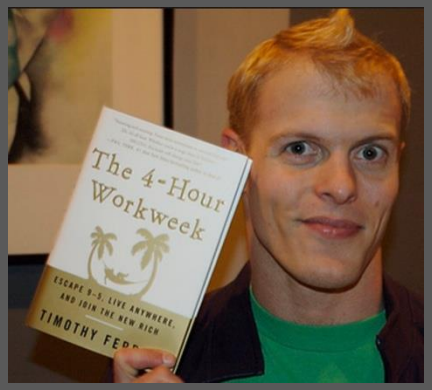
Tools of the Trade: Leveraging DevOps for Research Software Development

Software Engineering Course
UTSW - 2025

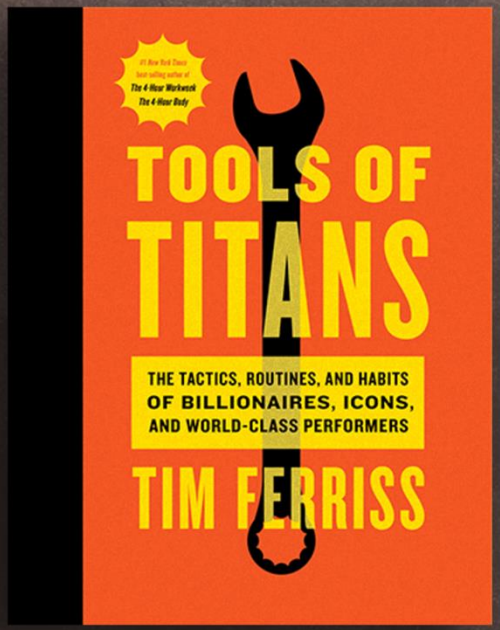
Andrew R. Jamieson, Ph.D.
Assistant Professor - Bioinformatics



From #1 The New York Times bestselling author Tim Ferriss

TOOLS OF TITANS

THE TACTICS, ROUTINES, AND HABITS OF BILLIONAIRES, ICONS, AND WORLD-CLASS PERFORMERS



TOOLS OF TITANS

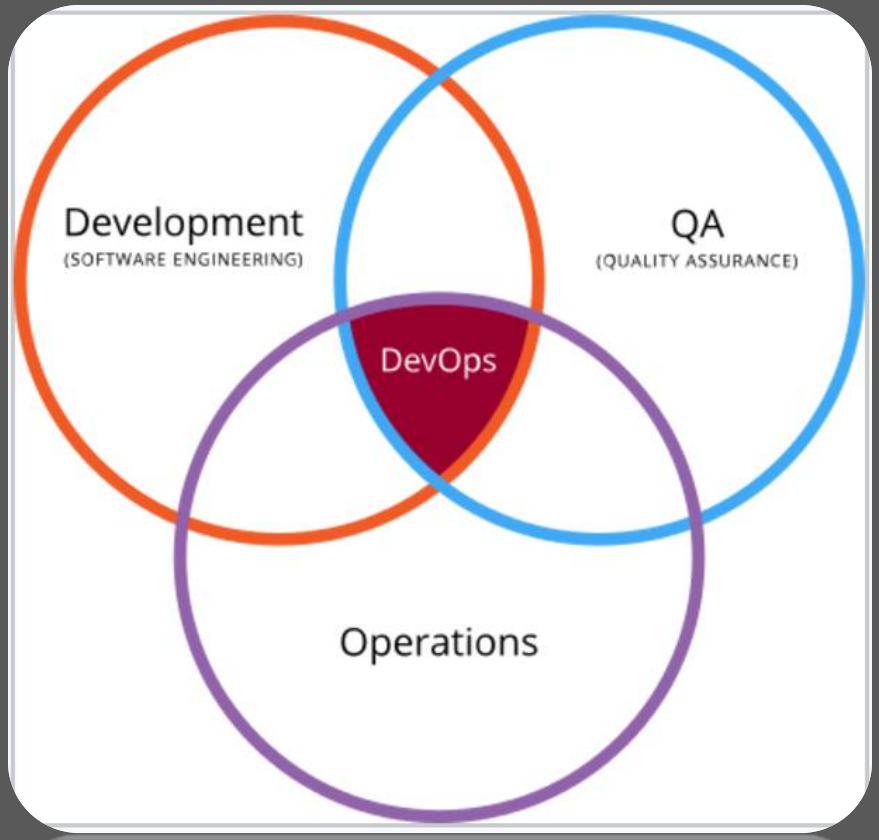
DevOps

DevOps

- Leverage best practices & tools of modern software development
  - Version Control (GIT)
  - Agile methodologies
  - Automatic software testing & deployment
    - Continuous Integration
    - Continuous Delivery

Goal: Accelerate Research Productivity

What is DevOps?



Development + Operations

Development + Operations

Code → {non-value add extra stuff} → production

Development + Operations

Code → {non-value add extra stuff} → production

Code → production

Minimize Downtime/avoid errors (i.e., keep customers happy) while :

- Fixing Bugs
- Adding features
- Upgrading the infrastructure

Development + Operations

Code → {non-value add extra stuff} → production

Code → production

Minimize Downtime/avoid errors (i.e., keep customers happy) while :

- Fixing Bugs
- Adding features
- Upgrading the infrastructure

Stay nibble, but deploy on large scale

Development + Operations

Code → {non-value add extra stuff} → production

Code → production

Automation

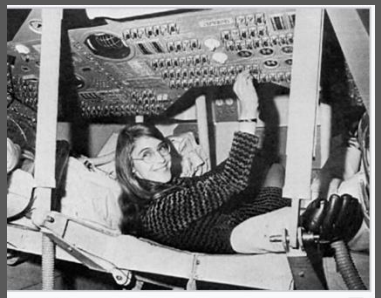
Minimize Downtime/avoid errors (i.e., keep customers happy) while :

- Fixing Bugs
- Adding features
- Upgrading the infrastructure

Stay nibble, but deploy on large scale

DevOps

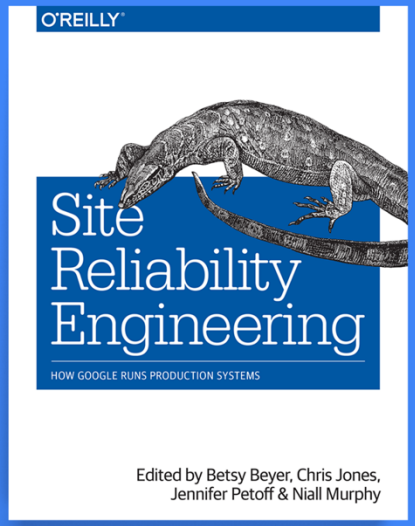
Origin Stories…



Hamilton during her time as lead Apollo flight software designer.

Google

What is SRE? In Conversation Resources Read SRE Book



Site Reliability Engineering

Edited by Betsy Beyer, Chris Jones, Jennifer Petoff and Niall Richard Murphy

Members of the SRE team explain how their engagement with the entire software lifecycle has enabled Google to build, deploy, monitor, and maintain some of the largest software systems in the world.

READ ONLINE FOR FREE BUY FROM GOOGLE BOOKS

Hamilton standing next to the navigation software that she and her MIT team produced for the Apollo project.

https://www.wired.com/2016/04/google-ensures-services-almost-never-go/

Code → production

Code → Science

Ideas to Action

Ideas to Results

DevOps for Research

Ideas to Results

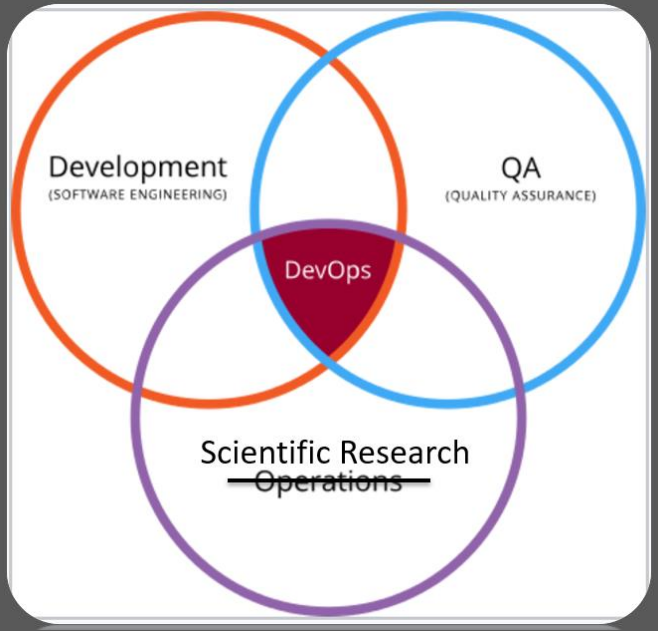
DevOps for Research ™

Ideas to Results

DevOps for Research ™

DevOps (via Wikipedia) :

DevOps is a culture, movement or practice that emphasizes the collaboration and communication of both software developers and other information-technology (IT) professionals while automating the process of software delivery and infrastructure changes. It aims at establishing a culture and environment where building, testing, and releasing software can happen rapidly, frequently, and more reliably.



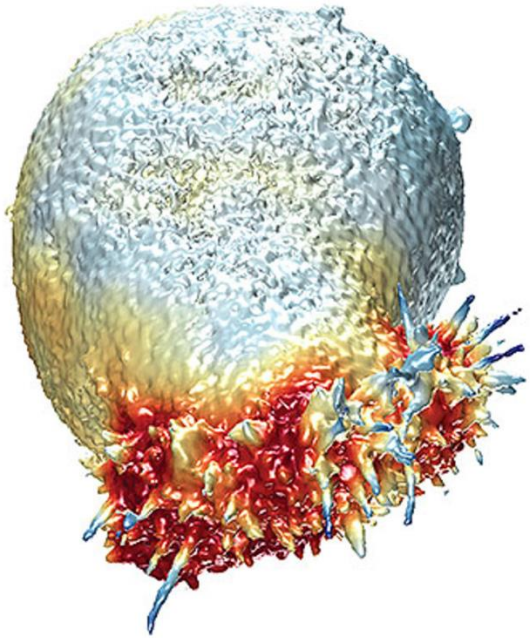
DevOps "Tools" (via Wikipedia) :

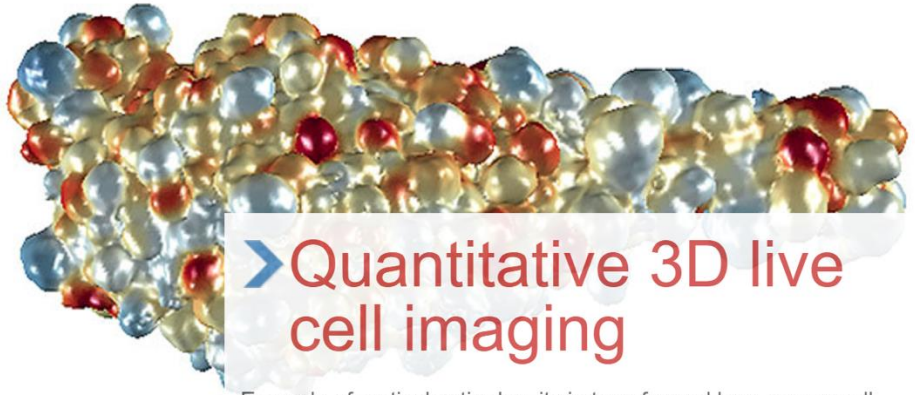
Code – Code development and review, Version control tools, code merging
Build – continuous integration tools, build status
Test – Test and results determine performance
Package – Artifact repository, application pre-deployment staging
Release – Change management, release approvals, release automation
Configure – Infrastructure config and management, Infrastructure as Code tools
Monitor – Applications performance monitoring, end user experience

Danuser lab

Danuser Lab

CONTACT US NEWS SOFTWARE





Quantitative 3D live cell imaging

Example of cortical actin density in transformed lung cancer cell model (left) and primary melanoma cell (right).

