MATLAB EXPO 2019

Software Development Practices within MATLAB

David Sampson
I want to help you...

Design  Architect  Implement  Share

...your MATLAB tools
What are your software development concerns?

- Accuracy
- Execution performance
- Development time
- Cost
- Compatibility
- Documentation
- Reusability
- Effective testing
- Integration
- Ease of collaboration
- Legacy code
- Liability
- Maintainability
- Model risk
- Robustness
- Developer expertise
- Software stack complexity
- …?
Agenda

- MATLAB Projects
- Version control integration
- Language features
- Development environment
- Testing & CI
- Toolbox distribution
- Design patterns
MATLAB Projects
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
Managing your code with Projects

1. Create project
Managing your code with Projects

1. Create project
2. Set path and startup tasks
Managing your code with Projects

1. Create project
2. Set path and startup tasks
3. Explore dependencies
Managing your code with Projects

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

Identify and run tests
Managing your code with Projects

1. Create project
2. Set path and startup tasks
3. Explore dependencies
4. Label files
Managing your code with Projects

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

- Maintain backups, history, and ability to restore
- Track changes and responsibility
- Simplify reconciling conflicting changes
- Generate discussion
- Save you from yourself
Version 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

MATLAB EXPO 2019
Implementation
Considerations when writing better, robust, and portable code

- Input validation
- Error handling
- Writing faster code using the MATLAB Profiler
- Writing code faster using the Live Editor
- Refactoring code to reduce complexity
- Writing code that works on all operating systems
Unactionable errors

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

Error in mypkg1.mypkg1a.mypkg1ab.mypfunc1 (line 9)
y(idx) = u(idx)*log(u_hat(idx))+(1-u(idx))*log(1-u_hat(idx));

Error in mypkg2.mypkg2a.mypfunc2 (line 5)
y = mypkg1.mypkg1a.mypkg1ab.mypfunc1( myVar1 .* myVar2 );

Error in mypkg3.mypkg3a.mypfunc3>@(x)mypkg2.mypkg2a.mypfunc2(x) (line 4)
y = arrayfun( @(x) mypkg2.mypkg2a.mypfunc2( x ), myVar );

Error in mypkg3.mypkg3a.mypfunc3 (line 4)
y = arrayfun( @(x) mypkg2.mypkg2a.mypfunc2( x ), myVar );

Error in myfunc (line 10)
Validating inputs

- validateattributes
- assert
- isempty, isnan, isfinite, ...
- narginchk
- inputParser
- Property validation for classes

```matlab
function y = myfunc( x )

% Validate inputs
validateattributes(x, 'double', {'size', [1 3], 'increasing'});

>> 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.
```

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

MATLAB R2019
Handling errors more elegantly

- error and warning
  - Use identifiers
- try/catch
- MException
- errordlg and warndlg
MATLAB Profiler

- Total number of function calls
- Time per function call
- Highlights largest code bottlenecks
- Statement coverage of code
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
Quickly and safely refactoring code

- Live Editor shortcuts to refactor blocks of code into functions
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

```matlab
>> 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 'emailsInFirstNameLastFormatParts' might be unused.
```

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

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

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

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

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

Windows:  
Mac/Linux: 
```
Code maintenance
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

Link to documentation for updates

Go directly to the line of code
Testing Frameworks

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

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

Test Pump Fault Model
This includes unit tests for the predictions

**Test: Model type**
Load the models and ensure they are the right types.

```matlab
load MLModels trainedModel
dmdl = 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.

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

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

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

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

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

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

testCase.type(myApp.editfield, myTextVar)
```
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
Continuous Integration workflow

- **Source Control**
  - Git
  - GitHub
  - GitLab
  - Subversion
  - …

- **Trigger**
  - Push
  - Merge Request
  - Pull Request
  - Check In
  - Scheduled
  - Manual

- **Build**
  - Run MATLAB / Simulink Tests
  - Run Performance Tests
  - Compile MEX Files
  - Generate Code*
  - Package Toolboxes
  - Build Components with MATLAB Compiler Stack*
  - Integrate with other software

- **Post Build**
  - Publish:
    - Test Results
    - Coverage Results
    - Performance Results
  - Accept Merge Request
  - Email Notification

* Transformation products may require Client Access Licensing
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
Jenkins plugin configuration

- Locate MATLAB
- Identify repository to load
- Set build triggers
- Add build step
Jenkins plugin reports

- View testing results
- View code coverage
- View testing reports
Sharing your code – The traditional way

- Unzip the zip file
- Find the instructions and release notes
- Decide whether you want the thing
- Remove folders from old versions from the path
- Add folders to the path
- Save the path for next time
- Find the documentation
- Do work
Sharing your code – How should you share code?

It depends on who you are sharing your code with:

- Co-authors → Project
- End-user with MATLAB → Toolbox or App
- End-user without MATLAB → Deployment (application, library, C code …)
Sharing your code with MATLAB users – Packaging your code

- Toolbox Packaging
- App Packaging

- Combine files into one installation file
- Installs in MATLAB Add-Ons or Apps tab
- Documents required products
Sharing your code outside of MATLAB – Application Deployment

Share your applications as:

- Standalone software
  - MATLAB Compiler
- Web applications
  - MATLAB Compiler
- Language-specific libraries
  - MATLAB Compiler SDK
- Generated code
  - MATLAB Coder
Design Patterns
Preface: handles and values

- MATLAB has both value and handle classes
- Everyday MATLAB datatypes exhibit value behaviour
- Handle classes facilitate multiple references to the same object
- MATLAB’s copy-on-write optimization limits memory consumption
- MATLAB’s reference counter disposes of unused handle objects

*Choose handle or value based on the need for multiple references.*
Design patterns

- Observer
- Adapter
- Singleton
- Builder
- Memento
Observer pattern

- When an object changes state, how can an arbitrary number of dependent objects react?
- How to avoid making the objects tightly coupled?

- Handle class for subject
- Event(s) on subject, possibly with custom event data
- Observers listen to events on subject

- Example: model with multiple views
Adapter pattern

- How can a class be reused that does not have an interface that a client requires?
- How can classes that have incompatible interfaces work together?
- How can an alternative interface be provided for a class?

- Private property to store an instance of the reused class
- Dependent properties to forward gets and sets
- Wrapper generator using meta.class APIs

- Examples: chart, modified timer, map services
Singleton pattern

- How can it be ensured that a class has only one instance?
- How can the sole instance of a class be accessed easily?
- How can a class control its instantiation?
- How can the number of instances of a class be restricted?

- Private constructor
- Private property to store the object
- `getInstance` static method

- Example: pointer manager
Builder pattern

- How can a class create different representations of a complex object?
- How can a class that includes creating a complex object be simplified?

- MATLAB handle class
  - `create*` method(s)

- Example: create unit from database
Memento pattern

- How can the internal state of an object be saved externally so that the object can be restored to this state later?
- Saving to and loading from disk or database is a common case

- `saveobj` instance method
- `loadobj` static method
- `ConstructOnLoad` class attribute

- Examples: renaming a class, removing a property
Closing remarks
Recap

- MATLAB Projects
- Version control integration
- Language features
- Development environment
- Testing & CI
- Toolbox distribution
- Design patterns
Thank you.

Questions?