A decorative graphic on the left side of the slide consisting of two overlapping parallelograms. The front one is blue and the back one is a light green. They are positioned diagonally, with the blue one partially covering the green one.

SE Best Practices in Three Hours

(Leveraging GitHub)



Who am I?

- Have a degree in “Engineering and Applied Science” from Caltech
 - No Computer Science degree existed yet!
- Spent 40 years in industry (primarily at Xerox) in research and development
 - Research project with 3 software engineers
 - Large project – 10 million lines of code, of which my group was responsible for over 3 million
 - Small prototype projects
- Sr. Software Engineer at Caltech’s Schmidt Academy for Software Engineering
 - SASE is an experiment intended to address the fact that much of the software developed for research is written by people with little or no CS training
 - We take recent graduates with CS degrees and strong STEM background, and embed them in research labs at Caltech who submit project proposals
 - I mentor half the cohort, and work on a project myself
 - Wide variety of projects; I’ve worked on astronomy, neuroscience and applied physics projects
 - Including one with Ann Kennedy, which is why I’m here

-



Prerequisites

- GitHub account and basic familiarity with git and GitHub
- Access to GitHub Classroom `jax-2023-oct-se-best-practices`
- (Hopefully these were done before this week!)



In three hours you will have...

- A basic understanding of Agile software engineering principles
- Captured SW requirements using GH issues
- Prioritized what to work on next using MoSCoW
- Used git branches to organize SW development
- Used GH Pull Requests to coordinate adding new capabilities to the main branch
- Understood the value of and participated in a Code Review and Pair Programming
- Automated “linting” and testing using GitHub Actions
- Gotten to dinner on time!



Phases of Software Development

[Most of] the phases of software development:

- Determine software requirements
- Software design
- Implementation
- Software testing
- Software documentation
- Release/deploy the software to the customer
- Ongoing maintenance

These phases can be mixed, or (sometimes) reordered

- Line between requirements and design is often blurry
- Implementation, testing and user docs often proceed concurrently
- Test Driven Development (TDD): Tests are written first



Agile?

Goal: eliminate waste wherever possible, especially *changing requirements or plans*

- This is especially true for research software
- Do work incrementally, in “iterations” (aka “sprints”)
 - Decide what is essential for the next iteration
 - Design the new feature(s)
 - Develop, test and document the new feature(s)

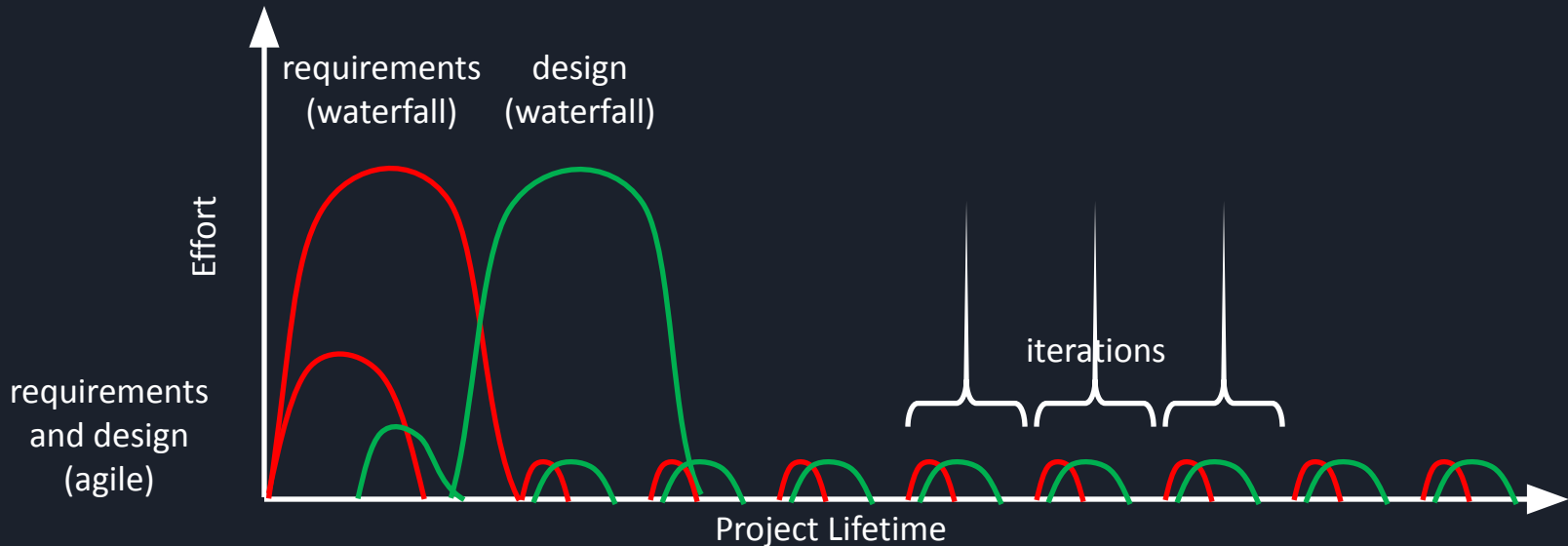
Anatomy of an Agile project:

- Project visioning
 - What are the goals of this project?
- Capture requirements as User Stories
 - “As a User, I can visualize differences in my data before and after processing”
 - “As an Administrator, I can add access to a new User”
- Multiple development iterations
- Interaction with “stakeholders” early and often

Agile vs. Waterfall

Agile does requirements gathering and design incrementally

Overall, may be more costly than a well-executed waterfall approach, *as long as requirements and design don't change!*





Anatomy of an Iteration

Duration

- 2-4 weeks, depending on the project

Planning Meeting

- No more than one hour per week of development time
- “Groom” backlog
- Use “MoSCoW” rating to choose which new work to undertake
- Flesh out approach/design options
- Capture more detailed tasks as GitHub Issues

Development Phase

- Each software engineer (or pair):
 - Selects a suitable task to work on
 - Creates a feature branch in git
 - Develops, debugs and tests the new code
 - Commits the code to the branch and pushes the branch to the repo
 - Opens a “Pull Request” to merge in the new code
- Work on “Must Haves” first
- “Should Haves” if time permits
- Status reporting
 - 1 - 3 times per week
 - Short meeting (“Stand-up”)
 - What was done
 - What is planned
 - Barriers

Post-Iteration Review

- What went well/badly/lessons learned
- Often combined with Planning Meeting for next iteration



Activity #1: Iteration Planning Meeting (15 min)

If you haven't already:

- Pair up
- Clone the repo
 - https://github.com/jax-2023-oct-se-best-practices/ink_limiting/settings
- Install the required python modules:
 - `pip install -r requirements.txt`

Planning Meeting:

- Create “MoSCoW” issue labels
- Create an Issue for each project requirement
- Take a quick look at the existing code
- Rank Issues using MoSCoW

Requirements:

- As a user, I can limit the ink in TIFF files from a command line
- As a user, I can process multiple files in a batch from a GUI
- As a user, I can select between proportional and undercolor removal methods
- As a user, I can before and after preview images side by side
- As a user, I don't have to wait longer than 5 seconds for a typical image to process



Git Branches

Using git branches...

- Protects the code base from incomplete, buggy or untested changes
- Allows you to easily try things/prototype approaches
- Automates easy merges
 - (You still need to manually complete merges when two people have changed the same file in incompatible ways.)

To create a new branch:

- `git branch <branchname>`

To list existing branches:

- `git branch -l`

To switch to an existing branch:

- `git checkout <branchname>`

To create a new branch and switch to it:

- `git checkout -b <branchname>`



Git Workflows

- There are lots of different ways to use git branches
- If you work on a large project, someone has probably already decided how it will use branches
- If not, ask these questions:
 - Are there currently any users of the code?
 - Are different people separately working on new features or capabilities?
- If there are current users, you need a development branch, typically named “develop”
- If different people are working separately, each needs to create and use a “feature branch”
- It’s best to use a GitHub “Pull Request” (PR) to control the merging back into “develop” and “main”
 - Better control over the quality of code being merged
 - Can require reviews and passage of testing (stay tuned) before merge is allowed