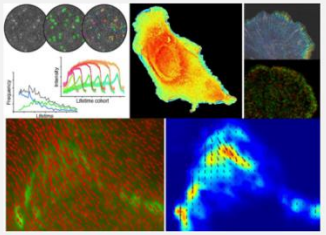
1 2 3 4

Who We Are

Meet the Lab

Open Positions

Software



Danuser lab - Software

Danuser Lab

CONTACT US

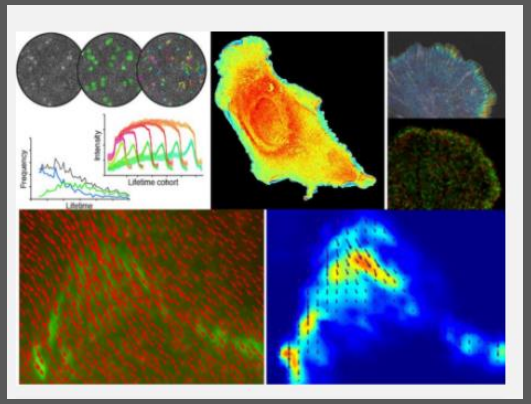
Software

Software

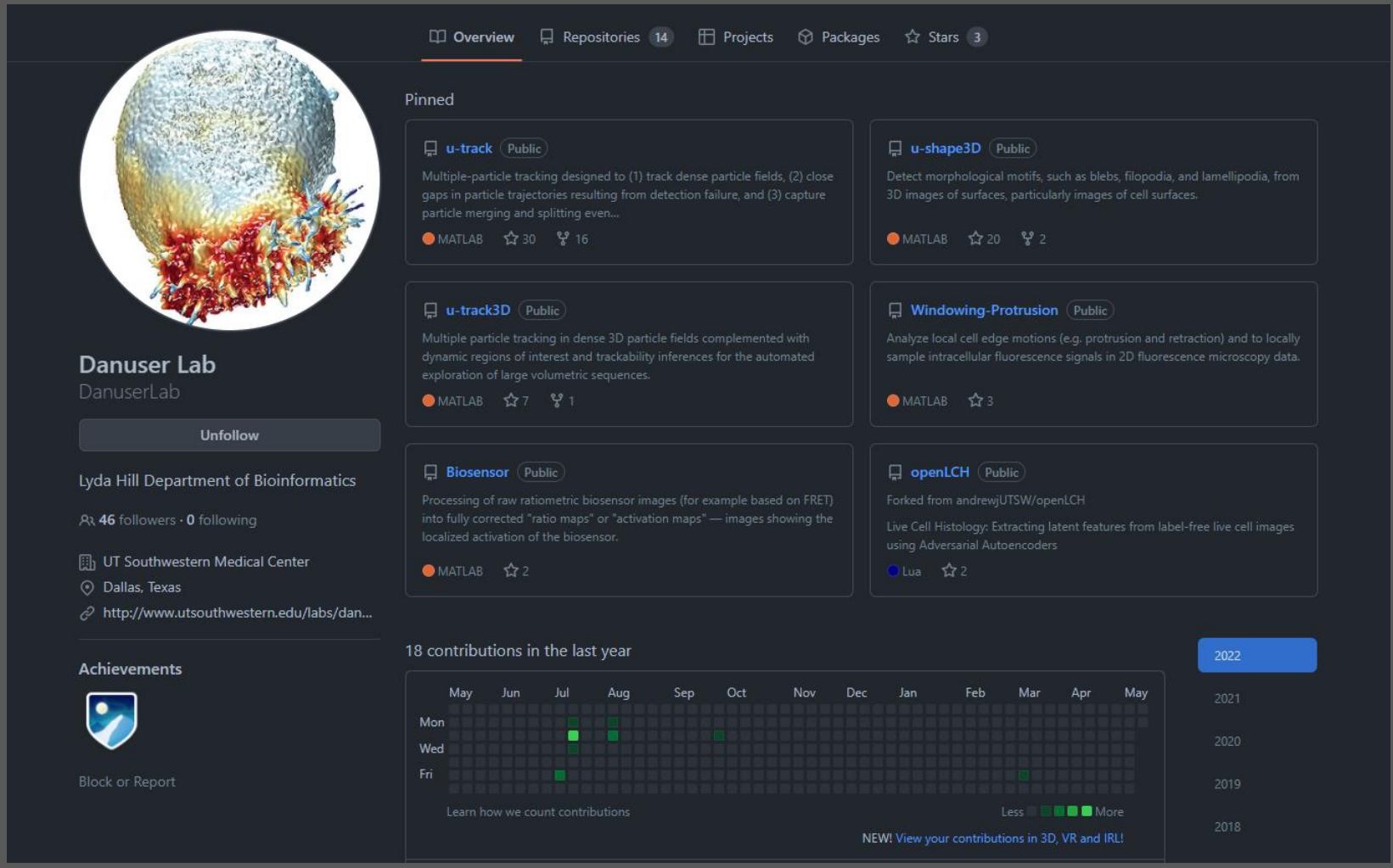
u-track	Particle tracking
Biosensors	Quantitative ratiometric image analysis
TFM	Traction force microscopy
cmeAnalysis	Clathrin-mediated endocytosis analysis
QFSM	Quantitative fluorescent speckle microscopy
DeBias	Decoupling global biases and local interactions between cell biological variables

u-track 2.0

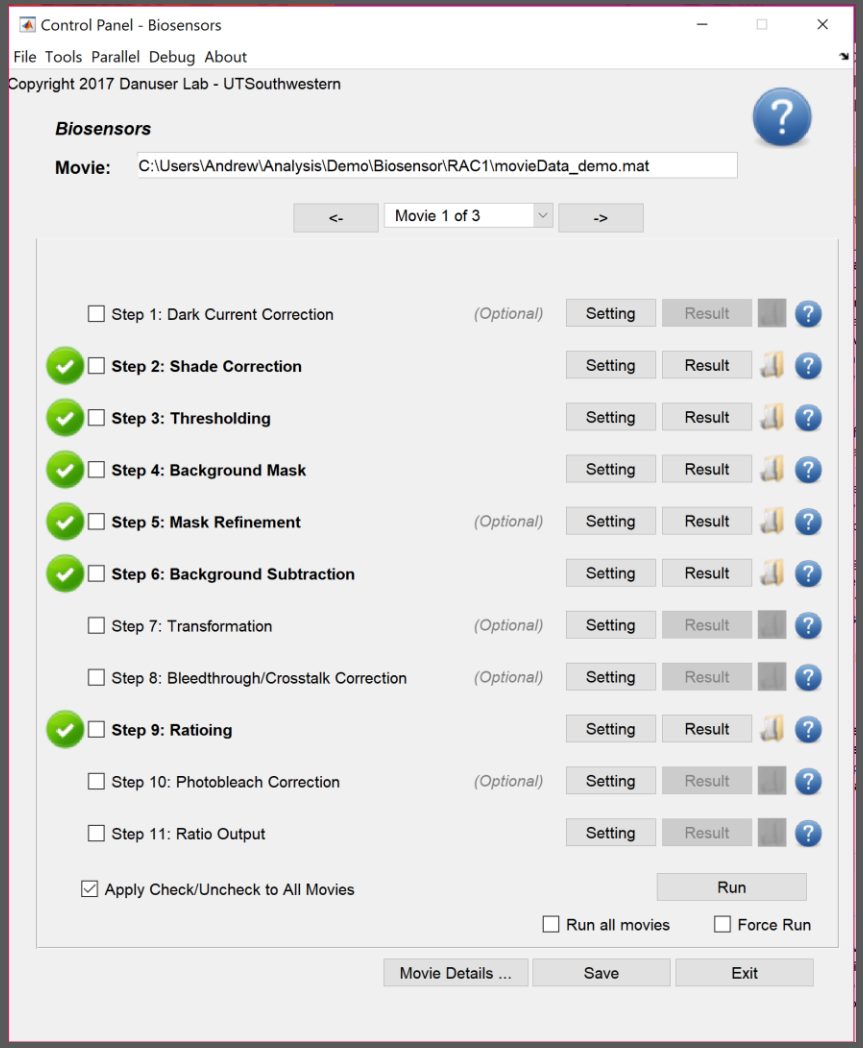
u-track is a multiple-particle tracking MATLAB software that is designed to (1) track dense particle fields, (2) close gaps in particle trajectories resulting from detection failure, and (3) capture particle merging and splitting events resulting from occlusion or genuine aggregation and dissociation events. Its core is based on formulating correspondence problems as linear assignment



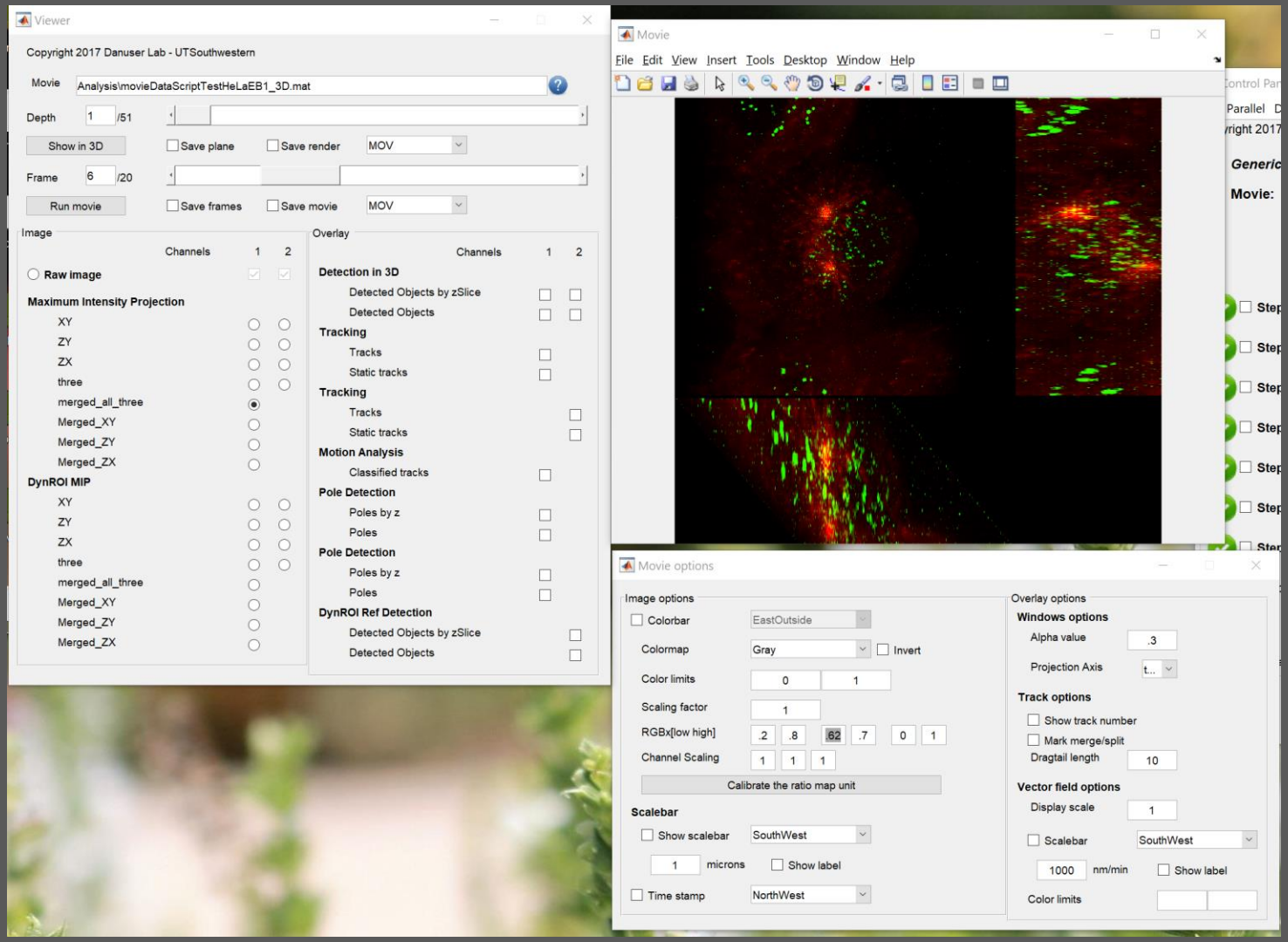
Danuser lab - Software



MATLAB GUI


Control Panel - Biosensors
File Tools Parallel Debug About
Copyright 2017 Danuser Lab - UTSouthwestern
Biosensors
Movie:
C:\Users\Andrew\Analysis\Demo\Biosensor\RAC1\movieData_demo.mat
<-
Movie 1 of 3
->
Step 1: Dark Current Correction (Optional) Setting Result
Step 2: Shade Correction Setting Result
Step 3: Thresholding Setting Result
Step 4: Background Mask Setting Result
Step 5: Mask Refinement (Optional) Setting Result
Step 6: Background Subtraction Setting Result
Step 7: Transformation (Optional) Setting Result
Step 8: Bleedthrough/Crosstalk Correction (Optional) Setting Result
Step 9: Ratioing Setting Result
Step 10: Photobleach Correction (Optional) Setting Result
Step 11: Ratio Output Setting Result
Apply Check/Uncheck to All Movies
Run
Run all movies
Force Run
Movie Details ...
Save
Exit

MATLAB GUI - Viewer



MATLAB GUI – "Under the hood"

Sophisticated Architecture

- OOP
- Extensible
- Flexible
- Modular
- Customizable

- Easy to build new..
  - Packages/ Pipelines
  - processes
  - GUIs

```
    methods (Static)

        function name = getName()
            name = 'U-Track';
        end
        function m = getDependencyMatrix(i,j)
            m = [0 0 0;  %1 DetectionProcess
                 1 0 0;  %2 TrackingProcess
                 1 1 0;];%3 PostTrackingProcess
            if nargin<2, j=1:size(m,2); end
            if nargin<1, i=1:size(m,1); end
            m=m(i,j);
        end

        function varargout = GUI(varargin)
            % Start the package GUI
            varargout{1} = trackingPackageGUI(varargin{:});
        end
        function classes = getProcessClassNames(index)
            classes = {
                'DetectionProcess',...
                'TrackingProcess',...
                'PostTrackingProcess'};
            if nargin==0, index=1:numel(classes); end
            classes=classes(index);
        end

        function objects = getConcretePackages()
            objects(1).name = 'Single particles';
            objects(1).packageConstr = @UTrackPackage;
            objects(2).name = 'Microtubules plus-ends';
            objects(2).packageConstr = @PlusTipTrackerPackage;
            objects(3).name = 'Nuclei';
            objects(3).packageConstr = @NucleiTrackingPackage;
        end

    end
end
```

MATLAB GUI – "Under the hood"

- U-track Package Class
  - Process Step Dependencies

```
function m = getDependencyMatrix(i,j)
    m = [0 0 0;  %1 DetectionProcess
         1 0 0;  %2 TrackingProcess
         1 1 0;];%3 PostTrackingProcess
    if nargin<2, j=1:size(m,2); end
    if nargin<1, i=1:size(m,1); end
    m=m(i,j);
end
```

```
    methods (Static)

        function name = getName()
            name = 'U-Track';
        end
        function m = getDependencyMatrix(i,j)
            m = [0 0 0;  %1 DetectionProcess
                 1 0 0;  %2 TrackingProcess
                 1 1 0;];%3 PostTrackingProcess
            if nargin<2, j=1:size(m,2); end
            if nargin<1, i=1:size(m,1); end
            m=m(i,j);
        end

        function varargout = GUI(varargin)
            % Start the package GUI
            varargout{1} = trackingPackageGUI(varargin{:});
        end
        function classes = getProcessClassNames(index)
            classes = {
                'DetectionProcess',...
                'TrackingProcess',...
                'PostTrackingProcess'};
            if nargin==0, index=1:numel(classes); end
            classes=classes(index);
        end

        function objects = getConcretePackages()
            objects(1).name = 'Single particles';
            objects(1).packageConstr = @UTrackPackage;
            objects(2).name = 'Microtubules plus-ends';
            objects(2).packageConstr = @PlusTipTrackerPackage;
            objects(3).name = 'Nuclei';
            objects(3).packageConstr = @NucleiTrackingPackage;
        end

    end
end
```

MATLAB GUI – "Under the hood"

