Lecture 6: October 2

Git, PRs, CI/CD

Agenda

- September Sprint Feedback
- October Sprint Planning
- Git / PR Reviews
- CI/CD
- Presentation 1
- For next week
- Team Charter

Agenda

- **September Sprint Feedback**
- October Sprint Planning
- Git / PR Reviews
- CI/CD
- Presentation 1
- For next week
- Team Charter

September Sprint Grading Criteria

Total Sprint Progress: 20% (September: 4%)

- Sprint Board
  - Tickets created for class assignments + project requirements
  - Tickets addressed as “done”, “won’t do”, or moved to next sprint
- Weekly Status Updates
  - Status update is posted weekly and on time
- Assignments (student info form, resume, project proposal, etc)
  - Assignments submitted on time

September Sprint Feedback

- ~50% of students are consistently posting weekly updates
- ~75% of students are keeping up with sprint boards
  - Some created project-specific tickets
  - Some left comments!
- Majority of students submitted assignments on time

Takeaway: Keep up with status updates, you have 2 more sprints of work

Agenda

- September Sprint Feedback
- **October Sprint Planning**
- Git / PR Reviews
- CI/CD
- Presentation 1
- For next week
- Team Charter

Month	Expected Status	Monthly Focus	Deliverables
September	N/A	- Figure out teams - Brainstorm projects	- Create teams - resume
Mid-September	- Teams selected - Handful of project ideas	- Final project selection - Begin meeting w/ mentors	- Project proposal - Hardware/software request - Writing: Executive Summary
October	- Project selected & approved	- Begin technical investigations (services, apis, language, etc) - Flesh out project functionality & requirements - Coding should start (scaffolding, ci/cd, prototyping)	- Writing: Technical summary - Presentation: Elevator pitch
November	- Main technologies selected - project is well-defined - Everyone is actively coding	- Answer all questions needed to complete TDD - Lot's of coding for alpha demo	- Writing: PRD - Presentation: Project Design
December	- Code complete for alpha demo	- more coding for beta demo - Formalize design discussions into proper TDD	- Presentation: Alpha prototype - Writing: TDD
January	- Continued focus on project development	- continued development for beta demo - focus on proper testing & integration	- Website Design
February	- Code complete for beta demo	- Refine code from a prototype into a fleshed out project -- testing, integration, polishing - continued development for prelim prototype (get as close to finished as you can here)	- Presentation: Beta prototype - Presentation: Elevator pitch/promotional
March	- Code complete for prelim demo	- final code polishing to wrap up project - complete any necessary integration work - add extra features if possible	- Presentation: Pelim prototype
April	**- Code 99% complete for final demo**	- finishing touches for final project submission - ideally you are done with coding by this point	- Presentation: Final demo - Promotional video
May			- Final package due

Month	Expected Status	Monthly Focus	Deliverables
September	N/A	- Figure out teams - Brainstorm projects	- Create teams - resume
Mid-September	- Teams selected - Handful of project ideas	- Final project selection - Begin meeting w/ mentors	- Project proposal - Hardware/software request - Writing: Executive Summary
October	- Project selected & approved	- Begin technical investigations (services, apis, language, etc) - Flesh out project functionality & requirements - Coding should start (scaffolding, ci/cd, prototyping)	- Writing: Technical summary - Presentation: Elevator pitch
November	- Main technologies selected - project is well-defined - Everyone is actively coding	- Answer all questions needed to complete TDD - Lot's of coding for alpha demo	- Writing: PRD - Presentation: Project Design
December	- Code complete for alpha demo	- more coding for beta demo - Formalize design discussions into proper TDD	- Presentation: Alpha prototype - Writing: TDD
January	- Continued focus on project development	- continued development for beta demo - focus on proper testing & integration	- Website Design
February	- Code complete for beta demo	- Refine code from a prototype into a fleshed out project -- testing, integration, polishing - continued development for prelim prototype (get as close to finished as you can here)	- Presentation: Beta prototype - Presentation: Elevator pitch/promotional
March	- Code complete for prelim demo	- final code polishing to wrap up project - complete any necessary integration work - add extra features if possible	- Presentation: Pelim prototype
April	**- Code 99% complete for final demo**	- finishing touches for final project submission - ideally you are done with coding by this point	- Presentation: Final demo - Promotional video
May			- Final package due

Sprint Goals

September Sprint: What problems do we want to solve?

