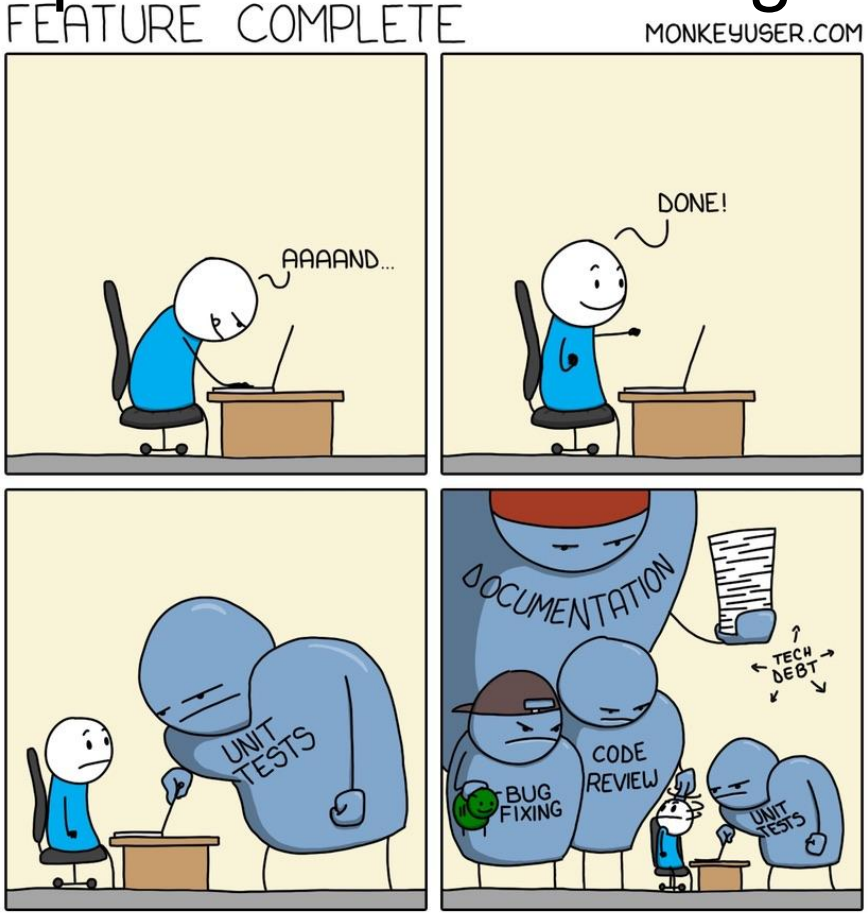
COSC 499: Capstone Software Engineering Project

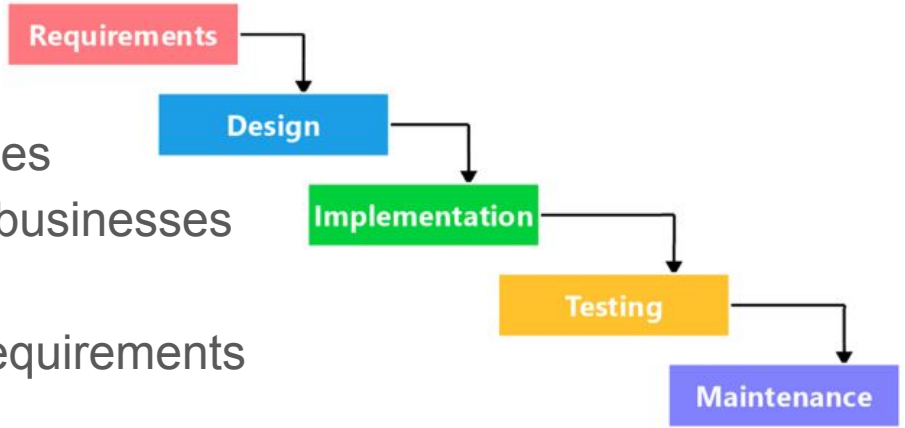


Traditional Waterfall Model

- Software development lifecycle (SDLC) describes the process of how a piece of software is developed by a group of engineers