Process Configuration

```
function procConstr = getDefaultProcessConstructors(index)
    biosensorsConstr = {
        @DarkCurrentCorrectionProcess,...
        @ShadeCorrectionProcess,...
        @ThresholdProcess,...
        @BackgroundMasksProcess,...
        @MaskRefinementProcess,...
        @BackgroundSubtractionProcess,...
        @TransformationProcess,...
        @BleedthroughCorrectionProcess,...
        @RatioProcess,...
        @PhotobleachCorrectionProcess,...
        @OutputRatioProcess...
        };
    if nargin==0, index=1:numel(biosensorsConstr); end
    procConstr=biosensorsConstr(index);
end
```

```
methods (Static)

    function name = getName()
        name = 'Biosensors';
    end

    function m = getDependencyMatrix(i,j)

        m = [0 0 0 0 0 0 0 0 0 0 0; %1 DarkCurrentCorrectionProcess
            2 0 0 0 0 0 0 0 0 0 0;  %2 ShadeCorrectionProcess
            0 1 0 0 0 0 0 0 0 0 0;  %3 SegmentationProcess
            0 0 1 0 0 0 0 0 0 0 0;  %4 BackgroundMasksProcess
            0 0 1 0 0 0 0 0 0 0 0;  %5 MaskRefinementProcess
            0 1 0 1 0 0 0 0 0 0 0;  %6 BackgroundSubtractionProcess
            0 0 1 0 2 1 0 0 0 0 0;  %7 TransformationProcess
            0 0 0 0 0 1 2 0 0 0 0;  %8 BleedthroughCorrectionProcessj
            0 0 1 0 2 1 2 2 0 0 0;  %9 RatioProcess
            0 0 0 0 0 0 0 0 1 0 0;  %10PhotobleachCorrectionProcess
            0 0 0 0 0 0 0 0 1 2 0]; %11OutputRatioProcess
        if nargin<2, j=1:size(m,2); end
        if nargin<1, i=1:size(m,1); end
        m=m(i,j);
    end

    function varargout = GUI(varargin)
        % Start the package GUI
        varargout{1} = biosensorsPackageGUI(varargin{:});
    end

    function procConstr = getDefaultProcessConstructors(index)
        biosensorsConstr = {
            @DarkCurrentCorrectionProcess,...
            @ShadeCorrectionProcess,...
            @ThresholdProcess,...
            @BackgroundMasksProcess,...
            @MaskRefinementProcess,...
            @BackgroundSubtractionProcess,...
            @TransformationProcess,...
            @BleedthroughCorrectionProcess,...
            @RatioProcess,...
            @PhotobleachCorrectionProcess,...
            @OutputRatioProcess...
            };
        if nargin==0, index=1:numel(biosensorsConstr); end
        procConstr=biosensorsConstr(index);
    end

    function classes = getProcessClassNames(index)
        biosensorsClasses = {
            'DarkCurrentCorrectionProcess',...
            'ShadeCorrectionProcess',...
            'SegmentationProcess',...
            'BackgroundMasksProcess',...
            'MaskRefinementProcess',...
            'BackgroundSubtractionProcess',...
            'TransformationProcess',...
            'BleedthroughCorrectionProcess',...
            'RatioProcess',...
            'PhotobleachCorrectionProcess',...
            'OutputRatioProcess'};
        if nargin==0, index=1:numel(biosensorsClasses); end
        classes=biosensorsClasses(index);
    end
```

MATLAB GUI – "Under the hood"

- Biosensor Package Class
  - Process Step Dependencies

```
function m = getDependencyMatrix(i,j)

    m = [0 0 0 0 0 0 0 0 0 0 0 0; %1 DarkCurrentCorrectionProcess
         2 0 0 0 0 0 0 0 0 0 0 0;  %2 ShadeCorrectionProcess
         0 1 0 0 0 0 0 0 0 0 0 0;  %3 SegmentationProcess
         0 0 1 0 0 0 0 0 0 0 0 0;  %4 BackgroundMasksProcess
         0 0 1 0 0 0 0 0 0 0 0 0;  %5 MaskRefinementProcess
         0 1 0 1 0 0 0 0 0 0 0 0;  %6 BackgroundSubtractionProcess
         0 0 1 0 2 1 0 0 0 0 0 0;  %7 TransformationProcess
         0 0 0 0 0 1 2 0 0 0 0 0;  %8 BleedthroughCorrectionProcessj
         0 0 1 0 2 1 2 2 0 0 0 0;  %9 RatioProcess
         0 0 0 0 0 0 0 0 1 0 0 0;  %10PhotobleachCorrectionProcess
         0 0 0 0 0 0 0 0 0 1 2 0]; %11OutputRatioProcess
    if nargin<2, j=1:size(m,2); end
    if nargin<1, i=1:size(m,1); end
    m=m(i,j);
end
```

```
methods (Static)

    function name = getName()
        name = 'Biosensors';
    end

    function m = getDependencyMatrix(i,j)

        m = [0 0 0 0 0 0 0 0 0 0 0 0; %1 DarkCurrentCorrectionProcess
             2 0 0 0 0 0 0 0 0 0 0 0;  %2 ShadeCorrectionProcess
             0 1 0 0 0 0 0 0 0 0 0 0;  %3 SegmentationProcess
             0 0 1 0 0 0 0 0 0 0 0 0;  %4 BackgroundMasksProcess
             0 0 1 0 0 0 0 0 0 0 0 0;  %5 MaskRefinementProcess
             0 1 0 1 0 0 0 0 0 0 0 0;  %6 BackgroundSubtractionProcess
             0 0 1 0 2 1 0 0 0 0 0 0;  %7 TransformationProcess
             0 0 0 0 0 1 2 0 0 0 0 0;  %8 BleedthroughCorrectionProcessj
             0 0 1 0 2 1 2 2 0 0 0 0;  %9 RatioProcess
             0 0 0 0 0 0 0 0 1 0 0 0;  %10PhotobleachCorrectionProcess
             0 0 0 0 0 0 0 0 0 1 2 0]; %11OutputRatioProcess
        if nargin<2, j=1:size(m,2); end
        if nargin<1, i=1:size(m,1); end
        m=m(i,j);
    end

    function varargout = GUI(varargin)
        % Start the package GUI
        varargout{1} = biosensorsPackageGUI(varargin{:});
    end
    function procConstr = getDefaultProcessConstructors(index)
        biosensorsConstr = {
            @DarkCurrentCorrectionProcess,...
            @ShadeCorrectionProcess,...
            @ThresholdProcess,...
            @BackgroundMasksProcess,...
            @MaskRefinementProcess,...
            @BackgroundSubtractionProcess,...
            @TransformationProcess,...
            @BleedthroughCorrectionProcess,...
            @RatioProcess,...
            @PhotobleachCorrectionProcess,...
            @OutputRatioProcess...
            };
        if nargin==0, index=1:numel(biosensorsConstr); end
        procConstr=biosensorsConstr(index);
    end
    function classes = getProcessClassNames(index)
        biosensorsClasses = {
            'DarkCurrentCorrectionProcess',...
            'ShadeCorrectionProcess',...
            'SegmentationProcess',...
            'BackgroundMasksProcess',...
            'MaskRefinementProcess',...
            'BackgroundSubtractionProcess',...
            'TransformationProcess',...
            'BleedthroughCorrectionProcess',...
            'RatioProcess',...
            'PhotobleachCorrectionProcess',...
            'OutputRatioProcess'};
        if nargin==0, index=1:numel(biosensorsClasses); end
        classes=biosensorsClasses(index);
    end
```

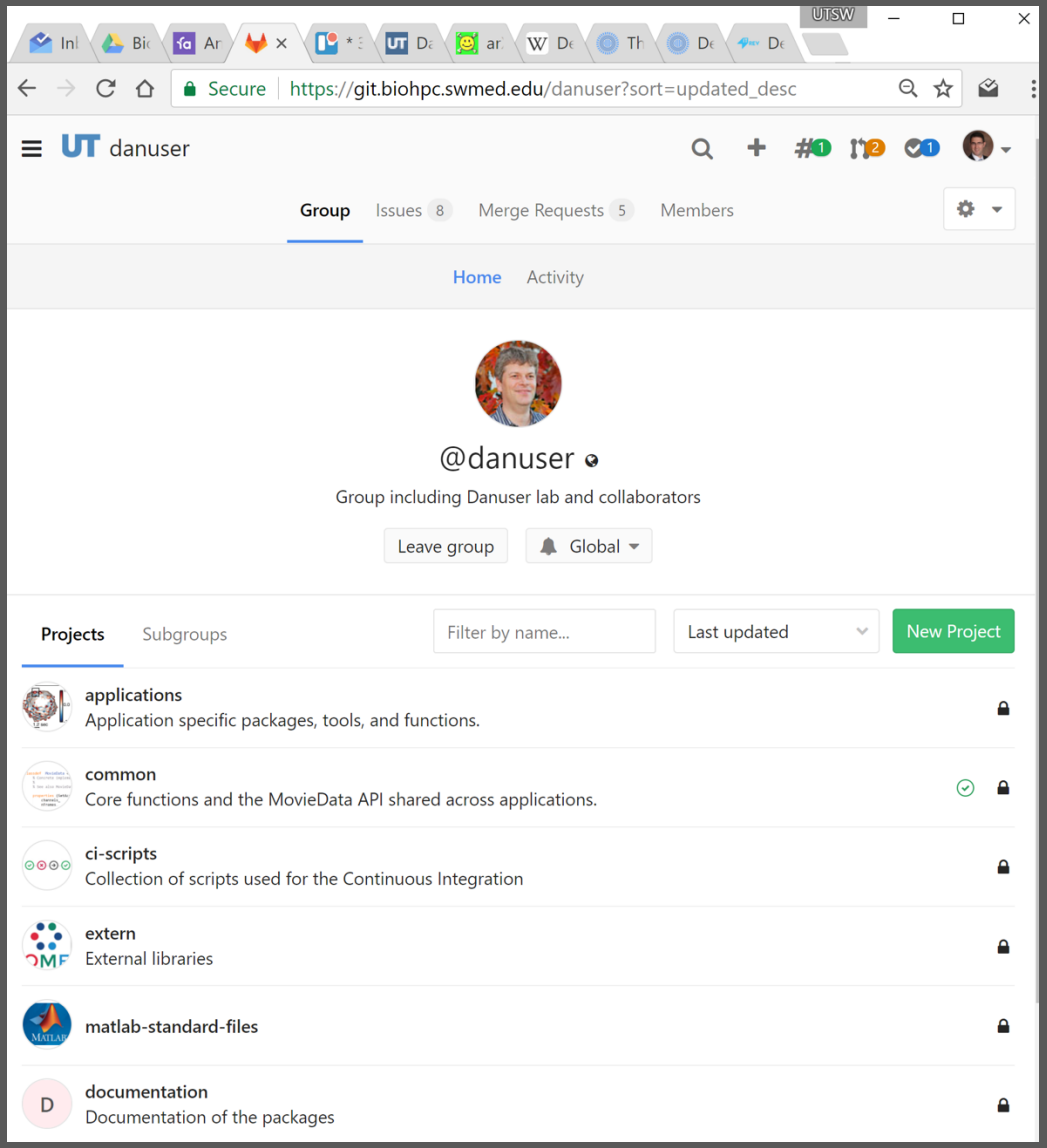
Team-Driven Scientific Software Development

- Maintain code quality/integrity
- Maintain code functionality
- Train lab members
- Develop custom code: packages/GUIs/algorithms
- Tackle technical challenges with team

Danuser Lab + Git

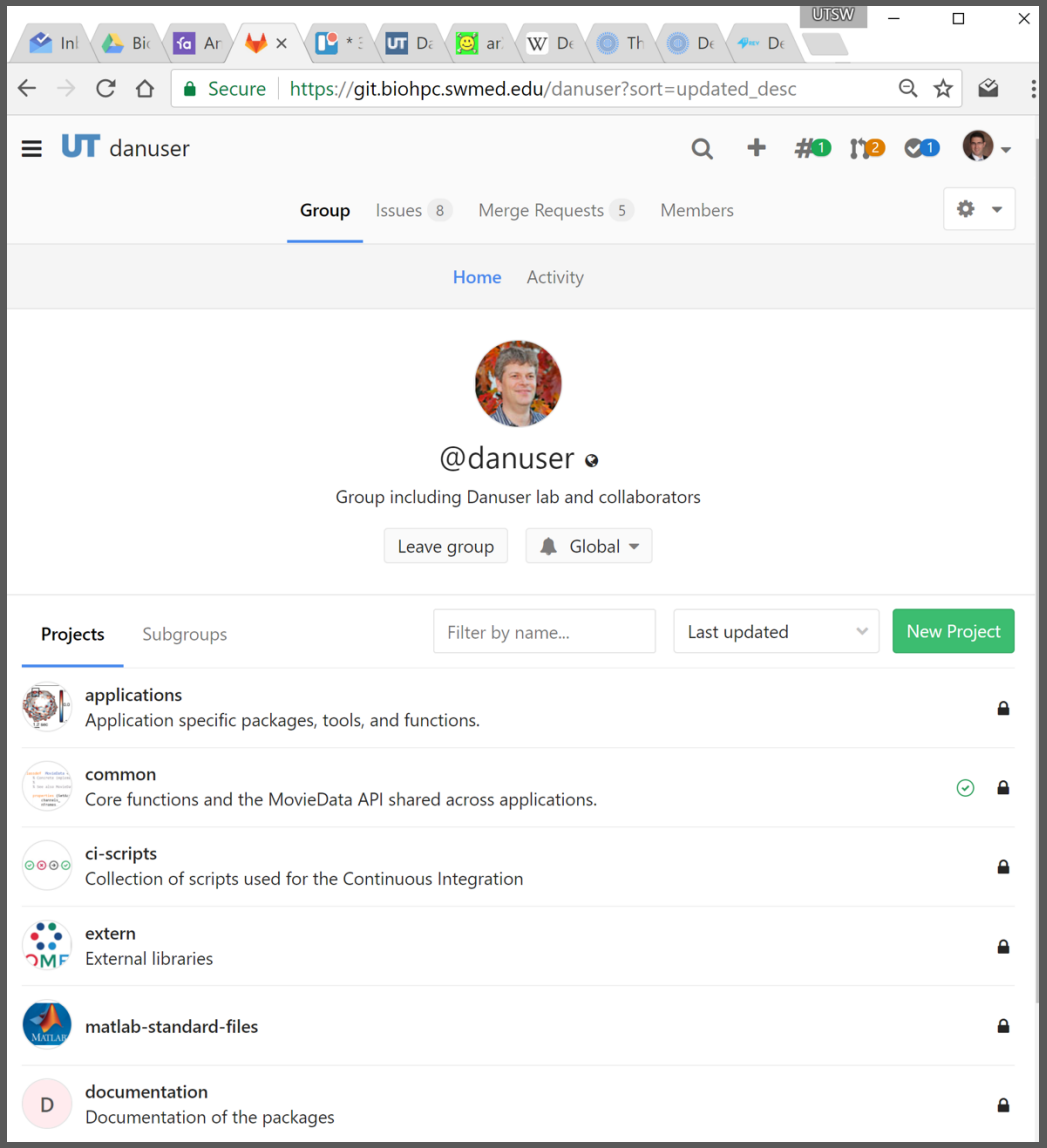
Danuser Lab + Git

- Three main repos
  - Extern
  - Applications
  - Common



Danuser Lab + Git

- Three main repos
  - Extern
  - Applications
  - Common



Continuous Integration & Delivery

- Automatic software testing & deployment
- For every change ("commit") to our git repository code: run suite of a tests.
  - If *any* failures:
    - Code is flagged: providing immediate, actionable feedback to team – pin pointing when and where problems emerge.
  - If all tests pass:
    - Give option to deploy latest, updated package directly to community in near real-time.