- Project definition
- Technical & algorithmic requirements

October Sprint: What solutions will solve these problems?

- What language
  - Front end or backend
  - iOS or Android
  - Web App or Mobile App
- What algorithms
  - What algorithms am I building?
  - What algorithmic theory applies here?
- What APIs
  - What libraries, databases, or programs do I need to connect to in order to build my solution?
  - API Documentation - good example of technical documentation

October Schedule

Date	Lab	Assignments
10/2	Git, PRs, CI/CD, Team Charter	Writing 1 (10/6)
10/9	Writing 1 feedback, Project Design & UX	Presentation 1 (10/16)
10/16	Presentation 1	Project Website (10/20)
10/23	NO LAB (focus time)	Writing 2 (10/27)
10/30	REMOTE LAB – **team progress review**	Presentation 2 (11/6)

Week of 11/3: “Demo 0” (individual progress check-in w/ instructor)

October Sprint Progress Rubric

Fall Semester

Full credit

- Tickets addressed as either “done”, “won’t do”, or moved to next sprint.
- Weekly standup updates & slack participation
- Code is PRed & merged to master. Branches & PRs are well-scoped. PRs are linked to tickets.

Partial credit

- Majority of tickets addressed as either “done”, “won’t do”, or moved to next sprint.
- Occasional standup updates & moderate participation
- Code is committed, PRs are sometimes present and sometimes well-scoped. PRs are sometimes linked to tickets.

Minimal credit

- Few tickets addressed as either “done”, “won’t do”, or moved to next sprint.
- Minimal standup updates & rare participation
- Minimal code is committed, PRs are missing or not well-scoped.

No credit

- No sprint board activity
- No standup updates
- No slack participation
- No code committed to master/main

Expectations: Sprint Board

- Create tickets to capture class assignments (writings, presentations, etc)
- Create tickets to capture project-specific work
  - Create project-specific epics to organize work
- Tickets should include:
  - Descriptions
  - Assignees
  - Due dates
  - Sprint
  - Status
  - Linked PR (when there is code)
- **All tickets should be completed, moved to next sprint, or marked as “won’t do” by the end of the sprint**

Expectations: Weekly Status Updates

- Create a new status update ticket for each week
  - Title should be **Status Update - Week of MM/YY** with the date matching the Sunday date on the course website
  - Due date should be **Wednesday** (this is a change from September!)
  - Epic should be **status update**
- Move ticket from TODO to DONE as week progresses
- Students should post weekly status updates covering:
  - What they completed (can link to other tickets)
  - What they are blocked by
  - What they are currently working on
  - **Each student must leave their own comment (do not update the description) before the due date to receive full credit**

Recommendation: Create all status update tickets at the beginning of the sprint

Example Weekly Status Update

Expectations: Code

- All students should contribute code during the October Sprint
- Code should be pushed to feature branches and PRed to main
- We will only evaluate code pushed to main
- Link PRs to tickets if possible

Example Ticket w/ Linked PR

End of October: “Demo 0”

- Teams will meet with all instructors during **10/30** lab to go over general progress and review what was accomplished in October
- During instructor meetings the week of **11/3**, students will meet individually with their instructor to review individual code & sprint progress