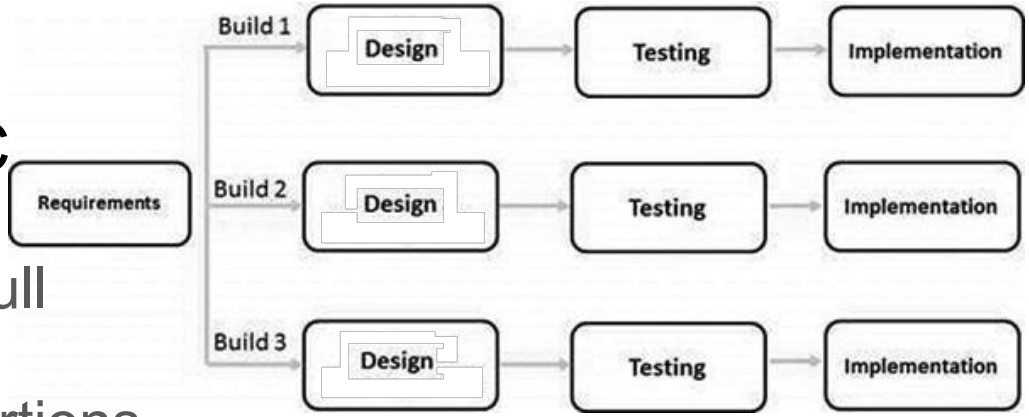
Traditional Waterfall Model

- Software development lifecycle (SDLC) describes the process of how a piece of software is developed by a group of engineers
- Waterfall model
  - Linear model with sequential phases
  - Easy to understand and adopt by businesses
  - Expensive to fix and maintain
  - Cannot accommodate changing requirements

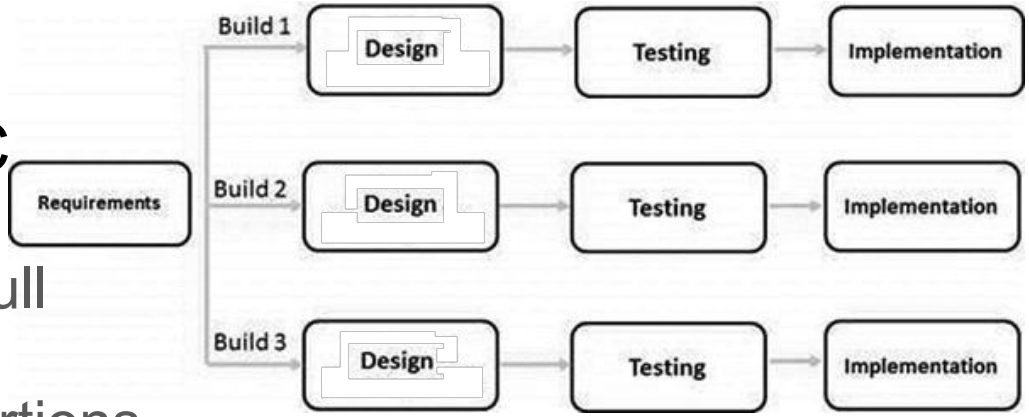


Iterative/Incremental SDLC

- Does not attempt to have full spec of requirements
- Incrementally add small portions

