Intro to Coding Best Practices for Shared Projects

Ing. John LaMaster

Technical University of Munich

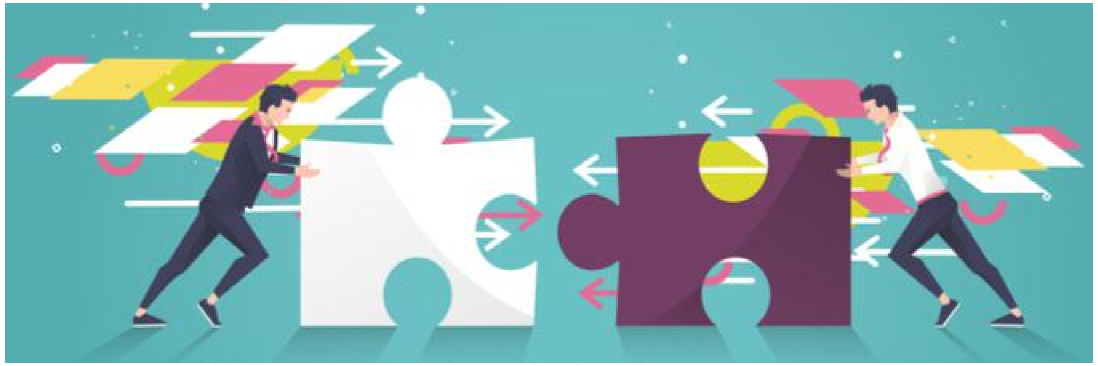
MRS Hackathon, Toronto

June 02, 2023

Focus

Coding *best practices* for shared projects

- Collaborations
- Theses
- Projects to be handed over to someone else



https://www.apm.org.uk/v2/media/vjufacmf/shutterstock603988838.jpg

Coding

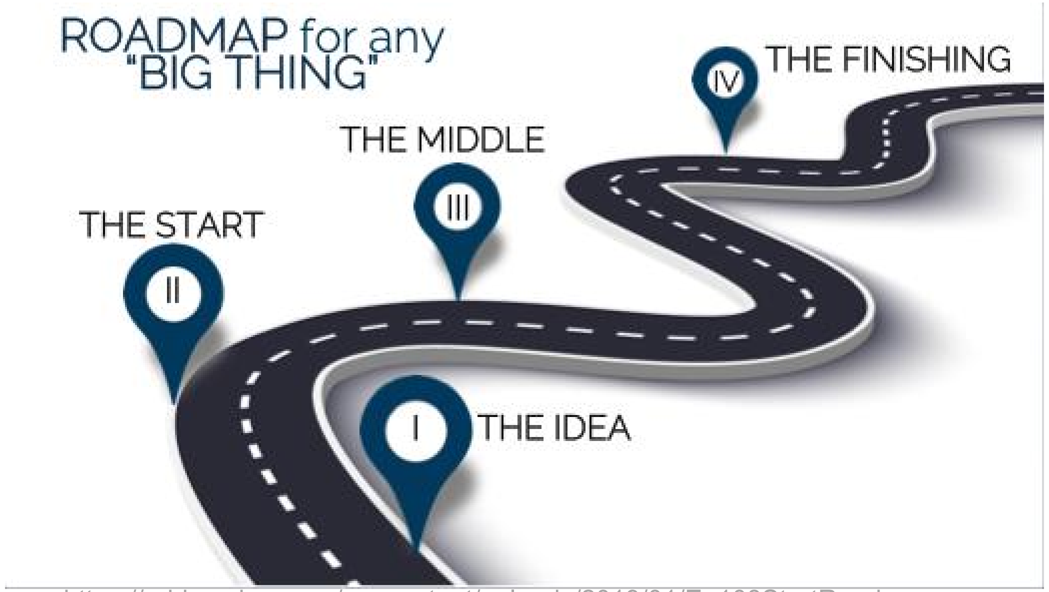
- Anyone can write working code. Making it work is easy.
- Most people write "*working code*", not "*good code*"
- *Good code* should be:
  - **Organized**
  - Reliable
  - **Maintainable**
  - Efficient
  - **Documented**

Organized

Getting Started with Big Projects

Plan ahead!

- Objectives
- Functionality
- Requirements
- Completion checklist
- What will happen with the code?



https://addcrusher.com/wp-content/uploads/2019/01/Ep108StartRoadmap.png

Example: Data Simulator

Starting Point

Goal:

1. Robust physics model
2. Include everything possible

“I'll figure it out as I go...”

