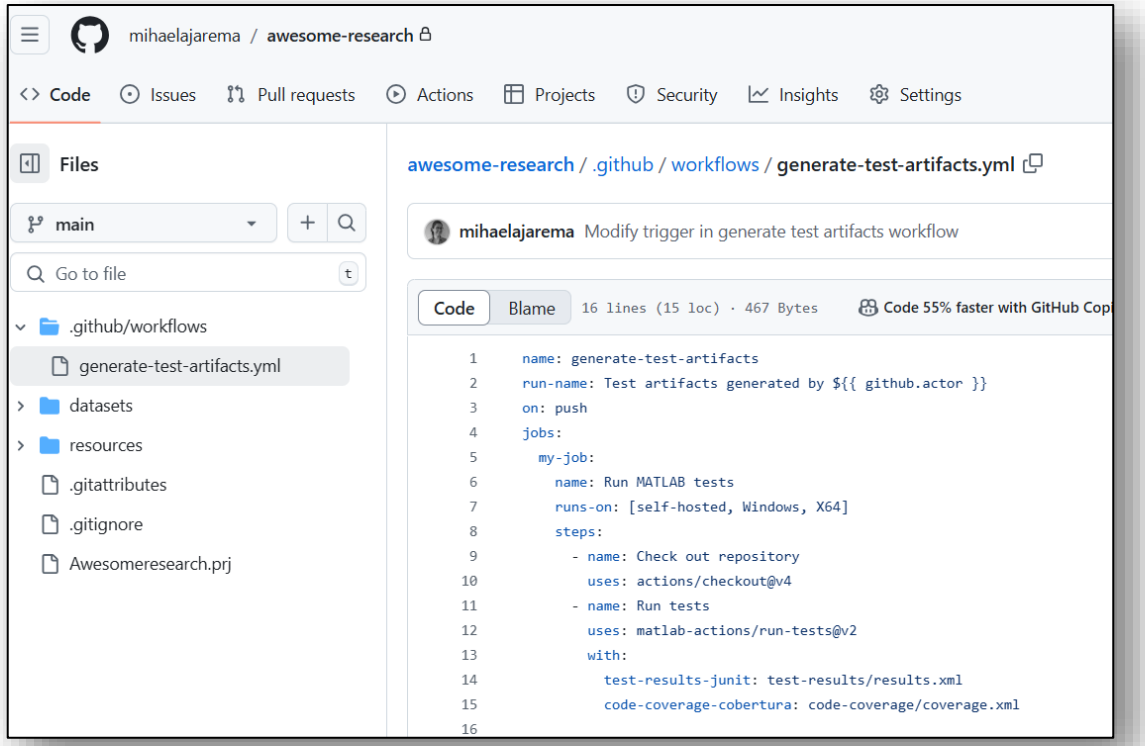
*Storing a current snapshot of our repository!

Your turn: Put your project under local source control

- Go to your version of **CleanCodePractices>exerciseProjectDraft** and put your project under local source control. If you have a GitHub account, you can share your project to GitHub instead.

Bonus: Use GitHub Actions to automate a build and test pipeline

- In your GitHub repository, create a .github/workflows directory and in it a YAML file to define a workflow that will be triggered by a push. The workflow should check out the repository and run tests using the Run MATLAB Tests action.



- If your repo is public and without transformation products, and you're using a self-hosted runner machine, you can do all this without needing to set up a license!

What‘s next?

So our code is more sustainable, what's next?

Clean code is easier to understand and **reuse**, provided others can find it and access it irrespective of their programming environments.