Continuous Integration & Delivery

- Automatic software testing & deployment
- For every change ("commit") to our git repository code: run suite of a tests.
  - If *any* failures:
    - Code is flagged: providing immediate, actionable feedback to team – pin pointing when and where problems emerge.
  - If all tests pass:
    - Give option to deploy latest, updated package directly to community in near real-time.

Continuous Integration & Delivery

- Automatic software testing & deployment
- For every change ("commit") to our git repository code: run suite of a tests.
  - If *any* failures:
    - Code is flagged: providing immediate, actionable feedback to team – pin pointing when and where problems emerge.
  - If all tests pass:
    - Give option to deploy latest, updated package directly to community in near real-time.

Continuous Integration & Delivery

- Levels of automatic software testing
  (by increasing complexity)
  i. Unit tests (function/methods)
  ii. Integration tests (class-level/multi-function/object)
  iii. Build & test packages (multi-class)
  iv. End to end system tests (multi-package workflow)

  - Backwards compatibility (e.g., MATLAB version)
  - OS/Platform compatibility (Linux, Windows, Mac)
  - Code performance profiling

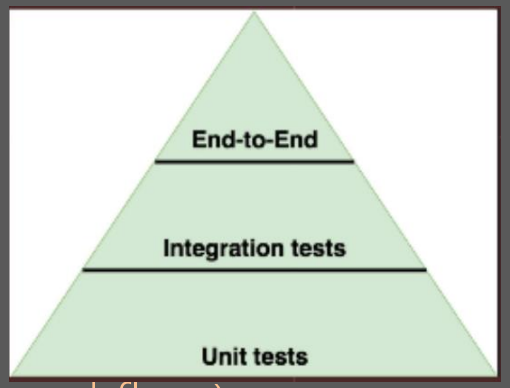
Continuous Integration & Delivery

- Levels of automatic software testing
  (by increasing complexity)
  i. Unit tests (function/methods)
  ii. Integration tests (class-level/multi-method/functions)
  iii. Build & test packages (multi-class)
  iv. End to end system tests (multi-package workflow)

  - Backwards compatibility (e.g., MATLAB version)
  - OS/Platform compatibility (Linux, Windows, Mac)
  - Code performance profiling (too slow?)
  - Regression Testing (runs but gives different results? Feature no longer work?)
  - Code coverage (% of code base run during test)

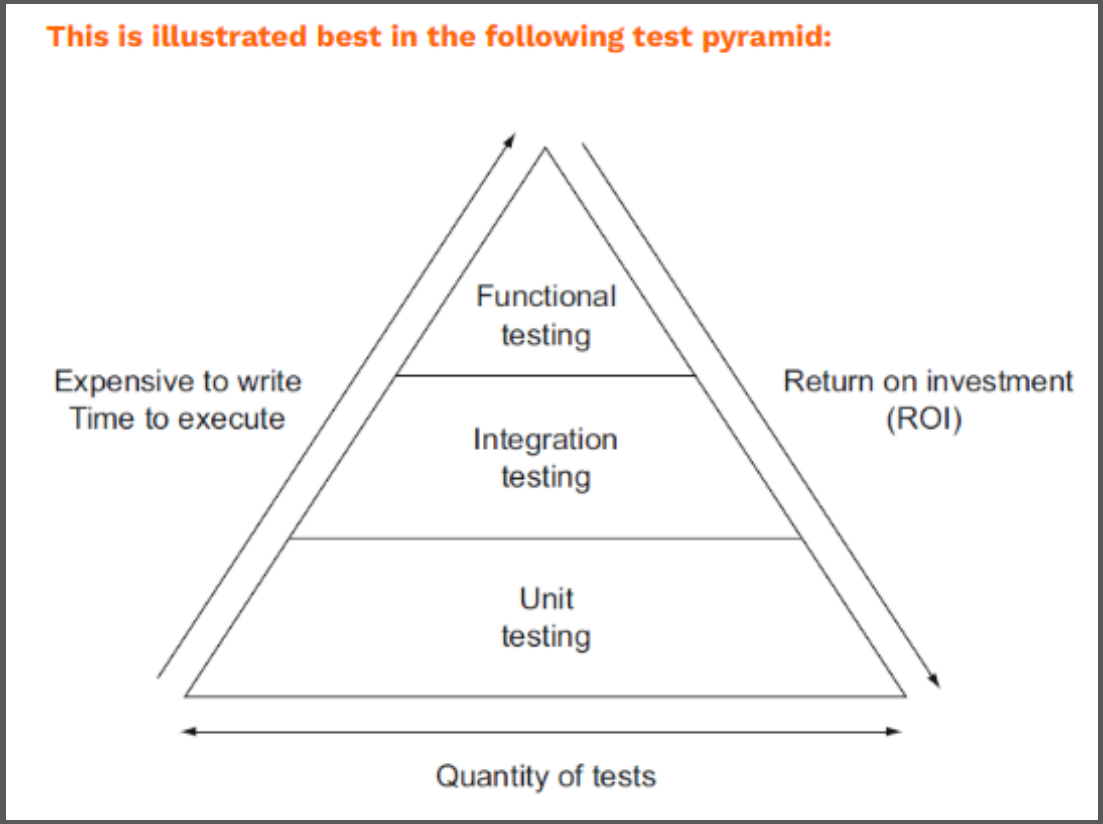
Continuous Integration & Delivery

- Levels of automatic software testing
  (by increasing complexity)
  i. Unit tests (function/methods)
  ii. Integration tests (class)
  iii. Build & test packages (multi-class)
  iv. End to end system tests (multi-package workflow)



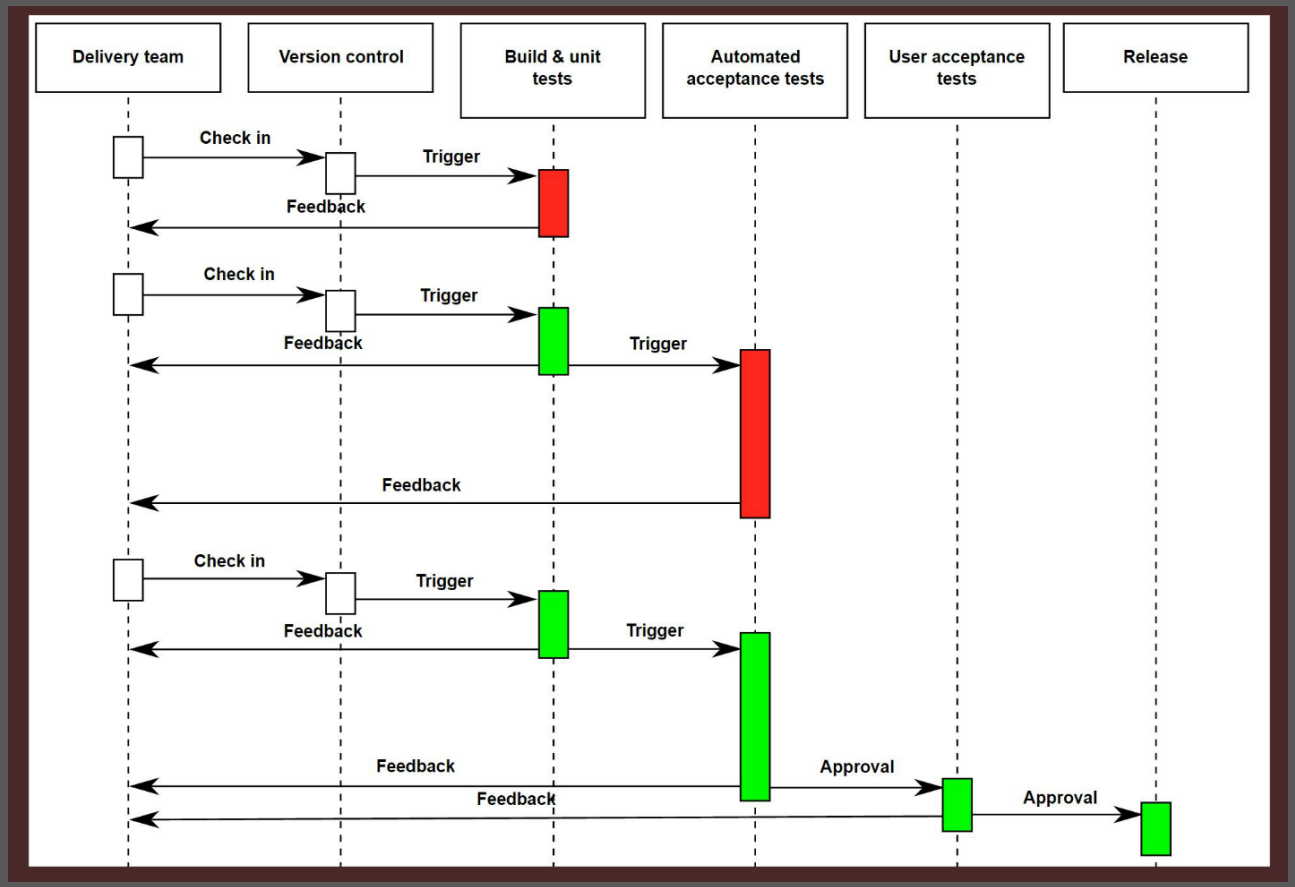
  - Backwards compatibility (e.g., MATLAB version)
  - OS/Platform compatibility (Linux, Windows, Mac)
  - Code performance profiling (too slow?)
  - Regression Testing (runs but gives different results? Things/features no longer work)
  - Code coverage (% of code base run during test)

Testing Pyramid

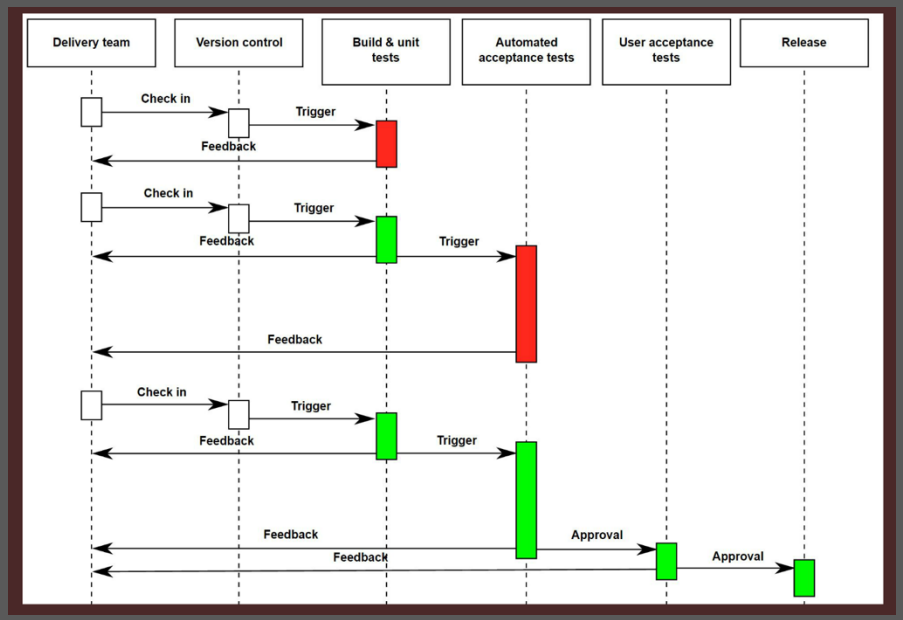


https://www.softwaretestinghelp.com/the-difference-between-unit-integration-and-functional-testing

Continuous Deployment Pipeline



Purpose of Continuous Delivery

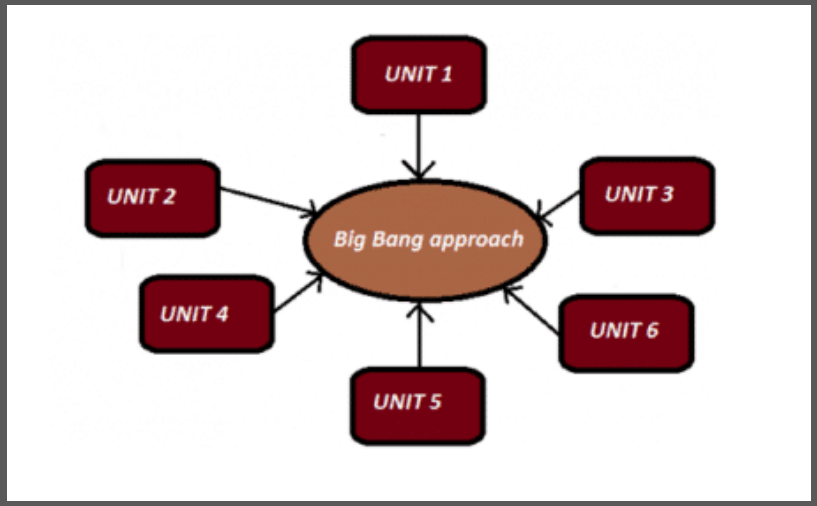


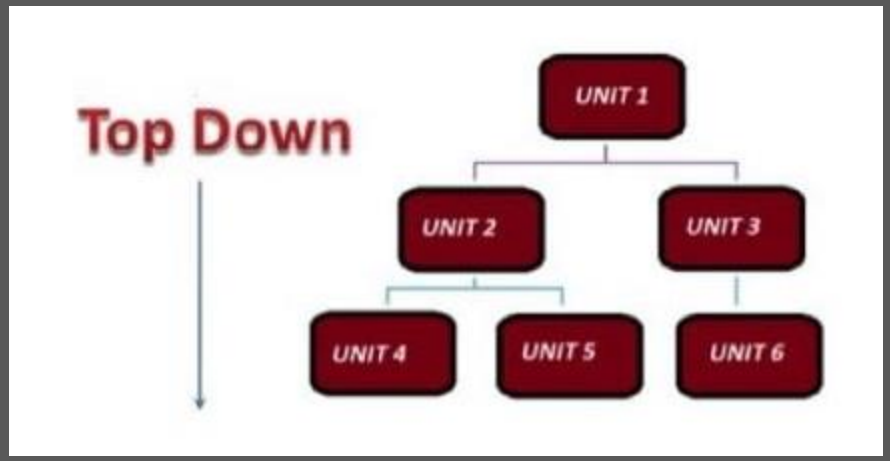
Visibility – All aspects of the delivery system including building, deploying, testing, and releasing are visible to every member of the team to promote collaboration.