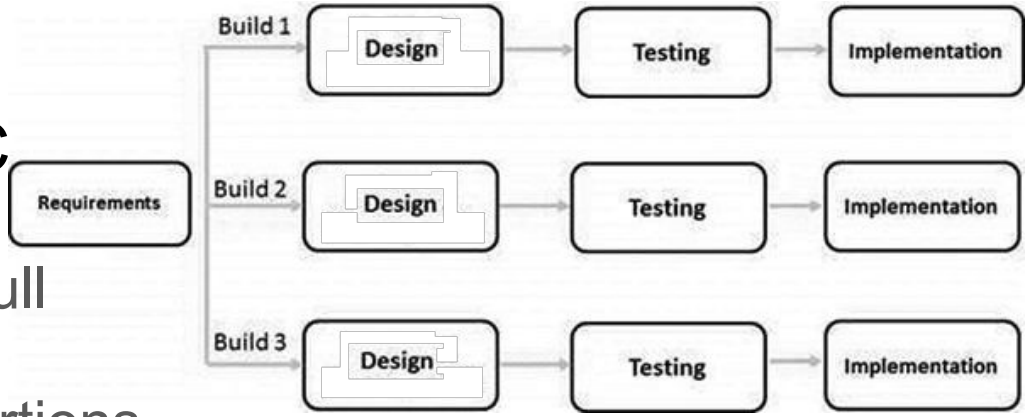


Iterative/Incremental SDLC



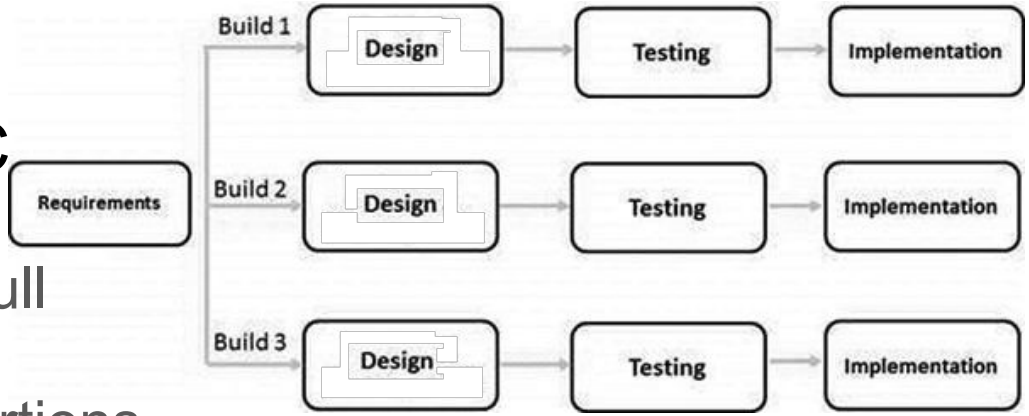
- Does not attempt to have full spec of requirements
- Incrementally add small portions
- At each iteration:
  - Modify design as needed
  - Add new features (implement small parts) to the system
  - Test, integration, and regression testing

Iterative/Incremental SDLC



- Does not attempt to have full spec of requirements
- Incrementally add small portions
- At each iteration:
  - Modify design as needed
  - Add new features (implement small parts) to the system
  - Test, integration, and regression testing
- Characteristics:
  - Allows for cycles
  - Observe working system (prototypes) early in the development process
  - Less costly for debugging, testing, and incorporating feedback