Findable

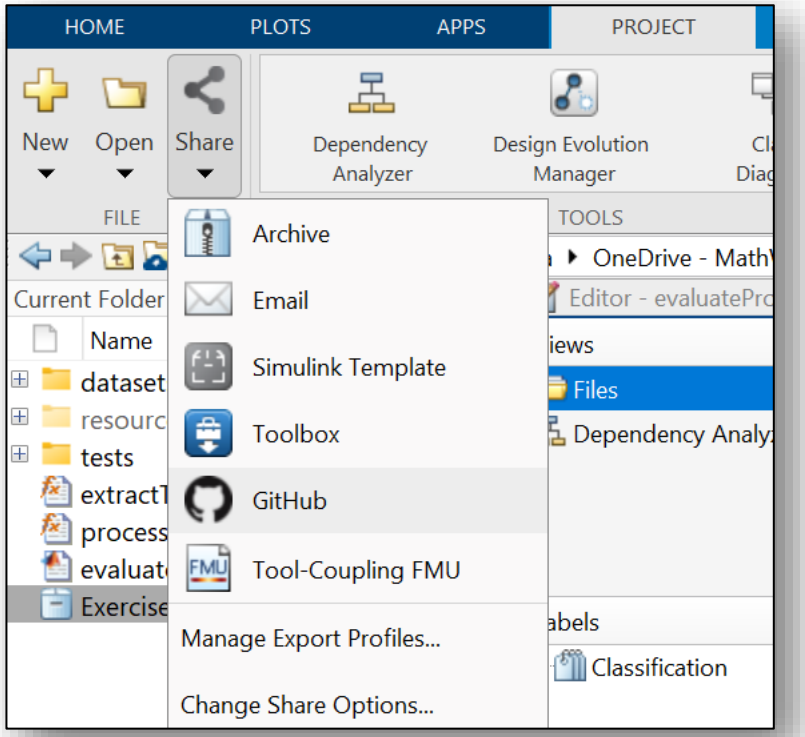
Accessible

Interoperable

Reusable

Findable – the quest for visibility

- **State:** Our Git repo is potentially local and thus not findable by others.
- **Actions to take:** Share our project by making it publicly available on GitHub*:



! *Tip: Make your code citable as well by securing a DOI for your repo, e.g., through Zenodo.*