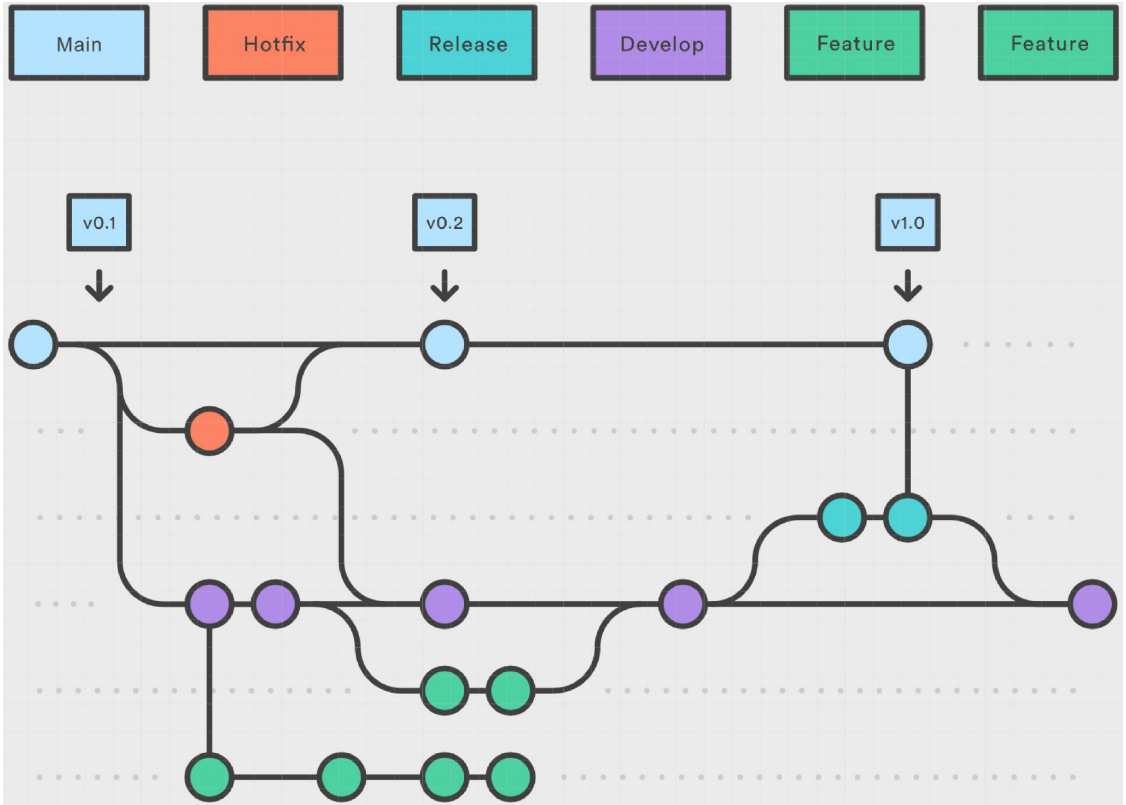
Use “Demo 0” as your milestone for the October sprint

Agenda

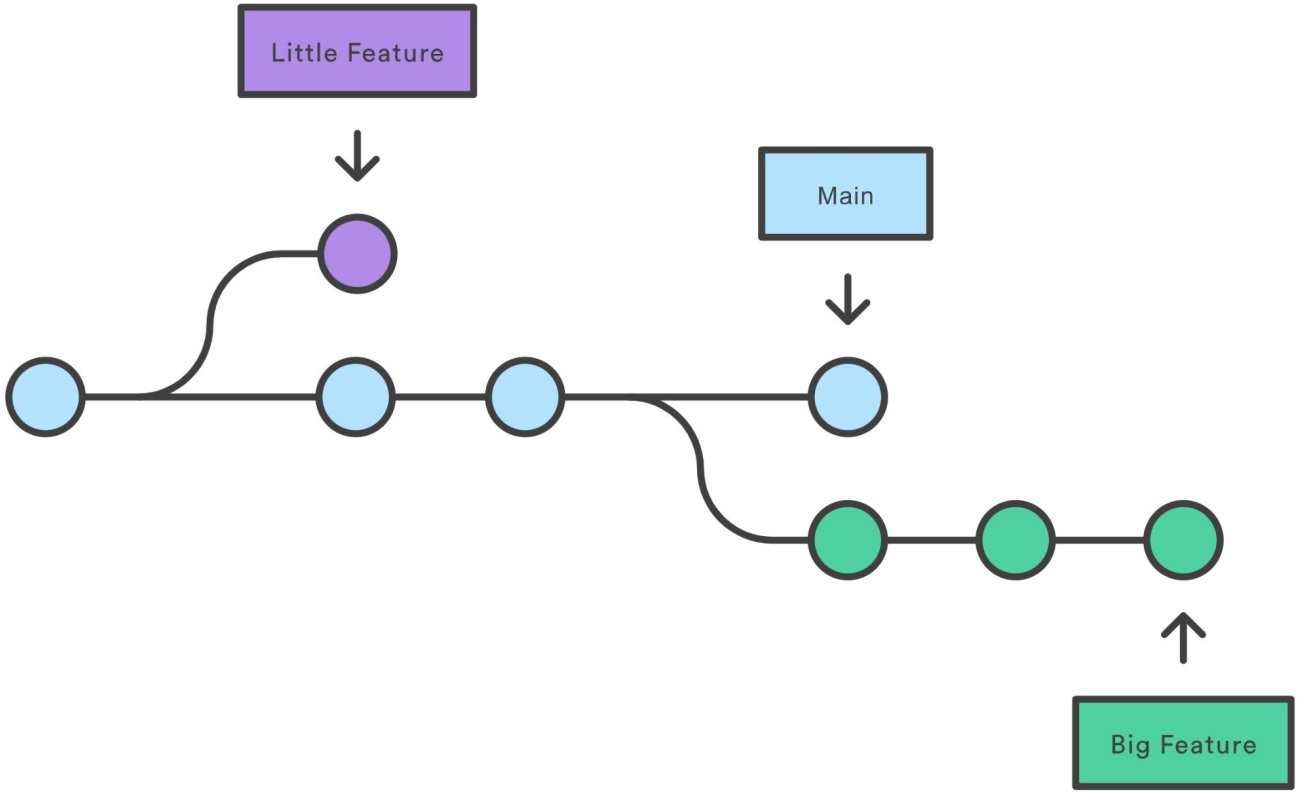
- September Sprint Feedback
- October Sprint Planning
- **Git / PR Reviews**
- CI/CD
- Presentation 1
- For next week
- Team Charter