Better Developed

Goal:

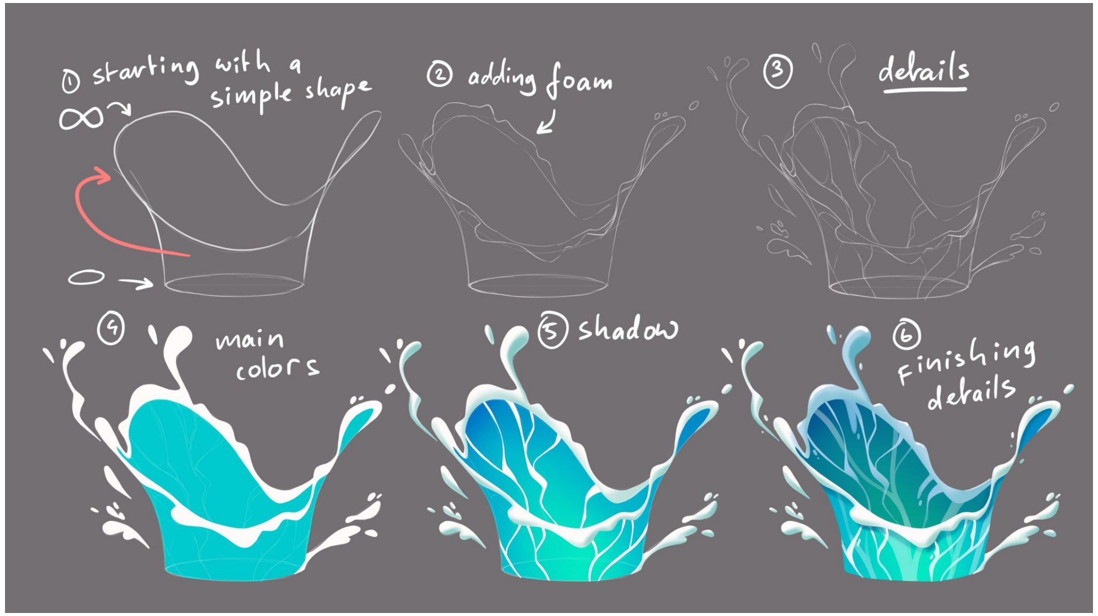
1. Maximize the degrees of freedom
2. Include all clinically relevant artifacts
   - List them out
3. Simulations should match specific use case(s)
   - What does this mean specifically?
4. Framework and compute requirements?
5. Community-driven development

Organized

Hierarchy of complexity!

Always start simple!

- Start with the basics
- Iteratively debug
- Add more complexity and repeat



https://pbs.twimg.com/media/FtRQeq_XoAAz7b6.jpg:large

Organized

Tips for staying organized...

- Layout your project
  - A good layout should work for multiple (similar) projects
- Use the folders!
  - Save things where they belong
- Use version control to track and update your changes
  - Very useful for when you break something!
- Save experiment descriptions
  - E.g. ablation_exp_001_config.json

```
dl_project
    datasets
    images
    manuscript
    tests
    scripts
        ablation_study
        HPO
        final
    src
        dataloaders
        models
        modules
            attention
            aux
            architectures
            dropout
            metrics
        options
        trainers
        util
```

Reliability

"the quality of being trustworthy or of performing consistently well."

- Oxford Dictionary

What does this look like in practice? – *Defensive programming*

- Assert statements and unit tests
  - Validate inputs
  - Boundary checks
- Error handling
- When are they necessary?

Reliability

***Example:** Fourier Transform*

Expectations:

- Input is at least a 3D matrix
  - n-dimensional tensor of 1D data
- The last dimension needs size=2
  - input data type is complex
  - Typical inputs should need to be transposed to satisfy this condition
- *torch.view_as_complex* expects contiguous inputs
- Separate outputs' real and imaginary in second to last dimension
- Output should be contiguous

Expected input:

signal.shape = [batchSize, ..., channels=2, spectral_length]

```
def Fourier_Transform(signal: torch.Tensor) -> torch.Tensor:
    assert(signal.ndim>=3)
    if signal.shape[-2]==2:
        signal = signal.transpose(-1,-2)
    assert(signal.shape[-1]==2)
    signal = torch.view_as_complex(signal.contiguous())
    signal = torch.view_as_real(fftshift(fft(signal, dim=-1),
                                         dim=-1)).transpose(-1,-2)
    return signal.contiguous()
```

Maintainability

The ability to update and improve a code base and to fix problems when they arise.