**Requires a GitHub account!*

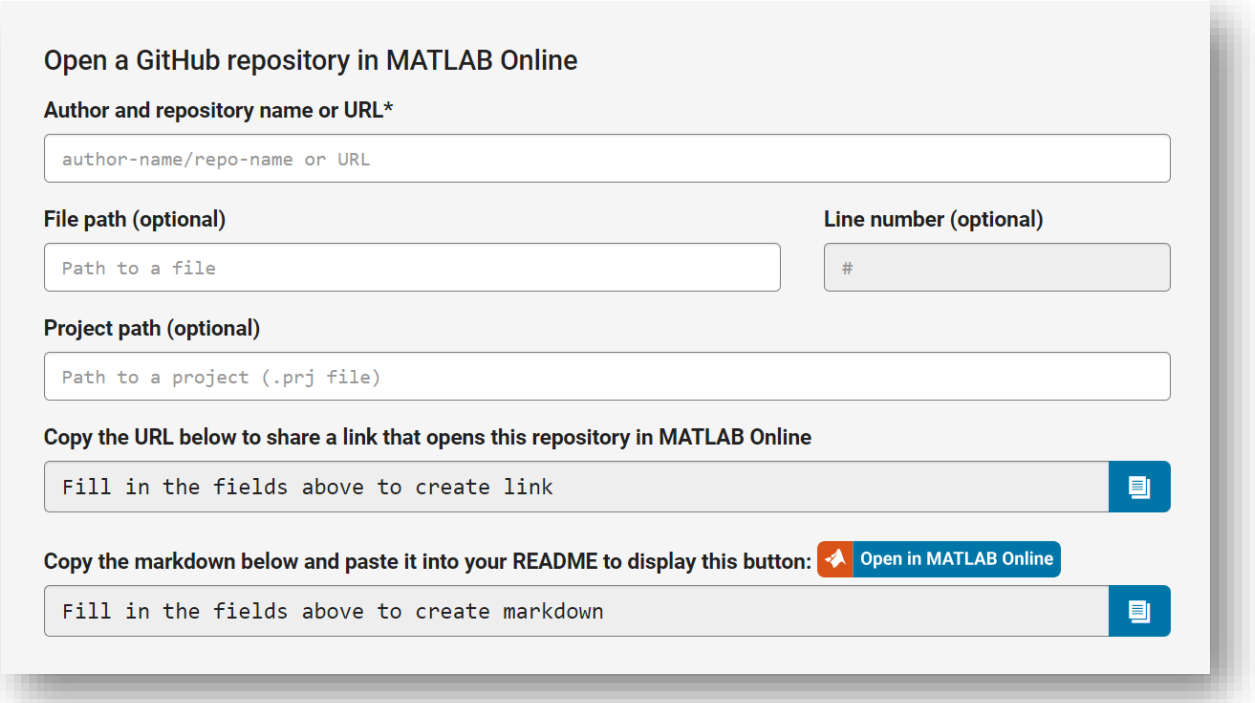
Findable – the quest for visibility

- **State:** Research software is available on public GitHub repo.
- **Actions to take:** Connect GitHub repo to MATLAB Central **File Exchange***
  - + No need to maintain code in two different locations.
  - + Community can discover & install code directly in MATLAB via Add-On Explorer.

*Global MATLAB user community to get & share free, open-source MATLAB code

Accessible – ensuring easy access

- **State:** Research software is available on public GitHub repo and linked to File Exchange.
- **Actions to take:** Create link to GitHub repo so ***anyone*** can open and view repo in MATLAB Online with just one click.



Access in MATLAB Online

	MATLAB Online (basic)	MATLAB Online
Cost	Free	License*
Version	Online only	Online and desktop
Products included	10 commonly used products Please see FAQ below	All products on your license available in MATLAB Online
Hours per month	20 hours per calendar month	Unlimited
Storage	5 GB	20 GB
Continuous compute time	15 minutes	90 minutes
Inactive idle timeout	15 minutes	60 minutes

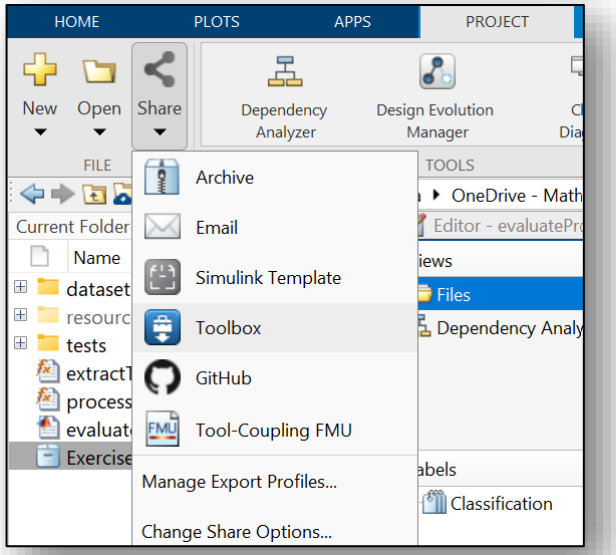
Interoperable – collaboration across formats & programming languages

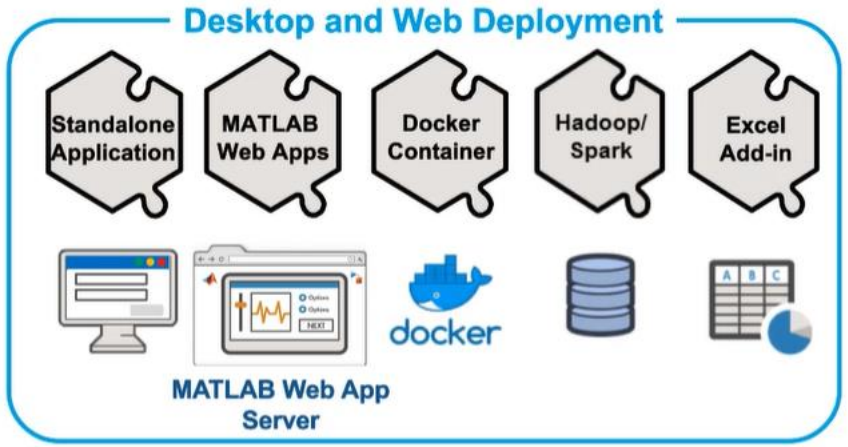
- **State:** Software is MATLAB-based, but we do leverage MATLAB's support for a wide range of standard file formats to ensure easy data import/export.