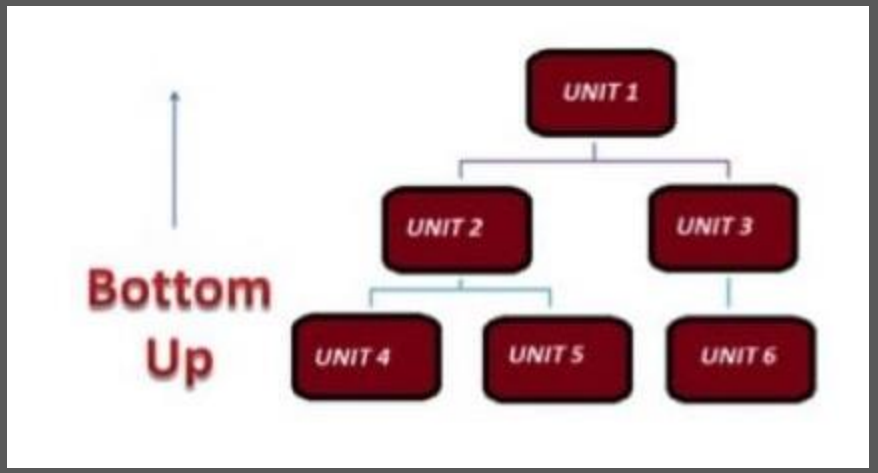
Feedback – Team members learn of problems as soon as possible when they occur so that they are able to fix them as quickly as possible.

Continually deploy – Through a fully automated process, you can deploy and release any version of the software to any environment.

Testing Design Principles



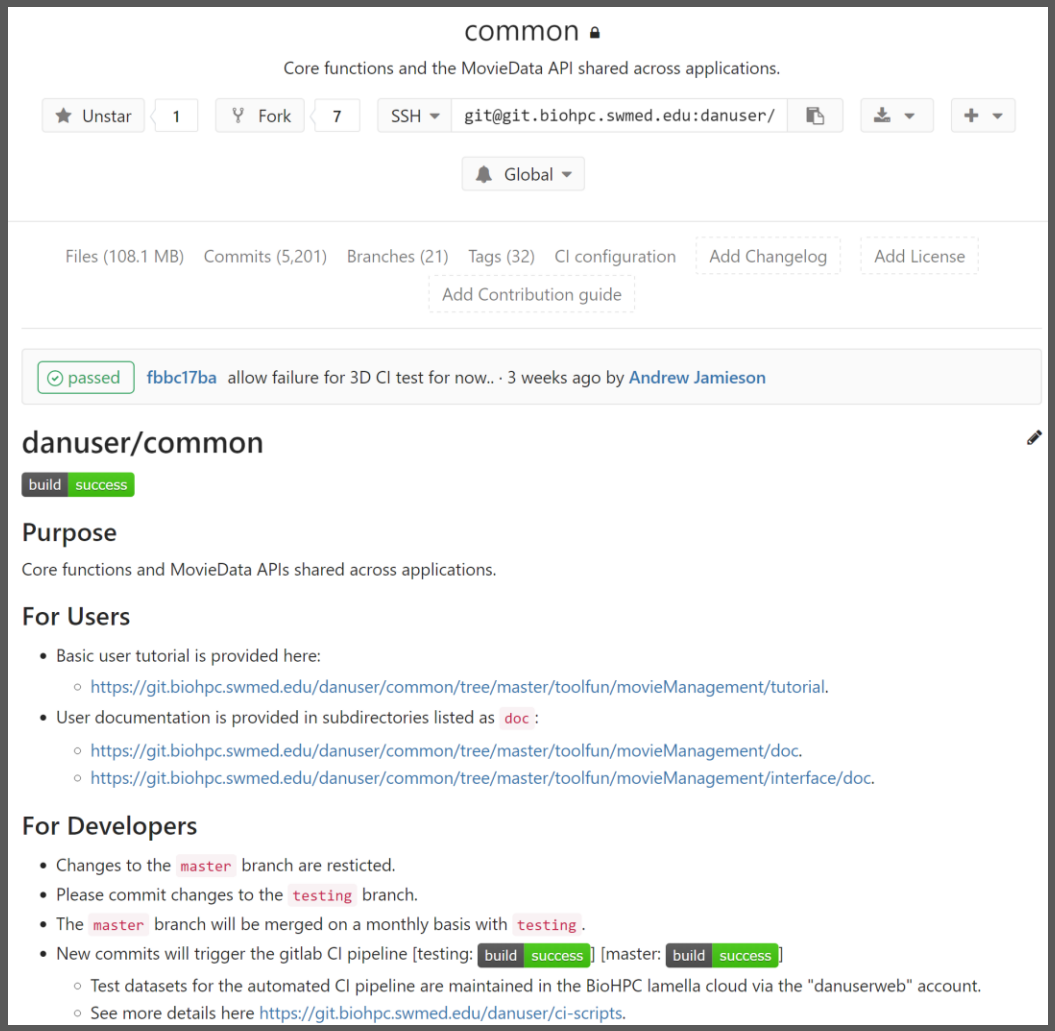




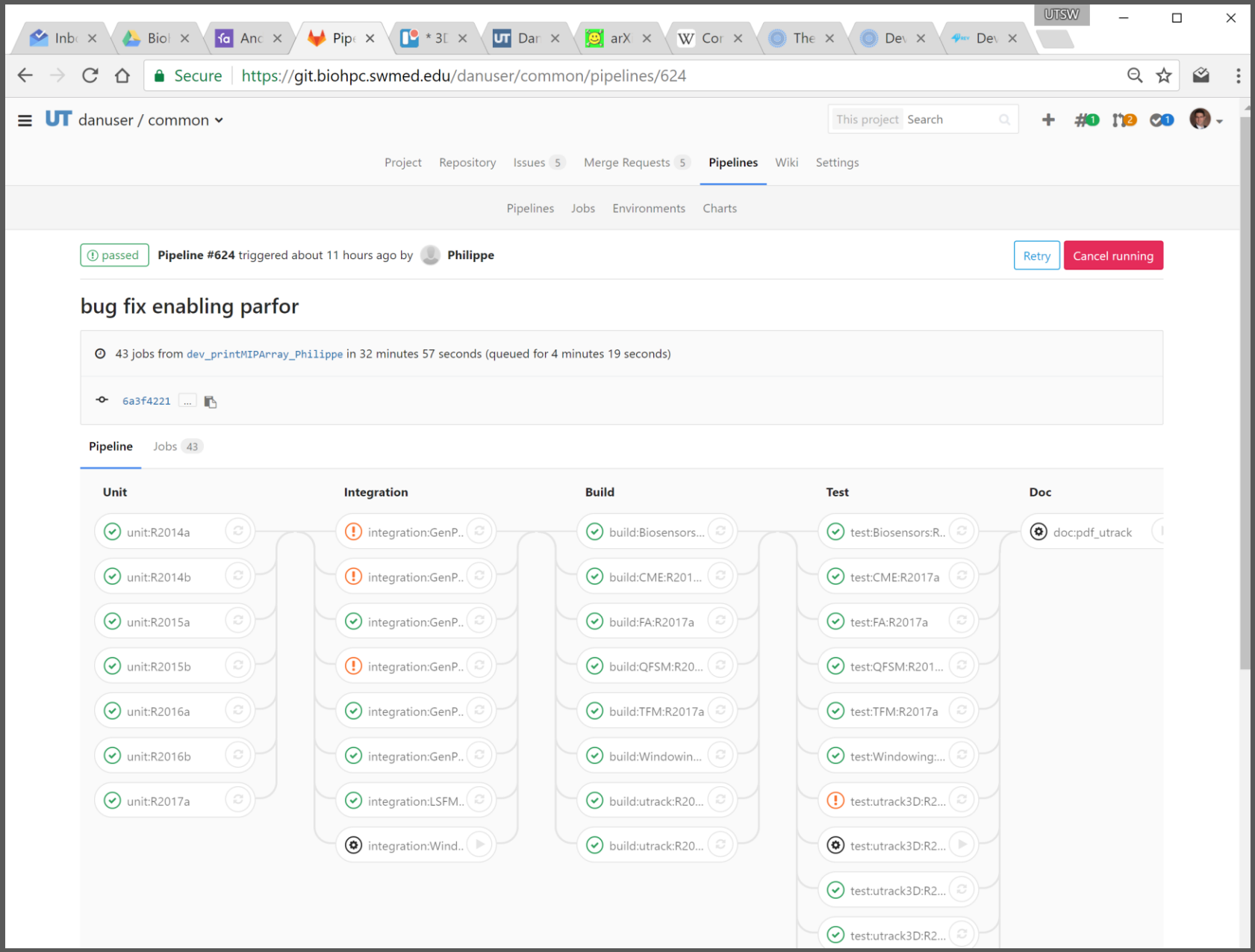
Danuser Lab + Git + CI/CD Pipeline

Let's take a look a gitlab!

Danuser Lab + Git + CI/CD Pipeline



https://git.biohpc.swmed.edu/danuser/common


UTSW
Secure
https://git.biohpc.swmed.edu/danuser/common/pipelines/624
danuser / common
This project
Search
Project
Repository
Issues 5
Merge Requests 5
Pipelines
Wiki
Settings
Pipelines
Jobs
Environments
Charts
passed
Pipeline #624 triggered about 11 hours ago by Philippe
Retry
Cancel running
bug fix enabling parfor
43 jobs from dev_printMIPArray_Philippe in 32 minutes 57 seconds (queued for 4 minutes 19 seconds)
6a3f4221
Pipeline
Jobs 43
Unit
unit:R2014a
unit:R2014b
unit:R2015a
unit:R2015b
unit:R2016a
unit:R2016b
unit:R2017a
Integration
integration:GenP..
integration:GenP..
integration:GenP..
integration:GenP..
integration:GenP..
integration:GenP..
integration:LSFM..
integration:Wind..
Build
build:Biosensors...
build:CME:R201...
build:FA:R2017a
build:QFSM:R20...
build:TFM:R2017a
build:Windowin...
build:utrack:R20...
build:utrack:R20...
Test
test:Biosensors:R..
test:CME:R2017a
test:FA:R2017a
test:QFSM:R201...
test:TFM:R2017a
test:Windowing:...
test:utrack3D:R2...
test:utrack3D:R2...
test:utrack3D:R2...
test:utrack3D:R2...
Doc
doc:pdf_utrack

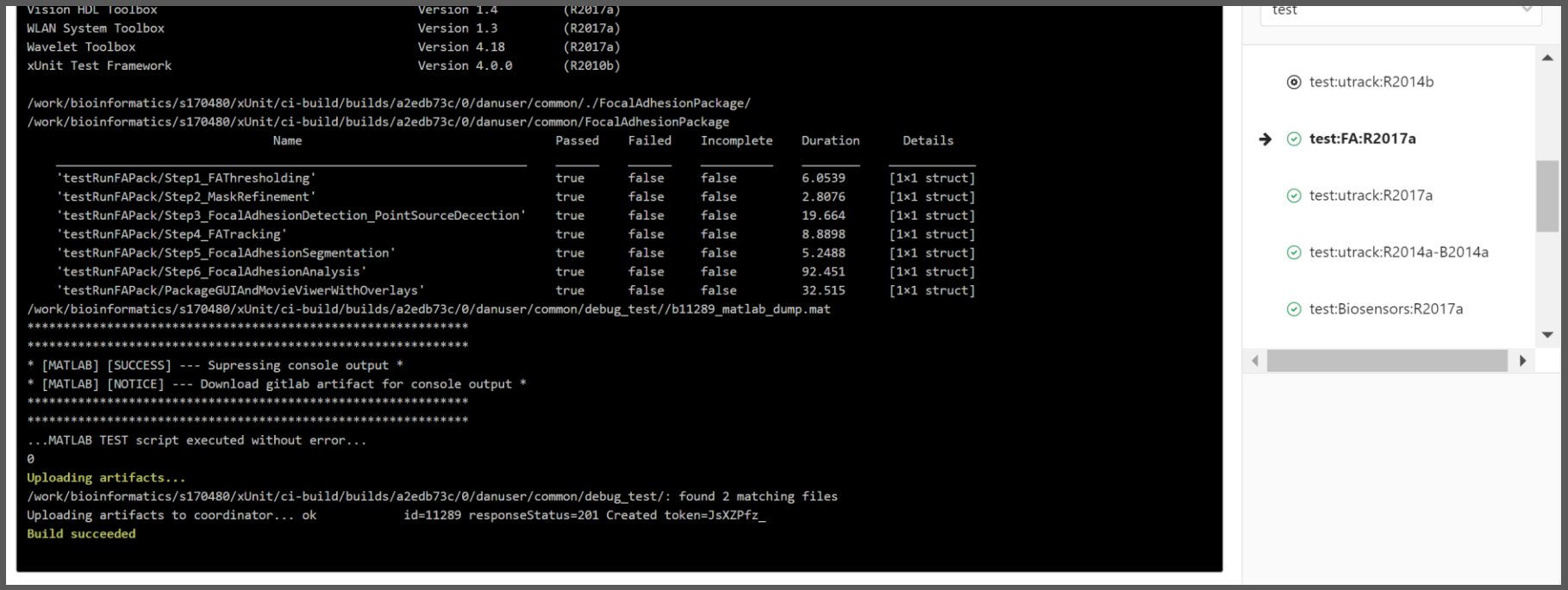
EfficientSubpixelRegistrati | testIndexLSFMTestBasic.m | .gitlab-ci.yml