To maintain code, one needs to be able to:

- Read the code,
- Understand the code, and
- Fix the code.

Maintainability

"Readability improves understanding; that further improves maintainability"

-Cynthia Nelson

Scenario: 10 colleagues each use their own coding conventions. How does that affect code readability and maintainability?

Maintainability

Coding conventions

For your own projects,

- **Always be consistent**
- Follow the conventions for your selected framework

For collaborations,

- Define (non-standard) conventions at the beginning
- **Always be consistent**
- Limit exceptions

Maintainability

Naming conventions

- **Camel case**
  - Ex. firstName and lastName
- **Snake case**
  - Ex. first_name and last_name
- **Kebab case**
  - Ex. first-name and last-name
- **Pascal case**
  - Ex. FirstName and LastName

Python:

- variable & function names use Snake case
- classes use Pascal case

JavaScript:

- variable & function names use Camel case
- classes use Pascal case

Maintainability

Formatting conventions

Spacing

```
names = ['d',   'dmm', 'g',   'gmm']
mult  = [  1, self.MM,   1, self.MM]
------------------------------------
dB0   = x * x.transpose(-3,-2) / dx
dB0 += y
dB0   = dB0 + z
dB0 += mean
```

Temporary variables

- for i in range(x): vs for n in range(x):
- for key, value in dct.items() vs
  for k, v in dct.items():

File and function names

Be consistent with existing code!

Consistency through out a project looks more professional

Quasi-exception: *nesting*

- Should still be consistent!

Nested for-loops:

Avoid using x and y!

- for i in range(data.shape[1]):
  for ii in range(data.shape[2]):
- Example pairs: i, ii vs j, k vs i, n vs n, m

Maintainability

Modularity and Reusability

- In-line coding vs
  Modular Programming

```
p = params[:,self.index['metabolites']].unsqueeze(-1).unsqueeze(-1)
fid = p.repeat_interleave(2, dim=-2).mul(self.syn_basis_fids)

# Define basis spectra coefficients
if gen: print('>>>>> Preparing metabolite coefficients')
fid = self.modulate(fids=self.syn_basis_fids,
                    params=params[:,self.index['metabolites']])
```

Benefits of Modular Programming:

- Fewer lines of code
- Easier to read
- Easier to maintain
- Easier to debug
- Easier to document
- Defensive programming!

Function calls in my simulator
- 35 regular functions (inside the simulator class),
- 22 auxiliary functions (outside the simulator class), and
- 3 primary functions for running the simulations.
  - Currently >2400 lines of code in modular format

Maintainability

In-line/Modular Mix

```
# Add Noise
if noise:
    if gen: print('>>>>> Adding noise')
    transients, zeros = None, None
    if multicoil>1:
        transients = params[:,self.index['coil_snr']]
        zeros = torch.where(params[:,self.index['coil_sens']]<=0.0,
                            1,0).sum(dim=-1,keepdims=True)
    noise = self.generate_noise(fid=fidSum,
                                max_val=mx_values,
                                param=params[:,self.index['snr']],
                                zeros=zeros, # num of zeroed out coils
                                transients=transients,
                                uncorrelated=True)
    d = -4 if multicoil>1 else -3
    fidSum = torch.stack((fidSum.clone() + noise, fidSum), dim=d)
    spectral_fit = torch.stack((spectral_fit.clone() + noise,
                                spectral_fit), dim=d)
    # Keep both noisey transients and clean transients
    # output.shape: [bS, ON\OFF, [noisy/clean], transients, channels, length]
    #                    transients, channels, length]
```

Modular Only

```
# Rephasing Spectrum
if phi0:
    fidSum = self.zero_order_phase(fid=fidSum,
                                   phi0=params[:,self.index['phi0']])

# Rephasing Spectrum
if phi1:
    fidSum = self.first_order_phase(fid=fidSum,
                                    phi1=params[:,self.index['phi1']])

# Frequency Shift
if fshift_g:
    fidSum = self.frequency_shift(fid=fidSum,
                                  param=params[:,self.index['f_shift']])

# Eddy Currents
if eddy:
    fidSum = self.first_order_eddy_currents(fid=fidSum,
                                            params=params[:,self.index['ecc']])
```

Documentation

Documentation will help:

You with developing **your own** code **AND**

You and your collaborators to understand **each other's** code

What do we mean by “*documentation*”?

