Advanced Software Development with MATLAB

From research and prototype to production…
What Are Your Software Development Concerns?

- Accuracy
- Compatibility
- Cost
- Developer Expertise
- Development Time
- Documentation
- Ease of Collaboration
- Effective Testing
- Integration

- Legacy Code
- Liability
- Maintainability
- Model Risk
- Reusability
- Robustness
- Software Speed
- Software Stack Complexity
- …?
Development Process

**Research & Discovery**
- Explore and evaluate ideas
- Weigh alternatives, trade-offs

**Prototype to Production**
- Formalize design & testing
- Optimize performance

**Production Deployment**
- Migrate design to production
- Deploy / Integrate / Test
Case Study: Energy Load Forecasting System

- **Current State**
  - Software developed by ex-colleague
  - System entirely undocumented
  - Stability issues take a long time to debug
  - Days of downtime after security threat disrupted server

- **Future State**
  - Backward compatible for legacy applications
  - Reusable, extensible, and documented components
  - Testing and support for test environments
Agenda

- Architecture and Design
- Code Quality
  - Writing Code
  - Testing Code
- Integration
- Additional Resources
Progression of Programming Techniques

Level of Abstraction / Sophistication

Data
- value
- variable
- structure
- (properties)
- class
  - (methods)

Algorithm
- function
- script
- command line
Object Oriented Design

- Modularity
  - Data Access
  - Model Development
  - Charts
  - Data Cleaning
  - User Interface
  - Reporting

- Encapsulation
  - Price Series
  - Compute Metrics
  - Optimization Algorithm
  - Historical Statistics
  - Constraint Matrices
  - Constraints
  - Simulation
  - Asset Weights
  - Plot Frontier
  - Compute Risks
  - Portfolio
Object Oriented Class Hierarchy

- Reusable pieces through inheritance and composition
- Consistent interfaces
- Easily testable
- Source control

Developed by MathWorks Consulting Services
File Structure

- Namespaces (+packages)
- Separation of Tests from Source
- Source control managed
Interface Control Document

- Agree on interface between external software and production server
- Separate development of front end and back end
- Allows for trivial updating

Backward Compatible Syntax

```
[svg charstream fig1, svg charstream fig2] = visualizeDemandForecast2(char systemZone, bool forceUpdate, char deployType, numeric imageWidth, numeric imageHeight)
```

New System Updates

```
updateNYISOData()
updateWeatherData()
setPreferences(char prefName, variable prefVal,...)
```

New System Outputs

```
char reportFilename = generateForecastReport()
timetable forecast = generateDayAheadForecast(char zone)
```

```
public String getText(
    @DefaultValue("NYISO\F") @QueryParam("zone") String zone,
    @DefaultValue("false") @QueryParam("forceUpdate") boolean forceUpdate,
    @DefaultValue("svgstream") @QueryParam("imageType") String imageType,
    @DefaultValue("600") @QueryParam("imageWidth") int imageWidth,
    @DefaultValue("450") @QueryParam("imageHeight") int imageHeight,
    @DefaultValue("false") @QueryParam("escapeHtml") boolean escapeHtml,
    @DefaultValue("html") @QueryParam("outputType") String outputType){
    Object[] result = this.forecaster.visualizeDemandForecast2(zone, forceUpdate,
```
Source Control

- Current Folder, Simulink Projects, outside of MATLAB
- Text files, MAT files, live scripts, et al.
Requirements Management

- Link between source, test, and requirements
- Identify which requirements have been met
Agenda

- Architecture and Design
- Code Quality
  - Writing Code
  - Testing Code
- Integration
- Additional Resources
Flexible, Extensible Interfaces

- **inputParser**
  - Parameter / Value Pairs
- **Inheritance**

```matlab
function obj = RegressionComparer(x, y, varargin)
% obj = RegressionComparer(x, y, 'param', value)

narginchk(2, inf)

% Build obj using superclass.
obj = obj@LoadForecaster.Prediction.ModelComparer(x, y);

% Configure parser for defaults.
parser = inputParser;
addParameter(parser, 'ComparisonFunction', @r2, @(xx)validateatt);
addParameter(parser, 'MinimizeMaximize', 'Maximize', @(xx)validateatt);

% Parse and extract
parse(parser, varargin{:});
obj.ComparisonFunction = parser.Results.ComparisonFunction;
obj.MinimizeMaximize = parser.Results.MinimizeMaximize;
```