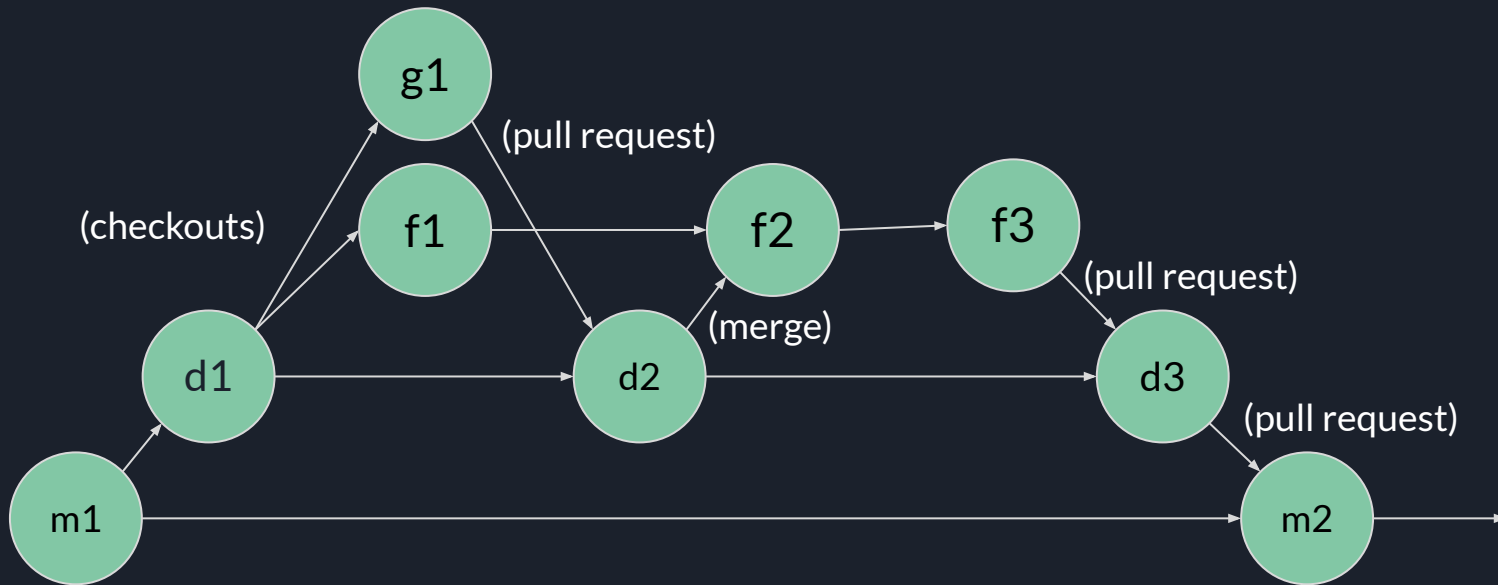
Common Workflow

her_feature

my_feature

develop

main





Collaborative Construction (Code Reviews and Pair Programming)

- Debugging and Testing are **not** the best ways to find software defects
- Design- and code-reviews are **much** better!
- These are called **collaborative construction techniques**
- Two widely-used techniques:
 - Code reviews
 - Pair Programming

Other benefits beyond finding defects::

- Peer-pressure to maintain high standards
- Facilitates knowledge transfer among teammates
 - Helps more people understand how the project's code works
- Fosters healthy team dynamics
 - E.g. reduces feelings of “my code”
increases feelings of “our code”



Code Reviews

Two Categories:

- Formal Review
 - Advanced preparation expected
 - May follow standardized checklist
 - Often led by someone other than the author
- Code Walkthrough
 - Generally author-led
 - May only review changes (“diffs”)

Goals (Google):

- Design: well-designed and appropriate?
- Functionality: behaves as user expects?
- Complexity: could it be simpler?
- Tests: correct and well-designed?
- Naming: clear and useful?
- Comments: clear and useful?
- Style: follow the style guide?
- Documentation: updated?