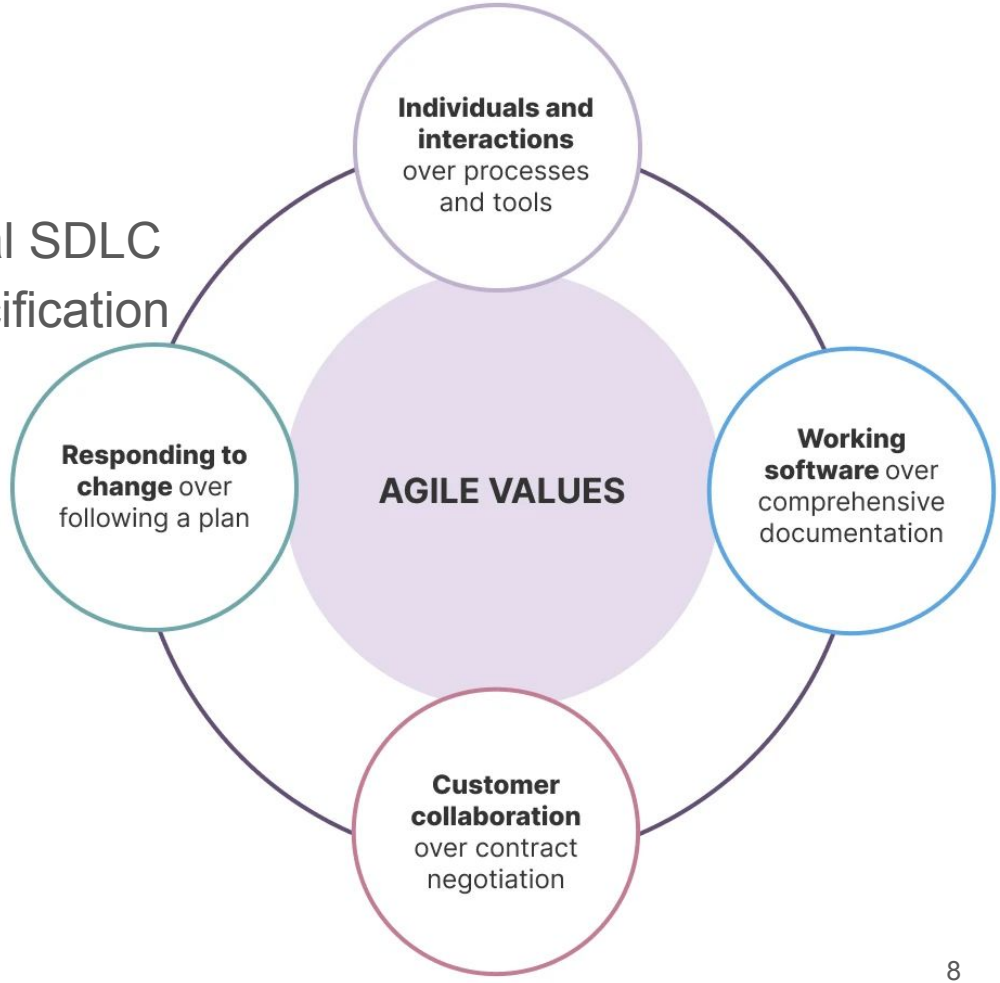
Iterative/Incremental SDLC



- Does not attempt to have full spec of requirements
- Incrementally add small portions
- At each iteration:
  - Modify design as needed
  - Add new features (implement small parts) to the system
  - Test, integration, and regression testing
- Characteristics:
  - Allows for cycles
  - Observe working system (prototypes) early in the development process
  - Less costly for debugging, testing, and incorporating feedback