Git

Git Workflow Diagram



Git Workflow Diagram for Senior Design



Developing a feature

```
git checkout main && git pull
git checkout -b js-my-feature
git push -u origin js-my-feature
```

(code changes)

```
git add .
git commit -m “made changes”
git push
```

```
git checkout main && git pull
git checkout js-my-feature
git merge main
```
(may need to resolve merge conflicts)

```
git push
```
(open PR)

Git Resources

- ChatGPT
- https://dangitgit.com/en
- https://www.atlassian.com/git/tutorials/using-branches
- https://code.visualstudio.com/docs/sourcecontrol/overview#_3way-merge-editor

PR Reviews

Purpose of Code Reviews

- Ensure that team members are aware of changes to the codebase
- Allow others to verify the correct things are being tested
- Facilitate discussions over implementation design

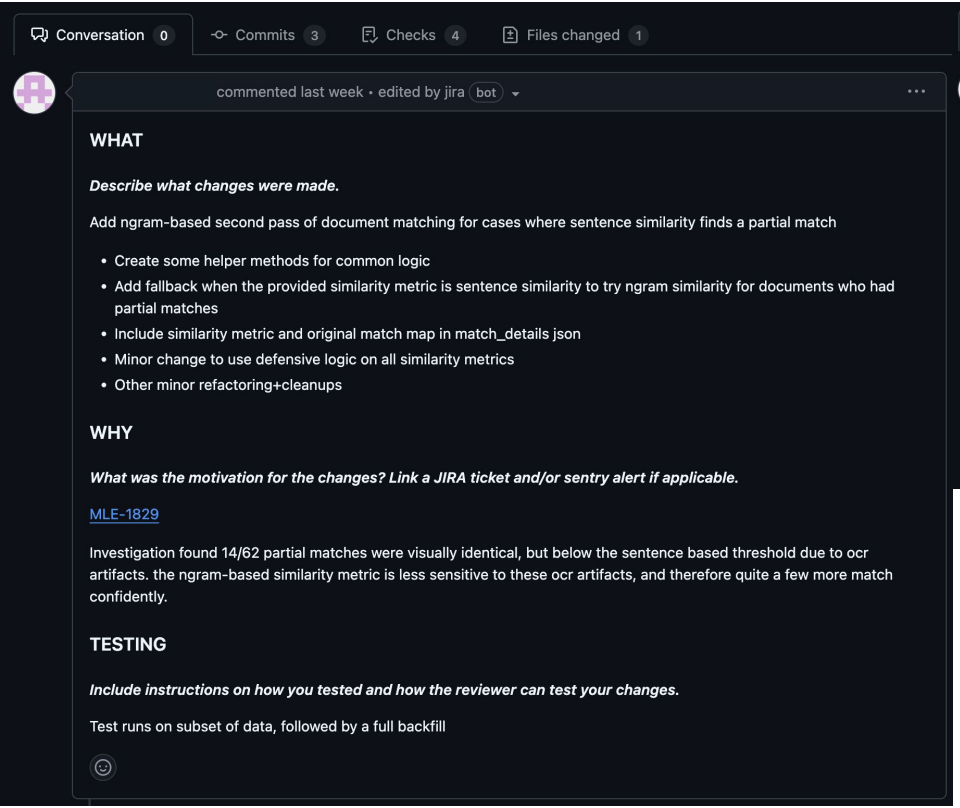
The overall code health should be improving over time, and developers should make progress on their tasks