- **Actions to take:** Leverage MATLAB's interoperability with other programming languages to call, e.g., Python code for AI shared by our peers directly in our MATLAB code.

Reusable – How else can we improve impact and usability?

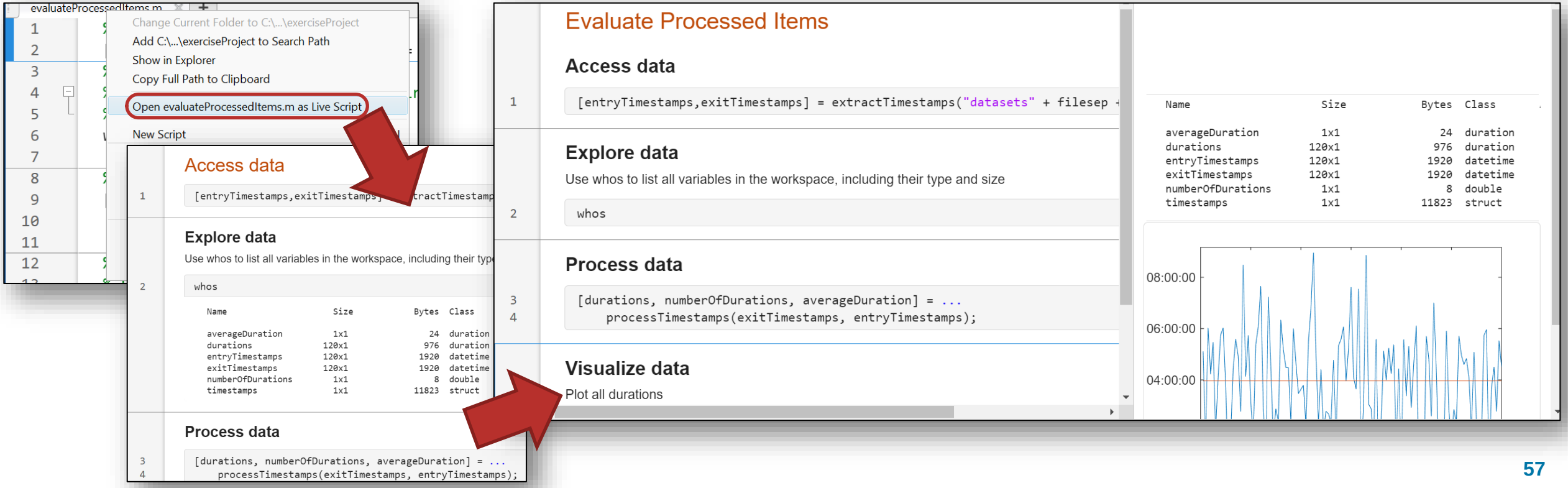
- Adopt appropriate open-source licensing for their MATLAB code, encouraging reuse and modification by others.
- Package code into a standalone application so ***any*** end user can run code royalty-free using MATLAB Runtime.





Reusable – How else can we improve impact and usability?

Enhance documentation using Live Scripts *(computational notebooks)* by creating interactive tutorials of typical workflows, and sharing them as PDF, Jupyter notebooks, etc. with non-MATLAB users.



Open call for working together to support your use cases

Reach out to:

Dr. Mihaela Jarema
Academia Group
mjarema@mathworks.com
https://www.linkedin.com/in/mihaelajarema/