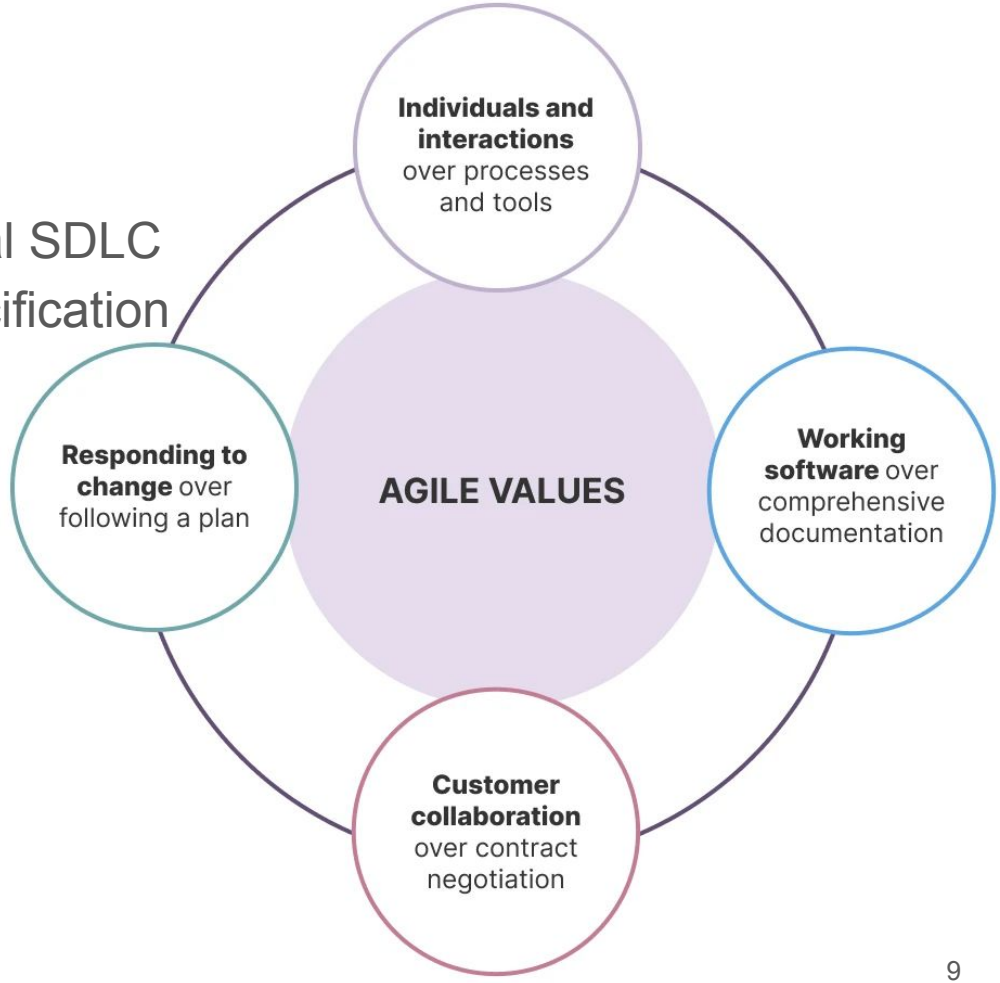
Agile SDLC

- Derived from Iterative/Incremental SDLC
- General plan rather than full specification