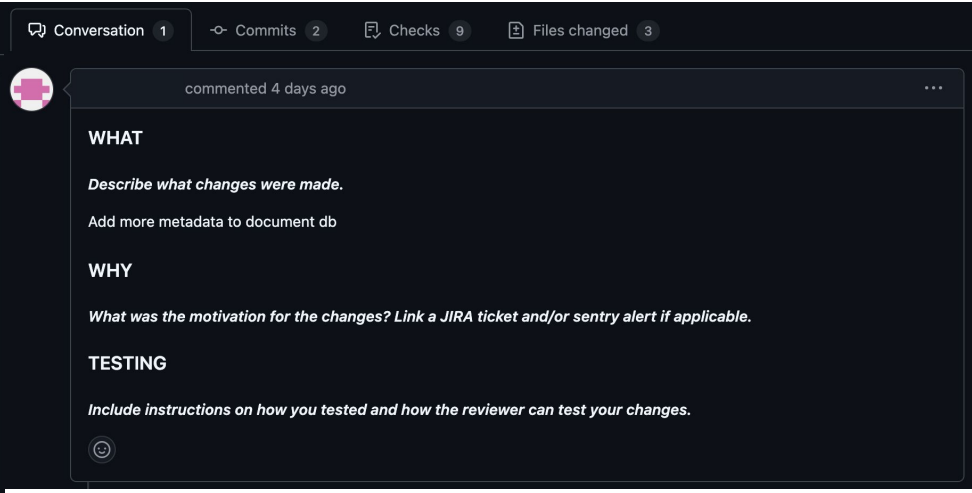
Reviewers should favor approving PRs once its in a state where it improves code health, even if the PR isn’t perfect

Authoring a Pull Request

- A single PR should represent a single piece of functionality
- Multiple PRs with small changes is better than one PR with lots of changes
- The description should include **what** changed and **why** the change is necessary
- Add pr comments to code changes to help reviewers navigate the diff
- Link PR to sprint task
- If the PR is large or complicated, meet with the reviewers to discuss

Example PRs


Conversation 0
Commits 3
Checks 4
Files changed 1
commented last week • edited by jira bot
WHAT
Describe what changes were made.
Add ngram-based second pass of document matching for cases where sentence similarity finds a partial match
Create some helper methods for common logic
Add fallback when the provided similarity metric is sentence similarity to try ngram similarity for documents who had partial matches
Include similarity metric and original match map in match_details json
Minor change to use defensive logic on all similarity metrics
Other minor refactoring+cleanups
WHY
What was the motivation for the changes? Link a JIRA ticket and/or sentry alert if applicable.
MLE-1829
Investigation found 14/62 partial matches were visually identical, but below the sentence based threshold due to ocr artifacts. the ngram-based similarity metric is less sensitive to these ocr artifacts, and therefore quite a few more match confidently.
TESTING
Include instructions on how you tested and how the reviewer can test your changes.
Test runs on subset of data, followed by a full backfill


Conversation 1
Commits 2
Checks 9
Files changed 3
commented 4 days ago
WHAT
Describe what changes were made.
Add more metadata to document db
WHY
What was the motivation for the changes? Link a JIRA ticket and/or sentry alert if applicable.
TESTING
Include instructions on how you tested and how the reviewer can test your changes.