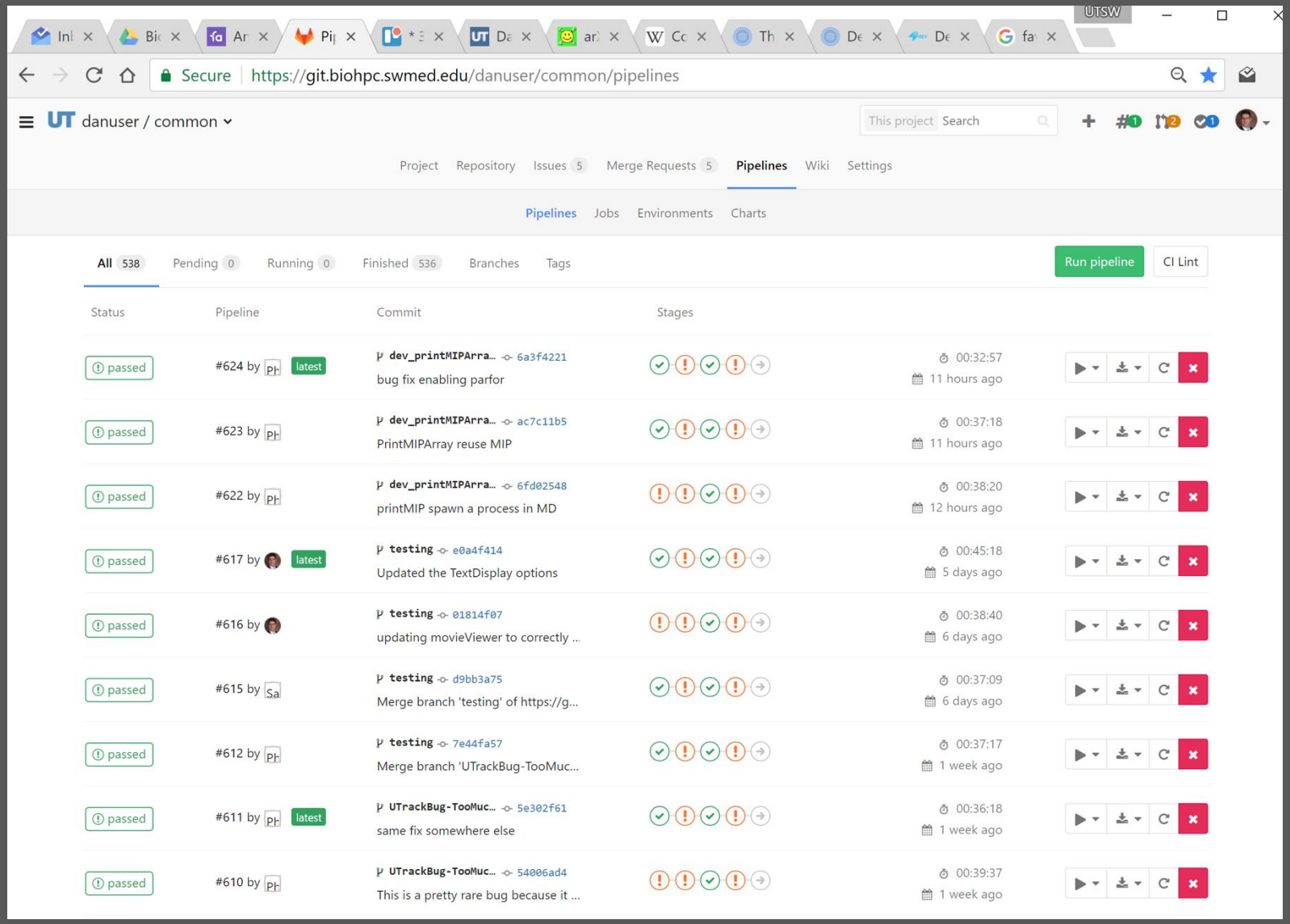
```yaml
# Gitlab CI Script for danuser/common
# Andrew R. Jamieson - 2017
stages:
  - unit
  - integration
  - build
  - test
  - deploy
  - doc
variables:
  CLEAN_REPO_PATH: "/work/bioinformatics/s170480/xUnit/git/${CI_RUNNER_ID}"
  CI_SCRIPT_BRANCH: "master"
  TEST_DEBUG_TARGET_FOLDER: "${CI_PROJECT_DIR}/debug_test/"
before_script:
  - pwd
  - date
  - uname -a
  - CI_SCRIPT_PATH="${CLEAN_REPO_PATH}/ci-scripts/"
  - module load python
  - module unload matlab
  - module list
  - mkdir -p "${CLEAN_REPO_PATH}"
  - cd $CLEAN_REPO_PATH
  - if [ ! -d "$CI_SCRIPT_PATH" ]; then (git clone git@git.biohpc.swmed.edu
  - cd $CI_SCRIPT_PATH
  - pwd
  - git checkout "${CI_SCRIPT_BRANCH}"
  - git pull
  - $CI_SCRIPT_PATH/setup_gitlabCI.sh
  - cd $CI_PROJECT_DIR
  - pwd
```

```
disp('======================================================================');
disp('Running (1st) TFM Process');
disp('======================================================================');
roiMD.getPackage(iPack).createDefaultProcess(1)
params = roiMD.getPackage(iPack).getProcess(1).funParams_;
params.referenceFramePath = refImgPath;
% Get refROI mask with:
% sampleCellImg = imread('Cell_11_w1561 TMR_t001.tif');
% figure, imshow(sampleCellImg,[]), hold on
% h=imrect;
% refROI = wait(h);
params.cropROI = [1249, 1159, 194, 162]; %; refROI;
params.ChannelIndex=[1 2];
params.minCorLength=21;
params.maxFlowSpeed=5;
params.doPreReg=true;
roiMD.getPackage(iPack).getProcess(1).setPara(params);
% Run the stage drift correction
roiMD.getPackage(iPack).getProcess(1).run();


%% Step 2: DisplacementFieldCalculationProcess
disp('======================================================================');
disp('Running (2nd) TFM Process');
disp('======================================================================');
roiMD.getPackage(iPack).createDefaultProcess(2)
params = roiMD.getPackage(iPack).getProcess(2).funParams_;
% Parameters in displacement field tracking
params.referenceFramePath = refImgPath;
params.alpha = 0.05;
params.minCorLength = 19;
params.maxFlowSpeed = 40;
params.highRes = true;
params.mode = 'accurate';
roiMD.getPackage(iPack).getProcess(2).setPara(params);
roiMD.save;
% Run the displacement field tracking
roiMD.getPackage(iPack).getProcess(2).run();


%% Step 3: DisplacementFieldCorrectionProcess
disp('======================================================================');
disp('Running (3rd) TFM Process');
disp('======================================================================');
roiMD.getPackage(iPack).createDefaultProcess(3)
params = roiMD.getPackage(iPack).getProcess(3).funParams_;
roiMD.getPackage(iPack).getProcess(3).setPara(params);
roiMD.getPackage(iPack).getProcess(3).run();
```


Vision HDL Toolbox Version 1.4 (R2017a)
WLAN System Toolbox Version 1.3 (R2017a)
Wavelet Toolbox Version 4.18 (R2017a)
xUnit Test Framework Version 4.0.0 (R2010b)
/work/bioinformatics/s170480/xUnit/ci-build/builds/a2edb73c/0/danuser/common/./FocalAdhesionPackage/
/work/bioinformatics/s170480/xUnit/ci-build/builds/a2edb73c/0/danuser/common/FocalAdhesionPackage
Name Passed Failed Incomplete Duration Details
'testRunFAPack/Step1_FAThresholding' true false false 6.0539 [1×1 struct]
'testRunFAPack/Step2_MaskRefinement' true false false 2.8076 [1×1 struct]
'testRunFAPack/Step3_FocalAdhesionDetection_PointSourceDecection' true false false 19.664 [1×1 struct]
'testRunFAPack/Step4_FATracking' true false false 8.8898 [1×1 struct]
'testRunFAPack/Step5_FocalAdhesionSegmentation' true false false 5.2488 [1×1 struct]
'testRunFAPack/Step6_FocalAdhesionAnalysis' true false false 92.451 [1×1 struct]
'testRunFAPack/PackageGUIAndMovieViwerWithOverlays' true false false 32.515 [1×1 struct]
/work/bioinformatics/s170480/xUnit/ci-build/builds/a2edb73c/0/danuser/common/debug_test//b11289_matlab_dump.mat
****
****
* [MATLAB] [SUCCESS] --- Supressing console output *
* [MATLAB] [NOTICE] --- Download gitlab artifact for console output *
****
****
...MATLAB TEST script executed without error...
0
Uploading artifacts...
/work/bioinformatics/s170480/xUnit/ci-build/builds/a2edb73c/0/danuser/common/debug_test/: found 2 matching files
Uploading artifacts to coordinator... ok id=11289 responseStatus=201 Created token=JsXZPfz_
Build succeeded
test
test:utrack:R2014b
test:FA:R2017a
test:utrack:R2017a
test:utrack:R2014a-B2014a
test:Biosensors:R2017a



https://git.biohpc.swmed.edu/danuser
https://git.biohpc.swmed.edu/danuser/common/-/tree/master/toolfun/test

Danuser Lab + Git + CI/CD Pipeline

Live demo…
Build Artifacts, scripts, shell vs docker, yml

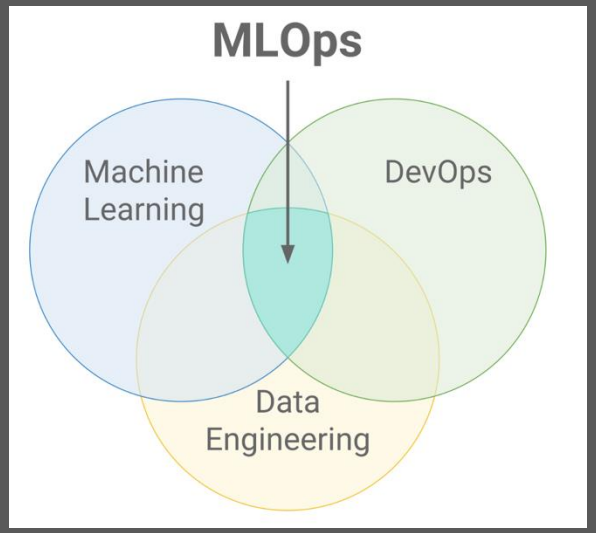
https://git.biohpc.swmed.edu/danuser/common/-/blob/master/.gitlab-ci.yml

GitHub Actions

- https://docs.github.com/en/actions
- In action example:
  - https://github.com/AdvancedImagingUTSW/ASLM
  - https://github.com/utsw-bicf/auto-docker
    - https://github.com/utsw-bicf/auto-docker/blob/develop/.github/workflows/container-ci.yml

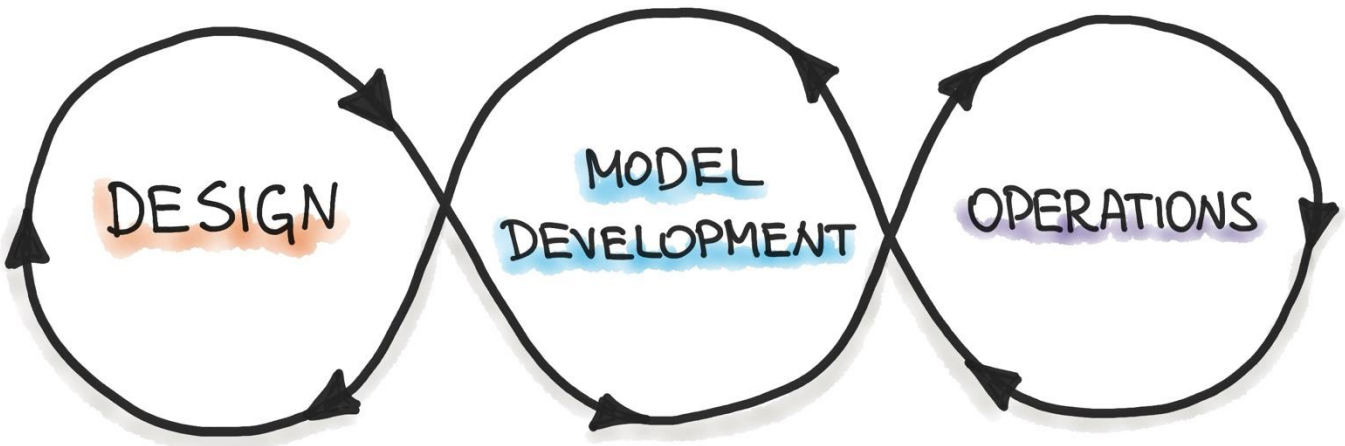
Going beyond DevOps

- ML Experiment Tracking / Model Management
  - W&B, Neptune, MLFlow
- MLOps / "AIOps"
- Data Version Control
  - DVC
- MLOps Goals (Wikipedia):
  - Deployment and automation
  - Reproducibility of models and predictions
  - Diagnostics
  - Governance and regulatory compliance
  - Scalability
  - Collaboration
  - Business uses
  - Monitoring and management



https://wandb.ai/jamiesonlab/StepClassTimeSFormerdivT+S/sweeps/29hgll6s?workspace=user-andrewj_utsw