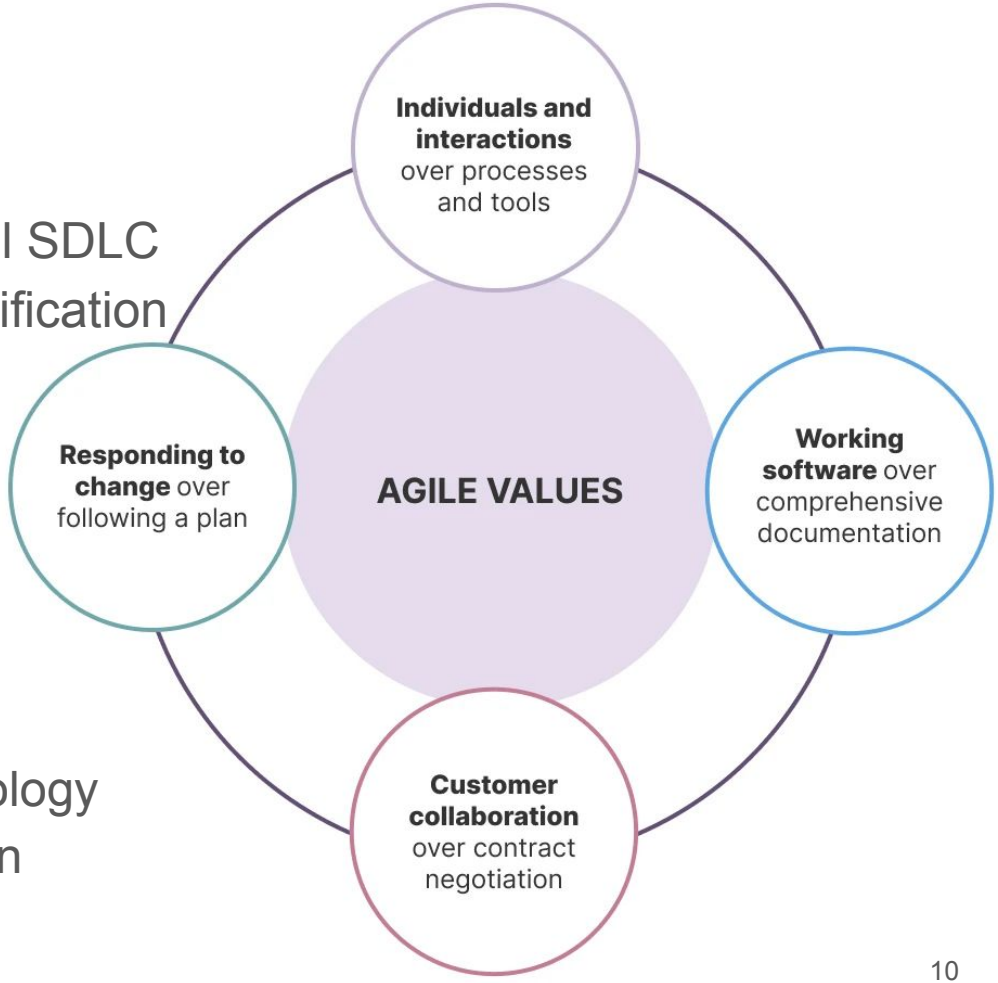
Agile SDLC

- Derived from Iterative/Incremental SDLC
- General plan rather than full specification
- Agile methodologies: Scrum, extreme programming (XP), etc.
- Agile Manifesto (2001): https://agilemanifesto.org/



Agile SDLC

- Derived from Iterative/Incremental SDLC
- General plan rather than full specification
- Agile methodologies: Scrum, extreme programming (XP), etc.
- Agile Manifesto (2001): https://agilemanifesto.org/
- **Challenges:**
  - Hard to budget and manage
  - Challenging to transfer technology due to a lack of documentation



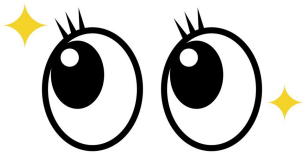
```python
class WeightRun(Run):
    def start(self, num_trials: int = 1, generate_graphs: bool = True):
        scenario = BowenScenario2()

        metrics = [
            AverageCosineDifference(
                name="Score Cosine Difference",
                attribute_filter=[Attributes.SCORE.value],
            ),
            AverageCosineDifference(
                name="Timeslot Cosine Difference",
                attribute_filter=[ScenarioAttribute.TIMESLOT_AVAILABILITY.value],
            ),
        ]

        student_provider = BowensDataProvider2()

        artifact: SimulationSetArtifact = SimulationSet(
            settings=SimulationSettings(
                num_teams=ceil(student_provider.num_students / 4),
                scenario=scenario,
                student_provider=student_provider,
                cache_key=f"weight_run_for_bowen/weight_run_2/",
            ),
            algorithm_set={
                AlgorithmType.PRIORITY: [
                    PriorityAlgorithmConfig(
                        MAX_KEEP=15,
                        MAX_SPREAD=30,
                        MAX_ITERATE=30,
                        MAX_TIME=100000,
                    ),
                ]
            },
        ).run(num_runs=num_trials)

        team_set = list(artifact.values())[0][0][0]

        insight_output_set = Insight.get_output_set(artifact, metrics)
        print(insight_output_set)

        data = [["ResponseId", "Q8", "Q4", "Q5", "zPos", "TeamId", "TeamSizeViolation"]]

        for team in team_set.teams:
            for student in team.students:
                attributes = student.attributes

                responseId = student_provider.get_student(student.id)

                timeslot = attributes[ScenarioAttribute.TIMESLOT_AVAILABILITY.value][0]
```

Code snippet in medium size repo

Context: New employee asked to run an algorithm with specific parameters

What do you foresee as potential challenges?