Reviewing a Pull Request

Goal: Ensure the changes are positive, even if they aren't perfect

- **Mountain**: feedback that blocks all related work and requires immediate action
- **Boulder:** feedback that blocks the work from being approved, but doesn't require immediate action
- **Pebble:** feedback that does not block the PR, but requires future action
- **Sand**: feedback that is not blocking, but should be considered if multiple team members concur.
- **Dust/nit:** feedback that is more a suggestion and not required

https://www.netlify.com/blog/2020/03/05/feedback-ladders-how-we-encode-code-reviews-at-netlify/

Code Reviews for Senior Design

- Team members should not push directly to main
- Team members should try to review each other’s code
- While mentors should not be reviewing all code changes, ask them to do a PR review sometime this semester!
- PRs do not need to be blocked by approvals

Agenda

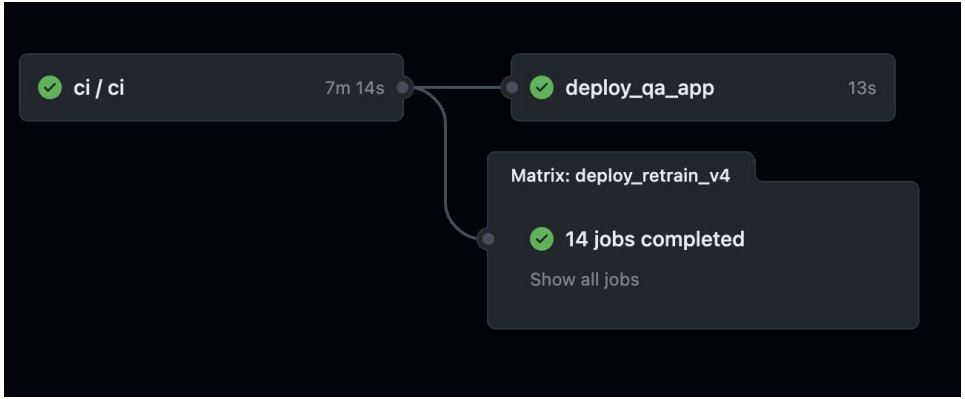
- September Sprint Feedback
- October Sprint Planning
- Git / PR Reviews
- **CI/CD**
- Presentation 1
- For next week
- Team Charter

Continuous Integration & Deployment

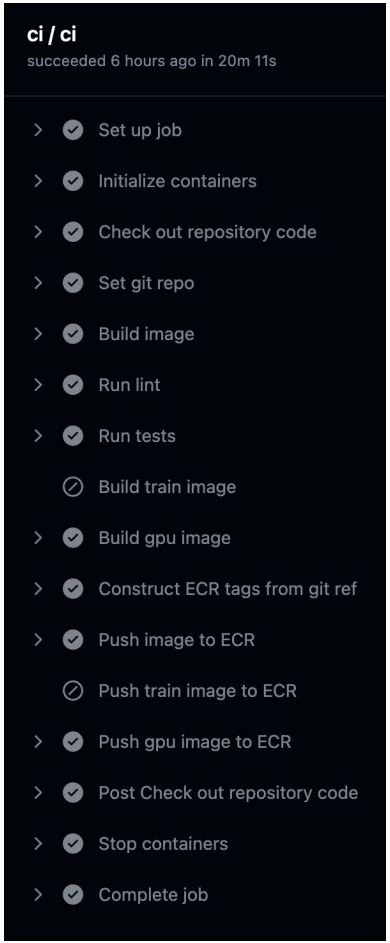
- Continuous Integration is a practice that involves frequently and automatically integrating code changes into a shared repository. The core idea is to detect and address integration issues early in the development process.
    - Unit tests, integration tests, linting. Blocks merging bad code. Frees up developers from manually testing

- Continuous Deployment is an extension to CI that automates the deployment process. It means every code change that passes CI tests is automatically deployed without manual intervention.
    - Builds artifacts, deploys to staging and/or prod environments

Example CI/CD Pipeline



- Run the CI step on every push
  - Gate merges on CI step
- Run the deploy step on every push to main
  - Gate deploy step on CI step



CI/CD Tools

- Circle CI, Travis, Jenkins, Argo, Codefresh, Spinnaker
- Github Actions
  - Free!
  - Easy to configure as part of your github repo

Example Github Action Pipeline

CI/CD for Senior Design

- This is not required, but highly recommended
- Use github actions for CI/CD execution
- Recommended CI steps (on every push):
  - Lint code
  - Run tests
  - Build artifacts
- Recommended CD steps (on merges to main or manual trigger):
  - Build artifacts
  - Deploy changes