- **Variable number of inputs**
- **Reuse superclass constructor**
- **Can add more later**
- **Optional inputs with defaults and validation**
Exceptions, Errors, Warnings

- MException
- error
- assert
- warning
- try / catch
- onCleanup

% Try to build and read the datastore. If failing, throw exception.
try
ds = datastore(fullfile(tdir,'*.csv'));
data = readall(ds);
catch MEcaught
  MEthrow = MException('LoadForecaster:NYISOAPI:DatastoreFailure','Error reading input datastore
  MEthrow = addCause(MEthrow, MEcaught);
  throw(MEthrow);
end

% Check database open
function checkDBOpen(obj)
  if ~isopen(obj.DatabaseConnection)
    ME = MException('LoadForecaster:DataManagement:Database:NoConnection', ...
    'Database connection is closed');
    throwAsCaller(ME);
  end
end
Polymorphism & Weak Typing

- Attribute Validation
- Property Validation
- Access Restriction

```matlab
function obj = ModelComparer(x, y)
    % obj = ModelComparer(x, y)
    marginchk(2, 2)
    % Validate and set
    validateattributes(x, {'numeric'}, {'2d'}, mfilename, 'x', 1);
    validateattributes(y, {'numeric'}, {'column', 'nrows', size(x, 1)}, mfilename, 'y', 2);
    obj.X = x;
    obj.Y = y;
end
```

```
>> rc = RegressionComparer(rand(10, 3), rand(11, 1))
Error using ModelComparer
Expected input number 2, y, to be an array with number of rows equal to 10.
Error in LoadForecaster.Prediction.ModelComparer (line 31)
    validateattributes(y, {'numeric'}, {'column', 'nrows', size(x, 1)}, mfilename, 'y', 2);
Error in LoadForecaster.Prediction.RegressionComparer (line 27)
    obj = obj#LoadForecaster.Prediction.ModelComparer(x, y);
```

```matlab
properties
    Figure Figure handle
    Height (1,1) {mustBeNumeric, mustBePositive, mustBeInteger} = 300
    Width (1,1) {mustBeNumeric, mustBePositive, mustBeInteger} = 500
end
```
Preferences

- Add preferences to your application that persist to customize application
- Allows for configuration in compiled applications

```matlab
function set.ModelDirectory(obj, newmd)
    if exist(newmd, 'dir') || isempty(newmd)
        setpref('LoadForecaster', 'ModelDirectory', newmd);
        obj.ModelDirectory = getpref('LoadForecaster', 'ModelDirectory');
    else
        error('LoadForecaster:Interface:Preferences:NonexistentDirectory',
    end
end

function md = get.ModelDirectory(~)
    if ispref('LoadForecaster', 'ModelDirectory')
        md = getpref('LoadForecaster', 'ModelDirectory');
    else
        md = '';
    end
end
```
Debugging

- Conditional breakpoints stop when something happens
- Test fixes by running code while paused
- Code analyzer to find quick fixes
- Use unit tests to set up state
Execution Performance

- Profiler
- Performance testing framework
- Parallel computing
- Automatic C / C++ / CUDA Code generation
Documentation

- Functions, classes self document
- Full markup documentation that can be included in Help Browser

LoadForecaster.DataManagement.NYISOAPI

Connector to NYISO website

This class allows for easy downloading of historical zonal load data from NYISO's public repository. The data are updated every five minutes and live in zip files.

References: <http://mix.nyiso.com/public/>

Class Details

Superclasses: LoadForecaster.DataManagement.LiveDataSource
Sealed: false
Construct on load: false

Constructor Summary

NYISOAPI: Connector to NYISO website

Property Summary

Available/Variables: Which variables are present
Updateable: Can this data source be updated?