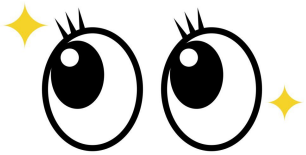
Writing Constructive Code Reviews



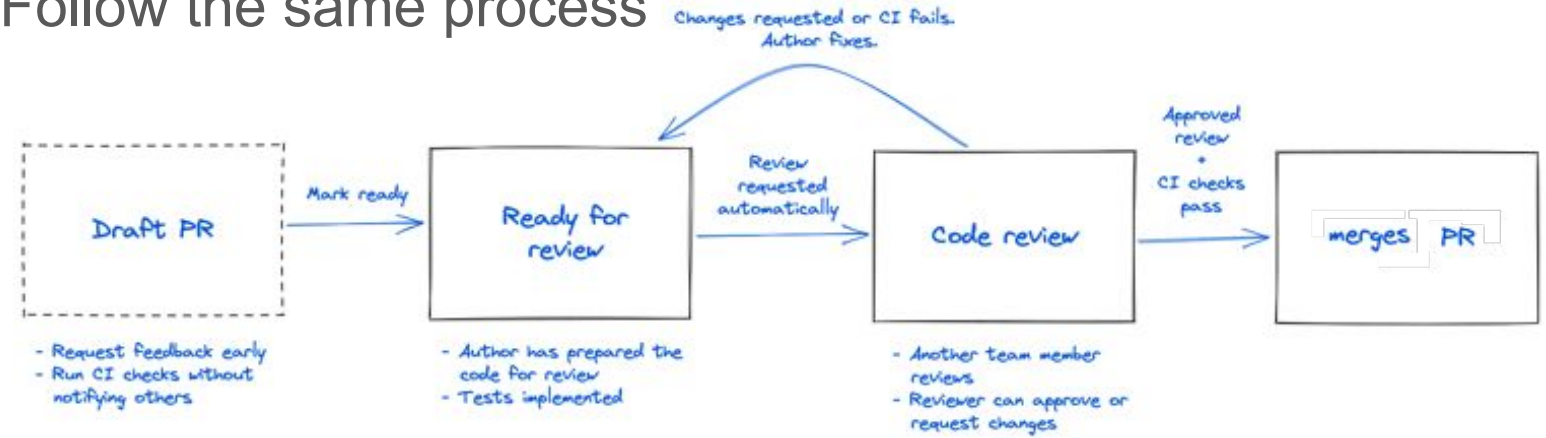
- Purpose:
  - Share knowledge
  - Spread ownership
  - Unify development practices
  - Quality control

Remember: You will be tested on *any* part of the repo, even if you didn't develop it

Writing Constructive Code Reviews



- Purpose:
  - Share knowledge
  - Spread ownership
  - Unify development practices
  - Quality control
- Follow the same process



What to Comment On?

- **Functionality** - behavior as intended?
- **Tests** - are they complete? do they pass?
- **Complexity and design** - easy for others to understand? follow standard patterns?

What to Comment On?

- **Functionality** - behavior as intended?
- **Tests** - are they complete? do they pass?
- **Complexity and design** - easy for others to understand? follow standard patterns?
- **Naming** - are they descriptive and follow pre-established conventions?
- **Comments** - are they clear and helpful?
- **Documentation** - are associated docs updated?

Things to Keep in Mind

- Keep PRs small
  - Earlier feedback
    Easier for others to review
  - Faster turnaround time

Things to Keep in Mind

- Keep PRs small
  - Earlier feedback
    Easier for others to review
  - Faster turnaround time
- Delegate nit-picking to a computer
  - e.g. linter

Things to Keep in Mind

- Keep PRs small
  - Earlier feedback
    Easier for others to review
  - Faster turnaround time
- Delegate nit-picking to a computer
  - e.g. linter
- Clear *mental model* of code:
  - Write meaningful feature requirements, PR descriptions, good naming conventions, commit messages, chat histories, descriptions for issue tracking, comment your code as needed

Things to Keep in Mind

- Keep PRs small
  - Earlier feedback
    Easier for others to review
  - Faster turnaround time
- Delegate nit-picking to a computer
  - e.g. linter
- Clear *mental model* of code:
  - Write meaningful feature requirements, PR descriptions, good naming conventions, commit messages, chat histories, descriptions for issue tracking, comment your code as needed
- PR authors have feelings too