Best Practices

- Courteous and respectful feedback
- Objective, constructive, neutral language
 - Critique the code, not the programmer
- Ask questions, seek understanding
 - Maybe there's a good reason it was done that way
- Explain clearly and completely



Pair Programming

Two developers work actively together to write the code:

- The ***Driver*** actually controls the keyboard and writes the code
- The ***Navigator*** observes what the driver is doing, asks questions, evaluates the work, pointing out potential issues
- Periodically, at reasonable points, they exchange roles

Benefits:

- An opportunity to find defects at the earliest possible moment
 - The longer a defect goes undetected, the more costly it is to fix
- Particularly good when the two bring complementary expertise, e.g. scientist and software engineer
- Tests to produce higher-quality code
- Improved satisfaction and team dynamics
 - Most people prefer working on things together
- Improved productivity



Pair Programming Guidance

For Drivers:

- Discuss design approach options up front
- “Think out loud” as you program
 - Invite comment and discussion
- After writing, spend time brainstorming how to test
- Give up the “steering wheel” at suitable points

For Navigators:

- If you see a possible issue, mention it!
- Consider the bigger picture
 - Does the code’s control flow handle all possibilities correctly?
 - Does the code fully satisfy the requirements
- Feel free to suggest structural/style improvements
 - Avoid nit-picking



Tools for Collaborative Construction

Static code analysis tools for Python:

- flake8 - coding style checking
- pylint - coding style and linting
- black - coding style enforcement
- mypy - check Python code against type annotations

Tools for Remote (or Local) collaboration:

- For VSCode users:
 - Live Share extension from Microsoft
- For PyCharm (and other JetBrains IDE) users:
 - Code With Me plug-in
- Generic:
 - Zoom (or equivalent) shared desktop



Activity #2: Pair Programming (30 minutes)

In your teams:

- Pair-program the first “must have” task you previously identified
- Work on the second “must have” if time permits
- I have supplied you with a set of tests in `test_inklimit.py` that should pass if you are successful
 - Run “`pytest`” from the terminal in your working directory

Hints about ink limiting:

- The image data in `balloons.tiff` is 2832 by 4256 pixels, where each pixel is 4 values (cyan, magenta, yellow and black)
 - I.e. array shape is (2832, 4256, 4)
- You can reduce them proportionally to get under the ink limit
- “Undercolor removal” (UCR) refers to the idea that equal values of C, M & Y can be replaced by an equivalent value of black (K)



Activity #3: Code Review

With one (or two) other pairs:

- Select which pair's code you will review
- Conduct a quick code walkthrough, focusing on the differences from the version you started with
- Did the teams approach the task in similar ways?

If you have time:

- Try running `flake8` and `mypy` on your code



Testing

Types of tests

- Regression tests
 - Make sure that capabilities that once worked continue to work in the face of continued development
- Validation tests
 - Make sure that a capability does what it's designed to do
- Performance tests
 - Make sure that an operation takes no more time than it's supposed to take

Testing Scope

- Unit tests
 - Test a capabilities of a single module in isolation
 - Should be fast and as thorough as possible
- Integration tests
 - Test cross-module interactions
 - Fewer in number, take longer
 - May require simulations of external resources, e.g. databases
- End-to-end tests
 - Test complete functionality
 - Few in number, take longer yet



Testing (2)

Testing Frameworks

- Aim to make tests easy to construct and run
- Centered around “assertions”
- In python, pytest and unittest frameworks are commonly used
 - Different, but mutually compatible
 - If you’re just starting, I recommend pytest
- Create “test_<module>.py” for each module
- pytest will automatically run all functions named test_*

Test setup and teardown


- You can write separate functions that are used by several tests, e.g. set up and substitutes for external components
- See test_inklimit.py in your repo as an example of unit and end-to-end tests



Test Automation in GitHub

GitHub Actions

- Can automatically do a number of useful things
 - Packaging
 - Creation of documentation
 - **Testing**
- Based on “trigger” events
 - Pull Request, etc.
- Runs a configured script written in YAML
- Test runs on GitHub’s servers by default
 - Servers for Linux, Windows and MacOS are available
 - Can be configured to run on your own server
- Results are reported in a Dashboard and e-mailed



Activity #4: Set up automated testing (20 minutes)

Try your tests locally...

- Install pytest, if you haven't already
 - `pip install pytest`
- Run pytest on your code
 - `pytest`
 - All tests should pass ;-)

Set up a GitHub Action to run testing...

- On GitHub for your repo, select Actions
- Find the “Python Application” template and click “Configure”
- Look through the script and note:
 - Triggered by Pull Request or Push to the “main” branch
 - Runs tests on ubuntu (Linux)
 - Runs the flake8 linter and pytest
- Click “Commit Changes...” on upper right
- Since the commit was to the main branch, it triggered a test run
 - Click on Actions again to monitor the automated test