Method Summary

- protected getData: data = getData(source, 'Param', value)
- protected getDataImpl: Implement getData for NYISO API
- protected isConnected: Is connected to live data source?
- protected isConnectedImpl: Implement isConnected for NYISO API
- protected valid: Test handle validity.
- protected keepVars: Extracts requested variables.
- protected ofFutureDates: Short circuit if future only requested
- protected notify: Notify listeners of event.
- protected parseInputs: Parser
- protected migrateData: Get rid of memory on disk
- protected vertcat: Vertical concatenation for heterogeneous arrays
- protected zipTable: Downloads zip files from NYISO and builds a table
idxGroupedByLevel = {};  
done = false;  
findHole = false;  % start with an object boundary  
while ~done  
  if (findHole)  
    I = FindOutermostBoundaries(holes);  
    holes = holes(~I);  % remove processed boundaries  
    idxGroupedByLevel = [idxGroupedByLevel, {holeIdx(I)}];  
    holeIdx = holeIdx(~I);  % remove boundaries  
  else  
    I = FindOutermostBoundaries(objs);  
    idxGroupedByLevel = [idxGroupedByLevel, {objIdx(I)}];  
  end  
  if (processHoles)  
    findHole = ~findHole;  
  end  
  if ( isempty(holes) && isempty(objs) )  
    done = true;  
  end  
end
Code Complexity

- checkcode -cyc
- Include test cases for complexity

```
Command Window
>> checkcode -cyc LoadForecaster.DataManagement.NYISOAPI
L 14 (C 26-36): The McCabe complexity of 'getDataImpl' is 3.
L 52 (C 25-39): The McCabe complexity of 'isConnectedImpl' is 1.
L 58 (C 25-33): The McCabe complexity of 'zip2table' is 4.

% Fail test if >= 10;
testCase.verifyLessThan(cycj, 10, sprintf('McCabe Complexity of %i\nIn: %s', cycj, filename));
```
Why use Automated Testing?

- Improve quality
- Understand and document code
- Reduce and not introduce risk
- Catch bugs early and often

Four Phase Test

1. Create test array
2. Setup file fixtures
3. Loop over all test functions
   - Setup fresh fixtures
   - Run test
   - Teardown fresh fixtures
4. Teardown file fixtures
5. End
Authoring Tests

- Script, function, and class based tests
- Setup and teardown for four phase testing
- Rich qualifications and diagnostics
- Parameterization to build combinations of inputs

```matlab
methods (TestMethodSetup)
%Run before each test method
function makeFigure(testCase)
    testCase.Figure = figure;
    plot(sin(1:100));
    xlabel('X')
    ylabel('Hello World','FontAngle','italic')
    patch([0 10 10 0],[0 0 1 1],'b','FaceAlpha',0.3)
end
end

methods (TestMethodTeardown)
%Run after each test method
function closeFigure(testCase)
    delete(testCase.Figure)
end
end
```

<table>
<thead>
<tr>
<th>Type of Test</th>
<th>Verification</th>
<th>Assumption</th>
<th>Assertion</th>
<th>Fatal Assertion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value is true</td>
<td>verifyTrue</td>
<td>assumeTrue</td>
<td>assertTrue</td>
<td>fatalAssertTrue</td>
</tr>
<tr>
<td>Value is false</td>
<td>verifyFalse</td>
<td>assumeFalse</td>
<td>assertFalse</td>
<td>fatalAssertFalse</td>
</tr>
<tr>
<td>Value is equal to specified value</td>
<td>verifyEqual</td>
<td>assumeEqual</td>
<td>assertEqual</td>
<td>fatalAssertEqual</td>
</tr>
<tr>
<td>Value is not equal to specified value</td>
<td>verifyNotEqual</td>
<td>assumeNotEqual</td>
<td>assertNotEqual</td>
<td>fatalAssertNotEqual</td>
</tr>
<tr>
<td>Two values are handles to same instance</td>
<td>verifySameHandle</td>
<td>assumeSameHandle</td>
<td>assertSameHandle</td>
<td>fatalAssertSameHandle</td>
</tr>
<tr>
<td>Value is not handled</td>
<td>verifyNotSameHandle</td>
<td>assumeNotSameHandle</td>
<td>assertNotSameHandle</td>
<td>fatalAssertNotSameHandle</td>
</tr>
</tbody>
</table>
Mocking