Documentation - Describe inputs/outputs

```
def generate_noise(self, fid, param, max_val, zeros, transients, uncorrelated):

def generate_noise(self,
            fid: torch.Tensor,              # torch.Size([bS, (transients), channels, specLength])
            param: torch.Tensor,       # torch.Size([bS])
            max_val: torch.Tensor,   # max_val.ndim==fid.ndim
            zeros: torch.Tensor,         # number of coils with no signal;
                                         # *coil sensitivities; torch.Size([bS, num_transients])
            transients: torch.Tensor=None, # coil SNR values
            uncorrelated: bool=True, # correlate real/imaginary?
           ) -> torch.Tensor:            # fid.shape
```

Documentation - Explain the steps

```
# Load the simualted data
with open(datapath, 'r') as file:
    data = io.loadmat(file)
    specDataCmplx = data['spectra']
    header = data['header']

# Reshape data to fit into the container
x, y = reshape if not isinstance(reshape, type(None)) else 1, 1
z = specDataCmplx.shape[0] / (x*y)
s = specDataCmplx.shape[-1]
specDataCmplx = specDataCmplx.reshape(x,y,z,s)

# Define the metadata
json_full, meta_dict = self.set_metadata(name, header)

# Write the NIfTI files
self.write_NIfTIMRS(specDataCmplx, self.currNiftiOrientation, header,
                    json_full, path, os.path.splitext(name)[0])
```

Documentation - Important Details

```
# Define the delta values in each direction centered at 1 to avoid
# zeroing out values when centered at 0.
center = 1
# output.shape = [bS, 1, length, 1, 1]
x = batch_linspace(center-dx,center+dx,num_pts[0]).permute(0,2,1).unsqueeze(-1)
# output.shape = [bS, 1, 1, length, 1]
y = batch_linspace(center-dy,center+dy,num_pts[1]).unsqueeze(-1)
# output.shape = [bS, 1, 1, 1, length]
z = batch_linspace(center-dz,center+dz,num_pts[2]).unsqueeze(-1).permute(0,1,3,2)

# Define the changes in B0
dB0  = x * x.transpose(-3,-2)  / dx.unsqueeze(-1)  # output.shape = [bS, 1, length, length, 1]
dB0 += y
dB0  = dB0.repeat(1,1,1,z.shape[-1]) + z            # output.shape = [bS, 1, length, length, length]
dB0 += mean - center
# dB0.shape = [bS, 1, length, length, length]
```

Documentation - Describe your files

Osprey Example

```
1 →  % op_autophase.m
2 →  % Jamie Near, McGill University 2015.
     %
3 →  % USAGE:
     % [out,phaseShift]=op_autophase(in,ppmmin,ppmmax,ph,dimNum);
     %
4 →  % DESCRIPTION:
     % Search for the peak located between ppmmin and ppmmax, and then phase the
     % spectrum so that that peak reaches the desired phase.
     %
5 →  % INPUTS:
     % in          = input data in matlab structure format.
     % ppmmin      = minimum of ppm search range.
     % ppmmax      = maximum of ppm search range.
     % ph          = desired phase value in degrees [optional.  Default=0].
     % dimNum      = which subSpec dimension to use for phasing? [Only for use in data with multiple subSpectra].
     %
6 →  % OUTPUTS:
     % out         = Output following automatic phasing.
     % phaseShift = The phase shift (in degrees) that was applied.
```

1. File name
2. Author
3. Function call
4. Description
5. Inputs
6. Outputs

Questions?

Happy Hacking!

Example: Data Simulator

Starting Point

Steps:

1. Write code for entire model

2: Test code

3: Publish code

Essentially true, yes, but...

- Poorly thought out
- No concrete steps
- No clear starting point

Better Developed

Steps:

1. Sketch outline for approach
   - E.g. 1 sample at a time or batches?
   - What is the bare minimum?
   - Order of operations

2: Write code for bare minimum example
   - Load basis set
   - Modulate and sum basis functions
   - Save outputs

Efficiency

- Mentioning for completeness

- Keep in mind how the required computational resources change
- Efficiency can be optimized to different levels

- Basic efficiency should be kept in mind during development
- Optimization should be done towards the end