MLOps

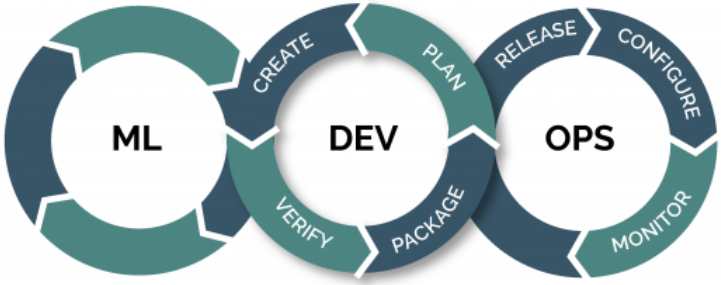


- Requirements Engineering
- ML Use-Cases Priorization
- Data Availability Check

- Data Engineering
- ML Model Engineering
- Model Testing & Validation

- ML Model Deployment
- CI/CD Pipelines
- Monitoring & Triggering

MLOps= ML + DEV + OPS

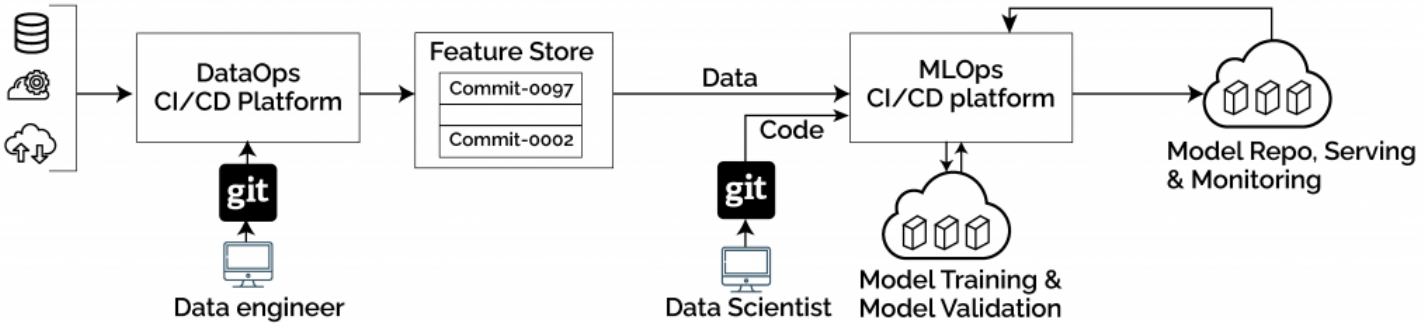


Experiment
Data Acquisition
Business Understanding
Initial Modeling

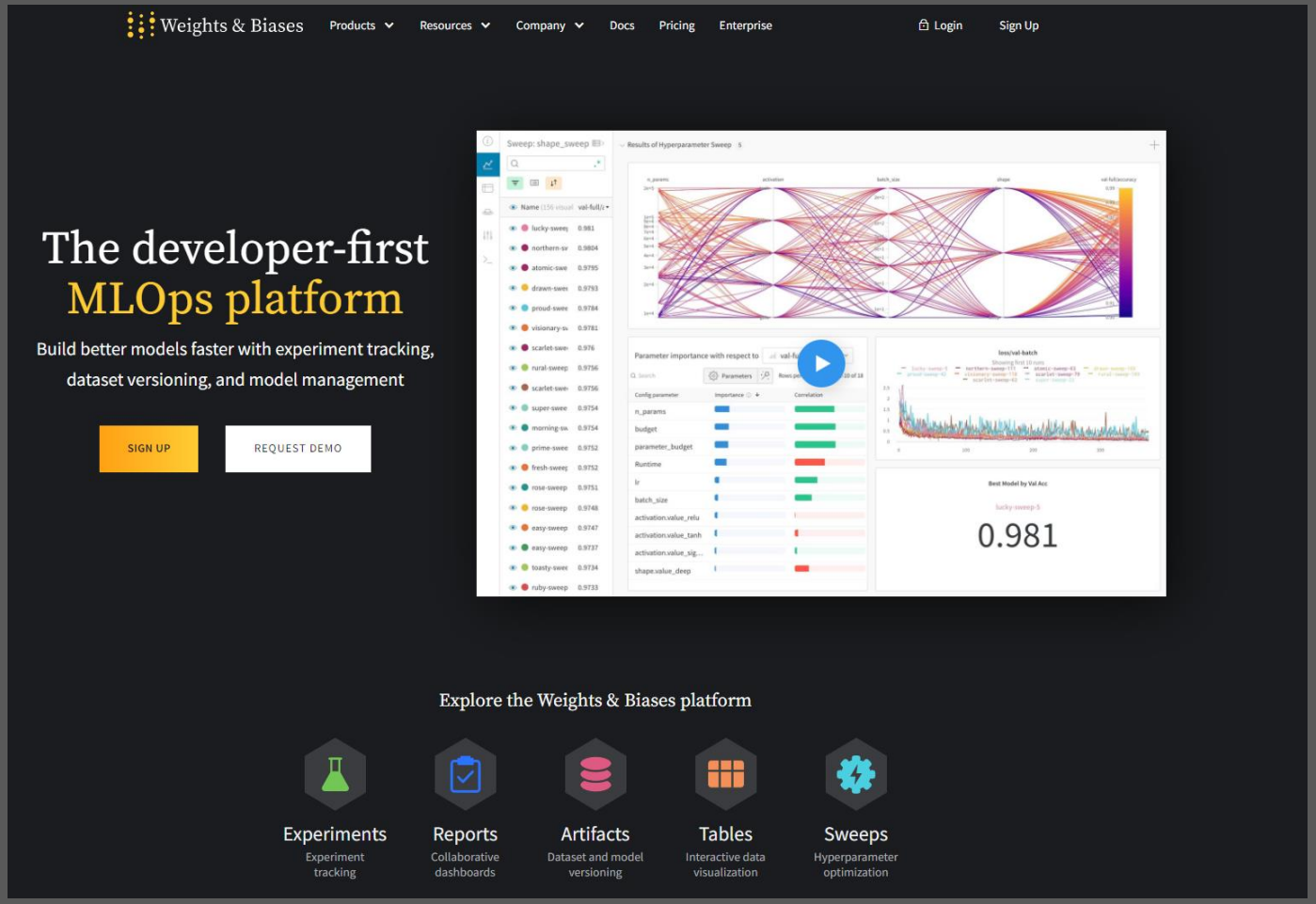
Develop
Modeling + Testing
Continuous Integration
Continuous Deployment

Operate
Continuous Delivery
Data Feedback Loop
System + Model Monitoring

1. Data → 2.Develop/Test Feature Pipelines → 3. Develop Model → 4. Train/Validate Model → 5. Deploy/Monitor



W&B Demo


Weights & Biases
Products
Resources
Company
Docs
Pricing
Enterprise
Login
Sign Up
The developer-first
MLOps platform
Build better models faster with experiment tracking, dataset versioning, and model management
SIGN UP
REQUEST DEMO
Explore the Weights & Biases platform
Experiments
Experiment tracking
Reports
Collaborative dashboards
Artifacts
Dataset and model versioning
Tables
Interactive data visualization
Sweeps
Hyperparameter optimization

TOOLS OF TITANS

DevOps

- Leverage best practices & tools of modern software development
  - Version Control (GIT)
  - Automatic software testing & deployment
    - Continuous Integration
    - Continuous Delivery
  - Agile methodologies

Goal: Accelerate Research Productivity

Code → production

Code → Science

Ideas to Action

Ideas to Results

DevOps for Research ™

Git Notes + Tips/Tricks

- My tools
  - Sublime Text, Sublime Merge, PyCharm, gitk-> git gui, helps to have a powerful IDE
  - Use “git blame”

- Best practices for committing new code components in shared development environments
  - Use of feature branches: Develop distinct components on separate branch, then merge after passing CI tests
  - Branch, commit, test, then merge

- Get creative with git branches to make life easy and avoid merge catastrophes:
  - Creating ***junk*** branches to test merges (both directions)
  - Merge master/main into a “feature” branch

Git command example: merging

- (from “main” branch, merge feature branch)
  - On main* branch
  - git checkout –b mainmerge1
  - git merge myfeature

  [check for any conflicts!]

  [if conflicts, fix!]

  - git checkout main (then fix as necessary)
  - git checkout myfeature (then fix as necessary)

  [if all good, merge for real, but first double check]

  - git checkout –b mainmerge2
  - git merge myfeature (just double check, working properly)

  [now merge for real]

  - git checkout main
  - git merge myfeature

  [done!]

Git command examples: merging

- (from feature branch, merge main)

  On myfeature* branch

  – git checkout –b mytestbranch

  – git merge main

  [check for any conflicts!]

  [if conflicts, fix!]

  - git checkout myfeature

  - git checkout main

  [if good merge for real]

  - git checkout –b mytest2

  - git merge main (double check all good)

  [now, really merge]

  - git checkout myfeaturebranch

  - git merge main

  (then try merging myfeaturebranch into main…)

Exercises

- Develop unit tests for GeneGPT
  - Try to cover all class methods (or other atomic functions)
  - Verify integrity of key computational steps
  - Verify objects/functions behave as expected (creation/input/output)
- Develop integration tests
  - Test compound operations/dependencies
- Develop regression tests for your package
  - For a given set of parameter configurations, verify your package outputs expected values/objects/data/files/plots
- Deploy Automatic Testing [Extra credit]:
  - Use gitlab/github repo actions to automatically evaluate code base after each commit.
  - Create new repo (make public to run matlab github actions)

Built ahead of time, the set of tests can almost act as a design specification for your programs!


DevOps Borat
@DEVOPS_BORAT
Follow
To make error is human. To propagate error to all server in automatic way is #devops.
Reply
Retweet
Favorited


I DON'T ALWAYS TEST MY CODE
BUT WHEN I DO I DO IT IN PRODUCTION


DevOps Borat
@DEVOPS_BORAT
Cultural Learnings of DevOps for Make Benefit Glorious Teams of Devs and Ops.
Kazakhstan

Profiling Your Code

- What does this mean?
- Why?
- How to do it?

Profiling Your Code

- What does this mean?
  - Compute Time
  - System Memory
- Why?
- How to do it?

Profiling Your Code

- What does this mean?
  - Compute Time
  - System Memory
- Why?
  - Uncover unexpected time sinks
  - How to better structure code (serialize/parallelize, batch/group segments)
  - Remove non-value add code, throw-away commands
  - Plotting/graphics issues
  - Small inefficiencies become big with scale
  - More rapid feedback loop for results
- How to do it?
  - Review tool: MATLAB Profiler

Guidance & Documentation

- Where to start?
  - https://www.mathworks.com/help/matlab/software-development.html
  - Lots of great resources/examples from Mathworks

Guidance & Documentation

- Where to start?
  - https://www.mathworks.com/help/matlab/software-development.html
  - Lots of great resources/examples from Mathworks

(Step 0: Pseudo-code)

- First step: get code working!
  - Use IDE & tools for:
    - debugging
    - code analysis (“linting”, style/syntax, warnings/errors, suggestions)

Profiling Your Code

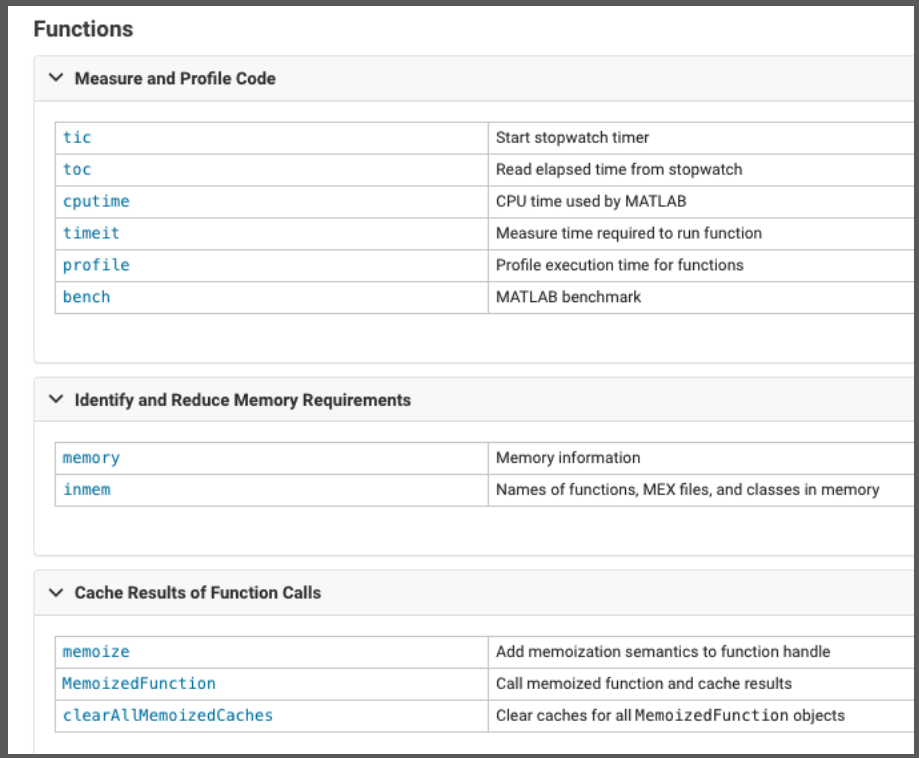
- https://www.mathworks.com/help/matlab/performance-and-memory.html
- "Write your code to be **simple and readable**, *especially for the first implementation*. "

- "Code that is prematurely optimized can be unnecessarily complex without providing a significant gain in performance. Then, if speed is an issue, you can measure how long your code takes to run and profile your code to identify bottlenecks. "

- "If necessary, you can take steps to improve performance."

MATLAB Code Profiling Tools & Tips

- Profiler App
- Functions



Functions

Measure and Profile Code

`tic`	Start stopwatch timer
`toc`	Read elapsed time from stopwatch
`cputime`	CPU time used by MATLAB
`timeit`	Measure time required to run function
`profile`	Profile execution time for functions
`bench`	MATLAB benchmark

Identify and Reduce Memory Requirements

`memory`	Memory information
`inmem`	Names of functions, MEX files, and classes in memory

Cache Results of Function Calls

`memoize`	Add memoization semantics to function handle
`MemoizedFunction`	Call memoized function and cache results
`clearAllMemoizedCaches`	Clear caches for all `MemoizedFunction` objects

MATLAB Code Profiling Tools & Tips

Measure the Performance of Your Code

R2022a

Overview of Performance Timing Functions

The `timeit` function and the stopwatch timer functions, `tic` and `toc`, enable you to time how long your code takes to run. Use the `timeit` function for a rigorous measurement of function execution time. Use `tic` and `toc` to estimate time for smaller portions of code that are not complete functions.