- Framework to emulate software or hardware that is inaccessible or slow
- Spy on mock usage for testing

```matlab
% Mock the database and the cursor
[dbmock, dbbehavior] = createMock(testCase, 'AddedMethods', ... ['isa', 'isopen', 'exec']);
[cursmock, cursbehavior] = createMock(testCase, 'AddedMethods', ... ['close', 'fetch'], 'AddedProperties', {'Data'});

% Behavior of cursor and db connection
when(withAnyInputs(dbbehavior.isa), AssignOutputs('database.jdbc.connection'));
when(withAnyInputs(dbbehavior.isopen), AssignOutputs(true));
when(withAnyInputs(dbbehavior.exec), AssignOutputs(cursmock));
when(withAnyInputs(cursbehavior.fetch), AssignOutputs(cursmock));

% Add All Data
addData(db, data);
testCase.verifyThat(withAnyInputs(dbbehavior.insert()), WasCalled('WithCount', 2));
testCase.verifyThat(withAnyInputs(dbbehavior.update()), WasCalled('WithCount', 2));
firstlut = db.LastUpdateTime;
nowish = datetime('now', 'TimeZone', 'America/New_York')+seconds(1);
testCase.verifyLessThan(table2array(firstlut), nowish);
```
Testing Infrastructure

- Run suites of tests from many files, directories, and packages
- Select tests based on tags or parameters
- Shared fixtures (e.g. path management, temporary folders)

```matlab
import matlab.unittest/fixtures.TemporaryFolderFixture
tdir = testCase.applyFixture(TemporaryFolderFixture);
```

```matlab
classdef TestType TestParameter
    properties
        RasterImageTypes = {'bmp', 'png', 'jpg', 'gif'} % Different raster image types
        StreamOutputTypes = {'uint8', 'int8'} % Different byte stream data types
    end
end
```

Command Window

```
>> TS = TestSuite.fromPackage('Test.Visualization', 'IncludingSubPackages', true)
TS =
   1x34 Test array with properties:
   Name
   BaseFolder
   ProcedureName
   SharedTestFixture
   Parameterization
   Tags
Tests Include:
   6 Unique Parameterizations, 0 Shared Test Fixture Classes, 0 Tags.
```
Test Outputs

- Coverage report

![Coverage report](image.png)
Test Outputs

- Test results report

MATLAB® Test Report

- Timestamp: 22-Sep-2017 12:16:21
- Host: AH-SDEWOLSK
- Platform: win64
- MATLAB Version: 9.3.0.701794 (R2017b)

- Number of Tests: 9
- Testing Time: 4.7620 seconds
- Overall Result: FAILED

---

Failure Summary

3 tests failed.

<table>
<thead>
<tr>
<th>Name of Failing Test</th>
<th>Failure Reasons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test.DataManagement.TestDatabase/testAllNansInDatSynchronize</td>
<td>Failed by verification.</td>
</tr>
<tr>
<td>Test.DataManagement.TestDatabase/testGetDataOneSource</td>
<td>Failed by verification.</td>
</tr>
</tbody>
</table>

Overview

C:\Documents\MATLAB\DataAnalytics\LoadForecastingApplication\Source\Test\DataManagement\TestDatabase.m

TestClosedConnection

- The test passed.
- Duration: 0.0932 seconds

TestGetData

- The test failed.
- Duration: 2.1368 seconds

Event:

- Verification failed.

Framework Diagnostic:

verifyEqual failed.

- The objects are not equal using "isequal".

- Actual Value:

  - you are empty timetable
  - Expected Value:

  - your timetable

  - Time
  - Price
  - Temperature\_MAX
  - Dewpoint\_MIN

  - 01-May-2007 01:00
    - 40.0
    - 18.0
    - 9.9
  - 01-May-2007 01:10
    - 40.0
    - 18.0
    - 9.9
  - 01-May-2007 01:15
    - 47.0
    - 35.0
    - 22.9

- Event Location: Test.DataManagement.TestDatabase/testGetData

Stack:

- C:\Documents\MATLAB\DataAnalytics\LoadForecastingApplication\Source\Test\DataManagement\TestDatabase.m
- TestDatabase.testGetData at 99
Continuous Integration

- Runs tests after code changes to track development progress
- Support for TAP, jUnit XML
Agenda