Agenda

- September Sprint Feedback
- October Sprint Planning
- Git / PR Reviews
- CI/CD
- **Presentation 1**
- For next week
- Team Charter

Presentation 1: Elevator Pitch

- **Due Date: 10/16**
- **Goal**
  - Convince us that what you are building is a great idea, and that you have a way to make it a reality
  - Build off of writing 1 & project proposal
  - Audience: non technical (investors, upper management, etc)
- **Requirements**
  - 4 minutes long + 2 mins for questions
  - What are you building and why? Who are you users? What are the goals? How is it different from current products/research?
  - Be prepared to answer non-technical questions
  - Grade is primarily based on presentation skills!

Upload slides to shared google drive prior to presentation day

Agenda

- September Sprint Feedback
- October Sprint Planning
- Git / PR Reviews
- CI/CD
- Presentation 1
- **For next week**
- Team Charter

For Next Week

Weekly Focus

- Plan out your sprint – what do you want to accomplish by “demo 0”

Mentor Meetings

- [Team]: October sprint planning

Deadlines

- [Individual]: Writing 1 - Executive Summary (**Oct. 6**)
- [Individual]: September team progress form (**Oct. 6**)
- [Team]: Team Charter (**Oct. 6, Github**)
- [Team]: Presentation 1 (**Oct. 16)**

Reminders

- Don’t forget to post weekly updates (due EOD!)

Agenda

- September Sprint Feedback
- October Sprint Planning
- Git / PR Reviews
- CI/CD
- Presentation 1
- For next week
- **Team Charter**

Team Charter: What is it?

- **What**: A formal document that defines the team’s mission, scope of operation, objectives, and participants’ roles and responsibilities

- **Why**: Establishes clear expectations and guidelines for team collaboration

Importance of Team Charters

- Aligns team members on project goals and expectations
- Clarifies roles and responsibilities
- Establishes communication protocols
- Helps prevent and resolve conflicts
- Increases team accountability

Components of a Team Charter

1. Goals and Objectives
2. Roles and Responsibilities
3. Communication Guidelines
4. Decision Making Guidelines
5. Conflict Resolution Strategies
1. Performance Standards
2. Resource Allocation

Goals & Objectives

- Brief description of team’s project
- Specific short and long term objectives

These can be taken from the project proposal slides

Roles & Responsibilities

- Clear definition of each team member’s role
- Specific responsibilities assigned to each role
  - Project specific (frontend, backend, etc)
  - Logistics: who creates weekly status tickets, who takes notes in meetings, etc
- Skills and strengths of team members

Ex:

- Backend developer: responsible for database design & api development
- Team lead: responsible for creating weekly tickets, running weekly meetings, keeping team on track

Communication Guidelines

- Preferred communication channels (slack, in person, etc)
- Frequency and format of team meetings (as a team, w/ instructors, w/ mentors)
- Reporting and documentation standards (where do notes go?)

Ex: “weekly mentor meetings every Wednesday at 8pm via Zoom”

Decision Making Guidelines

- Agreed-upon method for making team decisions
- Voting procedures or consensus-building approaches
- Escalation process for unresolved decisions

Ex: “Major decisions require a majority vote. If no majority, we will reach out to team mentor for guidance.”

Performance Standards

- Expectations for deliverables
- Time management and deadline adherence
- Code review and testing procedures
- Team member removal

Ex: “All code must pass tests and be reviewed by at least one other team member prior to merging”

Resource Allocation

- Distribution of workload
- Time commitments expected from each member
- Shared resources and how to access them (hardware, compute, etc)

Ex: “Each team member commits to 10 hours per week on the project. Work is assigned based on each member’s expertise and availability.”

Team Charter for Senior Design

1. Download a copy of the team charter template
2. As a team, work together to fill in the template. Feel free to update as you see fit.
   a. Be as specific and thorough as possible
   b. This is a living document, edit as needed
3. Commit the charter to your github repo as **team_charter.md**
4. In a **separate commit**, **each member** should add their name to the signature section.
5. Use the rest of lab to complete this, if you don't finish it is due **10/6**.