For additional details about the performance of your code, such as function call information and execution time of individual lines of code, use the MATLAB® Profiler. For more information, see Profile Your Code to Improve Performance.

Time Functions

To measure the time required to run a function, use the `timeit` function. The `timeit` function calls the specified function multiple times, and returns the median of the measurements. It takes a handle to the function to be measured and returns the typical execution time, in seconds. Suppose that you have defined a function, `computeFunction`, that takes two inputs, `x` and `y`, that are defined in your workspace. You can compute the time to execute the function using `timeit`.

```
f = @() myComputeFunction(x,y); % handle to function
timeit(f)
```

Time Portions of Code

To estimate how long a portion of your program takes to run or to compare the speed of different implementatio functions, `tic` and `toc`. Invoking `tic` starts the timer, and the next `toc` reads the elapsed time.

```
tic
   % The program section to time.
toc
```

```
tic
A = rand(12000,4400);
B = rand(12000,4400);
toc
```

```
Elapsed time is 0.867440 seconds.
```

Sometimes programs run too fast for `tic` and `toc` to provide useful data. If your code is faster than 1/10 second, consider measuring it running in a loop, and then average to find the time for a single run.

The cputime Function vs. tic/toc and timeit

It is recommended that you use `timeit` or `tic` and `toc` to measure the performance of your code. These functions return wall-clock time. Unlike `tic` and `toc`, the `timeit` function calls your code multiple times, and, therefore, considers first-time costs.

MATLAB Code Profiling Tools & Tips

- Code Structure
- https://www.mathworks.com/help/matlab/matlab_prog/techniques-for-improving-performance.html

Environment

Be aware of background processes that share computational resources and decrease the performance of your MATLAB® code.

Code Structure

While organizing your code:

- Use functions instead of scripts. Functions are generally faster.
- Prefer local functions over nested functions. Use this practice especially if the function does not need to access variables in the main function.
- Use modular programming. To avoid large files and files with infrequently accessed code, split your code into simple and cohesive functions. This practice can decrease first-time run costs.

Programming Practices for Performance

Consider these programming practices to improve the performance of your code.

- Preallocate — Instead of continuously resizing arrays, consider preallocating the maximum amount of space required for an array. For more information, see Preallocation.
- Vectorize — Instead of writing loop-based code, consider using MATLAB matrix and vector operations. For more information, see Vectorization.
- Place independent operations outside loops — If code does not evaluate differently with each `for` or `while` loop iteration, move it outside of the loop to avoid redundant computations.
- Create new variables if data type changes — Create a new variable rather than assigning data of a different type to an existing variable. Changing the class or array shape of an existing variable takes extra ti
- Use short-circuit oper … l only when the result is not fully dete
- Avoid global variables
- Avoid overloading bu
- Avoid using "data as … and saving them, for example, in a MA

Tips on Specific MATLAB Functions

Consider the following tips on specific MATLAB functions when writing performance critical code.

- Avoid clearing more code than necessary. Do not use `clear all` programmatically. For more information, see `clear`.
- Avoid functions that query the state of MATLAB such as `inputname`, `which`, `whos`, `exist(var)`, and `dbstack`. Run-time introspection is computationally expensive.
- Avoid functions such as `eval`, `evalc`, `evalin`, and `feval(fname)`. Use the function handle input to `feval` whenever possible. Indirectly evaluating a MATLAB expression from text is computationally expensive.
- Avoid programmatic use of `cd`, `addpath`, and `rmpath`, when possible. Changing the MATLAB path during run time results in code recompilation.

MATLAB Code Profiling Tools & Tips

Preallocation & Vectorization

- https://www.mathworks.com/help/matlab/matlab_prog/techniques-for-improving-performance.html

Preallocation

R2022a

`for` and `while` loops that incrementally increase the size of a data structure each time through the loop can adversely affect performance and memory use. Repeatedly resizing arrays often requires MATLAB® to spend extra time looking for larger contiguous blocks of memory, and then moving the array into those blocks. Often, you can improve code execution time by preallocating the maximum amount of space required for the array.

The following code displays the amount of time needed to create a scalar variable, `x`, and then to gradually increase the size of `x` in a `for` loop.

```
tic
x = 0;
for k = 2:1000000
   x(k) = x(k-1) + 5;
end
toc
```

```
Elapsed time is 0.301528 seconds.
```

```
tic
x = zeros(1,1000000);
for k = 2:1000000
   x(k) = x(k-1) + 5;
end
toc
```

```
Elapsed time is 0.011938 seconds.
```

MATLAB Code Profiling Tools & Tips

Preallocation & Vectorization

- https://www.mathworks.com/help/matlab/matlab_prog/techniques-for-improving-performance.html

Using Vectorization

MATLAB® is optimized for operations involving matrices and vectors. The process of revising loop-based, scalar-oriented code to use MATLAB matrix and vector operations is called *vectorization*. Vectorizing your code is worthwhile for several reasons:

- *Appearance*: Vectorized mathematical code appears more like the mathematical expressions found in textbooks, making the code easier to understand.
- *Less Error Prone*: Without loops, vectorized code is often shorter. Fewer lines of code mean fewer opportunities to introduce programming errors.
- *Performance*: Vectorized code often runs much faster than the corresponding code containing loops.

Vectorizing Code for General Computing

This code computes the sine of 1,001 values ranging from 0 to 10:

```
i = 0;
for t = 0:.01:10
    i = i + 1;
    y(i) = sin(t);
end
```

This is a vectorized version of the same code:

```
t = 0:.01:10;
y = sin(t);
```

Array Operations

Array operators perform the same operation for all
suppose you collect the volume (V) of various cone
calculate the volume for that single cone:

```
V = 1/12*pi*(D^2)*H;
```

Now, collect information on 10,000 cones. The vect
languages, you need to set up a loop similar to this

```
for n = 1:10000
   V(n) = 1/12*pi*(D(n)^2)*H(n);
end
```

With MATLAB, you can perform the calculation for

```
% Vectorized Calculation
V = 1/12*pi*(D.^2).*H;
```

MATLAB Profiler App

profile

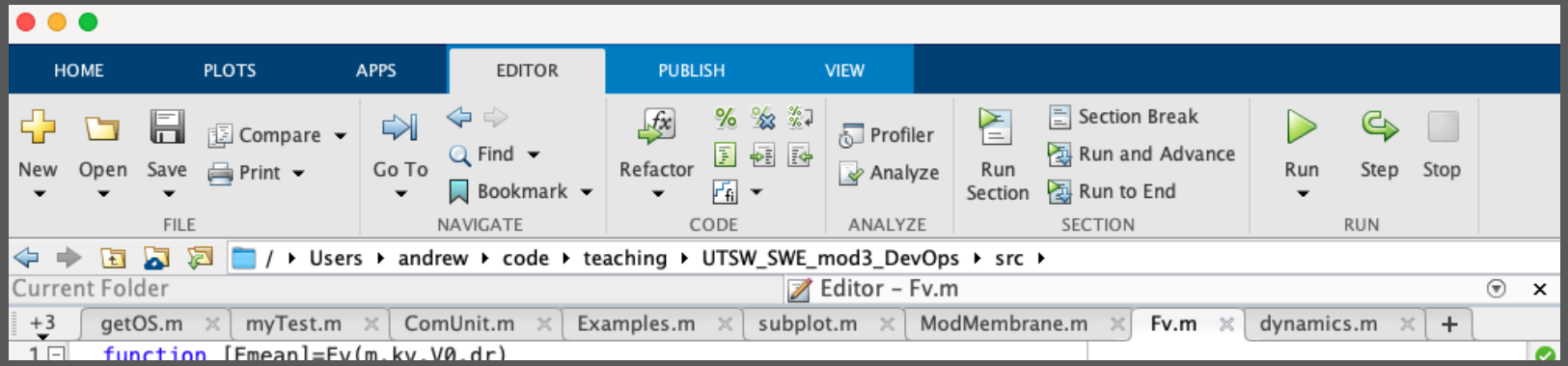
Profile execution time for functions

Syntax

```
profile action
profile action option1 ... optionN
profile option1 ... optionN

p = profile('info')

s = profile('status')
```



MATLAB Profiler App

Profile Your Code to Improve Performance

R2022a

What Is Profiling?

Profiling is a way to measure the time it takes to run your code and identify where MATLAB® spends the most time. After you identify which functions are consuming the most time, you can evaluate them for possible performance improvements. You also can profile your code to determine which lines of code do not run. Determining which lines of code do not run is useful when developing tests for your code, or as a debugging tool to help isolate a problem in your code.

You can profile your code interactively using the MATLAB Profiler or programmatically using the `profile` function. For more information about profiling your code programmatically, see `profile`. If you are profiling code that runs in parallel, for best results use the Parallel Computing Toolbox™ parallel profiler. For details, see Profiling Parallel Code (Parallel Computing Toolbox).

Tip

Code that is prematurely optimized can be unnecessarily complex without providing a significant gain in performance. Make your first implementation as simple as possible. Then, if speed is an issue, use profiling to identify bottlenecks.

Profile Your Code

To profile your code and improve its performance, use this general process:

1. Run the Profiler on your code.
2. Review the profile summary results.
3. Investigate functions and individual lines of code.

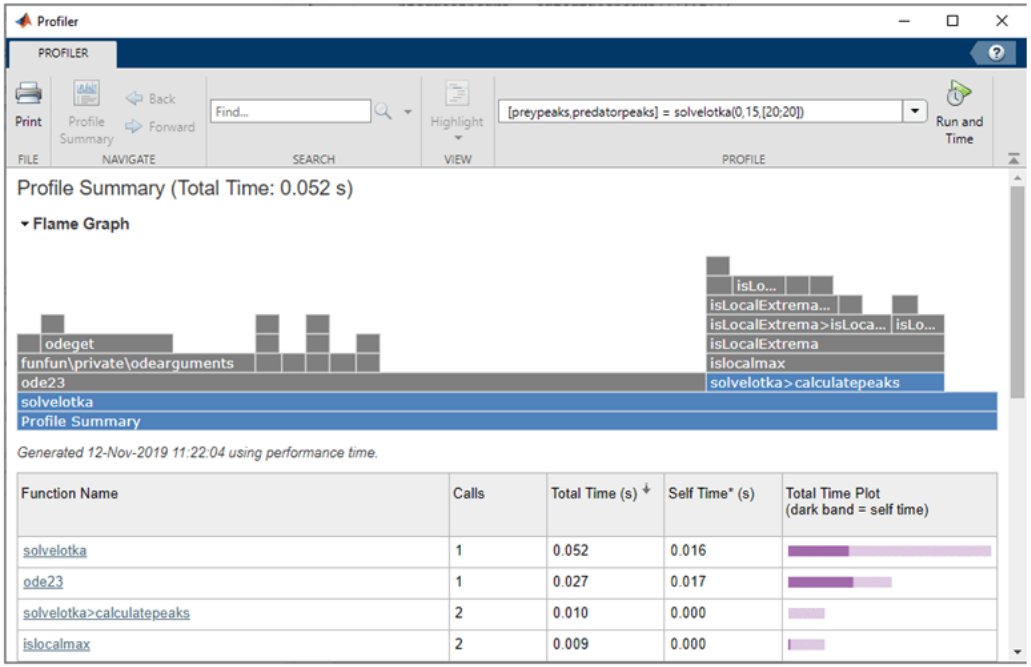
   For example, you may want to investigate functions and lines of code that use a significant amount of time or that are called most frequently.
4. Save the profiling results.
5. Implement potential performance improvements in your code.

   For example, if you have a `load` statement within a loop, you might be able to move the `load` statement outside the loop so that it is called only once.
6. Save the files, and run `clear all`. Run the Profiler again and compare the results to the original results.
7. Repeat the above steps to continue improving the performance of your code. When your code spends most of its time on calls to a few built-in functions, you have probably optimized the code as much as possible.

MATLAB Profiler App

Review the Profile Summary Results

After running the Profiler on your code, the Profile Summary presents statistics about the overall execution of your code and provides summary st
image below shows the Profile Summary for the `solvelotka` function.



Performance Analysis of Tests

- Use MATLAB tools to statistically sample run time of key tests.

runperf

Run set of tests for performance measuren

Syntax

```
results = runperf
results = runperf(tests)
results = runperf(tests,Name,Value)
```

Description

`results = runperf` runs all the tests in your current f
`matlab.perftest.TimeResult` objects. Each elemer

```
>> rr.Samples

ans =

  256×7 table
```

Name	MeasuredTime	Timestamp	Host	Platform
TestExampleRBC/testComUnitCreate	0.0030541	18-May-2022 14:57:09	TexMBP.local	maci64
TestExampleRBC/testComUnitCreate	0.001282	18-May-2022 14:57:09	TexMBP.local	maci64
TestExampleRBC/testComUnitCreate	0.00098066	18-May-2022 14:57:09	TexMBP.local	maci64
TestExampleRBC/testComUnitCreate	0.0043448	18-May-2022 14:57:09	TexMBP.local	maci64
TestExampleRBC/testComUnitCreate	0.0022615	18-May-2022 14:57:09	TexMBP.local	maci64
TestExampleRBC/testComUnitCreate	0.0023379	18-May-2022 14:57:10	TexMBP.local	maci64
TestExampleRBC/testComUnitCreate	0.0015005	18-May-2022 14:57:10	TexMBP.local	maci64
TestExampleRBC/testComUnitCreate	0.0012975	18-May-2022 14:57:10	TexMBP.local	maci64
TestExampleRBC/testComUnitCreate	0.0032864	18-May-2022 14:57:10	TexMBP.local	maci64

```
>> rr.sampleSummary

ans =

  2×7 table
```

Name	SampleSize	Mean	StandardDeviation
TestExampleRBC/testComUnitCreate	256	0.0029934	0.0019167
TestExampleRBC/testComUnitCreateError	140	0.023993	0.0085297

```
rr = runperf('./tests/TestExampleRBC.m', 'Name', '*ComUnit*');
```