- Architecture and Design
- Code Quality
  - Writing Code
  - Testing Code
- Integration
- Additional Resources
Integrating with Other Languages, Frameworks, Databases

- Calling Out
  - C / C++
  - Python
  - .NET
  - REST / HTTP
  - WSDL / SOAP
  - Java
  - System
  - FORTRAN
  - Perl
  - Verilog / VHDL (HDL Verifier)
  - Databases ODBC / JDBC / Some NoSQL (Database Toolbox)
  - OPC / Historian (OPC Toolbox)
  - CUDA (Parallel Computing Toolbox or GPU Coder)

- Calling In
  - C / C++
  - Python
  - .NET
  - Java
  - Excel
  - REST (MATLAB Production Server)
  - FORTRAN
  - System
  - COM
Getting out of MATLAB

Embedded Hardware

C, C++, HDL, PLC, CUDA

Enterprise Systems

Standalone Application, Excel Add-in, Hadoop/Spark, C/C++, Java, Python, .NET, MATLAB Production Server

MATLAB Runtime

MATLAB Runtime
What is Production Server?

- Enterprise class framework for running packaged MATLAB programs
- Server software
  - Manages packaged MATLAB programs and worker pool
  - Single server can use runtimes from different releases
- Lightweight client libraries for: C/C++, .NET, Python, Java, and RESTful JSON interface
MATLAB Production Server Workflow

**Development**

- Initial Test Application
- Debug Algorithm
- MATLAB Algorithm
- MATLAB Compiler SDK
- Deployable Archive (CTF)
- Verify data handling and initial behavior

**Production**

- Web Application
- Function Calls
- MATLAB Production Server
Integration of MATLAB Based Analytics with your IT infrastructure

Data

- Databases: Cassandra, SQL Server, MongoDB
- Cloud Storage: Azure Blob, Azure SQL, S3
- IoT & Big Data: Kafka, Azure IoT Hub, Cloudera

Business System

- Visualization: Spotfire, QlikView, Microsoft Power BI
- Web: Microsoft IIS, WebSphere
- Custom App: Apache Tomcat

Platform

- Public Cloud: Microsoft Azure, Amazon Web Services, Rackspace
- Private Cloud: OpenStack, VMware
Agenda

- Architecture and Design
- Code Quality
  - Writing Code
  - Testing Code
- Integration
- Additional Resources
MATLAB Central Community

Every month, over 2 million MATLAB & Simulink users visit MATLAB Central to get questions answered, download code and improve programming skills.

MATLAB Answers: Q&A forum; most questions get answered in only 60 minutes

File Exchange: Download code from a huge repository of free code including tens of thousands of open source community files

Cody: Sharpen programming skills while having fun

Blogs: Get the inside view from Engineers who build and support MATLAB & Simulink

ThingSpeak: Explore IoT Data

And more for you to explore…
Training: MATLAB Programming Techniques

After this 2-day course you will be able to:

- Ensure code correctness.
- Create robust and user-friendly application interfaces.
- Optimize performance by choosing appropriate function and data types.
- Create custom toolboxes.
Commerzbank Develops Production Software System for Calculating Derived Market Data

Challenge
Compute a variety of derived market data from raw market data

Solution
Used MATLAB to read data from a data management system in a Windows and Linux architecture, perform analyses and optimizations, visualize results, and deploy mission-critical calculations. Working with MathWorks consultants, Commerzbank business analysts developed a proof-of-concept implementation of the cross-platform architecture.

Results
- Integration with existing system simplified
- Implementation time reduced by months
- Updates made in days, not weeks

“Our solution required a Windows client and Linux server software. We used MATLAB to rapidly develop both by taking advantage of distributed computing, a MEX-file interface to access our financial data, and fast, built-in functions for optimization, regression, and more.”

Julian Zenglein
Commerzbank
Want to Learn More?

Engineers

Sean de Wolski
Senior Application Engineer
Sean.deWolski@mathworks.com

Marshall Alphonso
Senior Finance Engineer, New York
Marshall.Alphonso@mathworks.com

Account Managers

Chuck Castricone
Account Manager
Chuck.Castricone@mathworks.com

Mike DeLucia
Senior Account Manager
Mike.DeLucia@mathworks.com