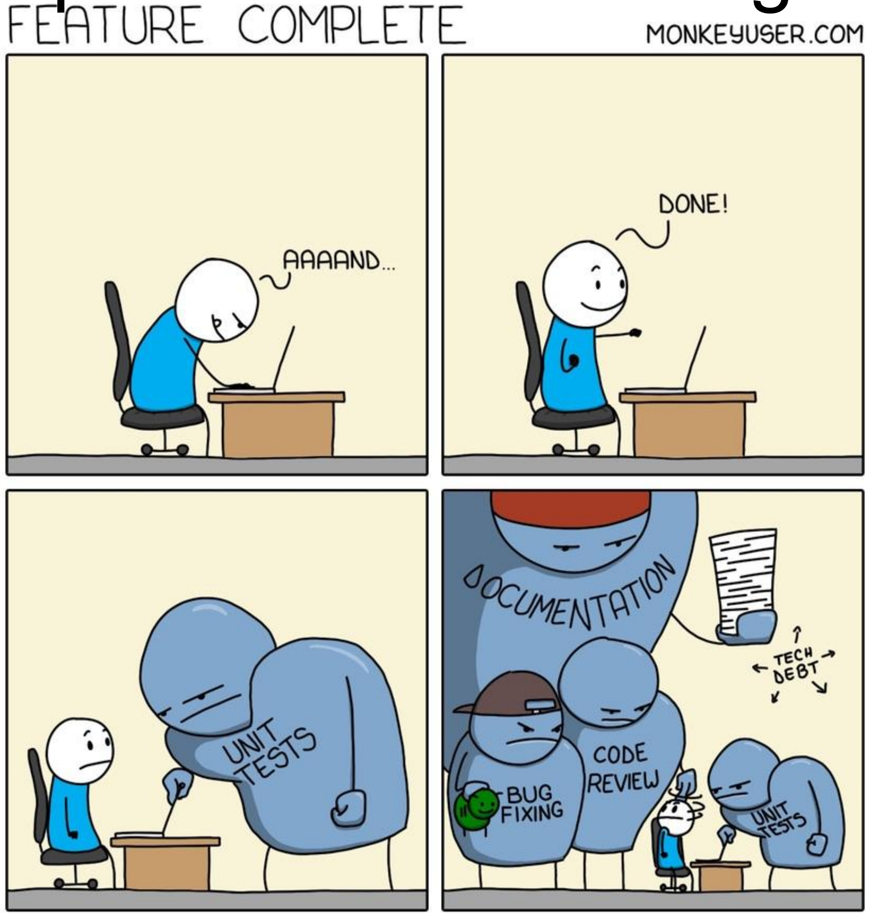
COSC 499: Capstone Software Engineering Project

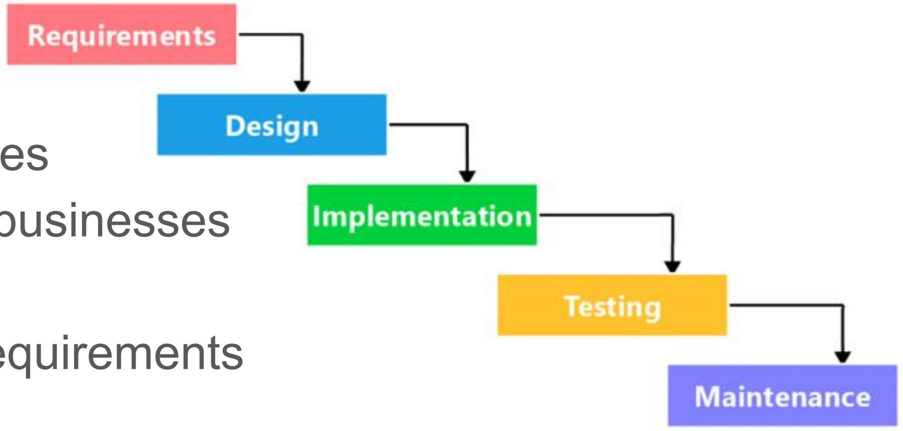


Traditional Waterfall Model

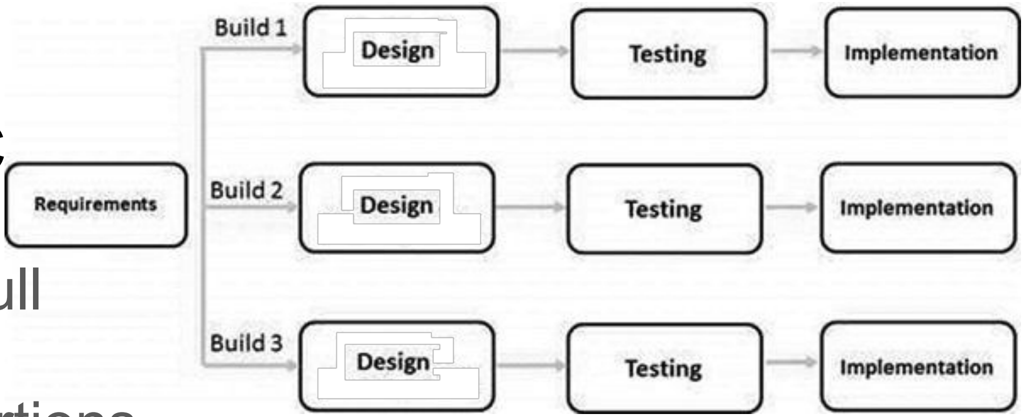
- Software development lifecycle (SDLC) describes the process of how a piece of software is developed by a group of engineers

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

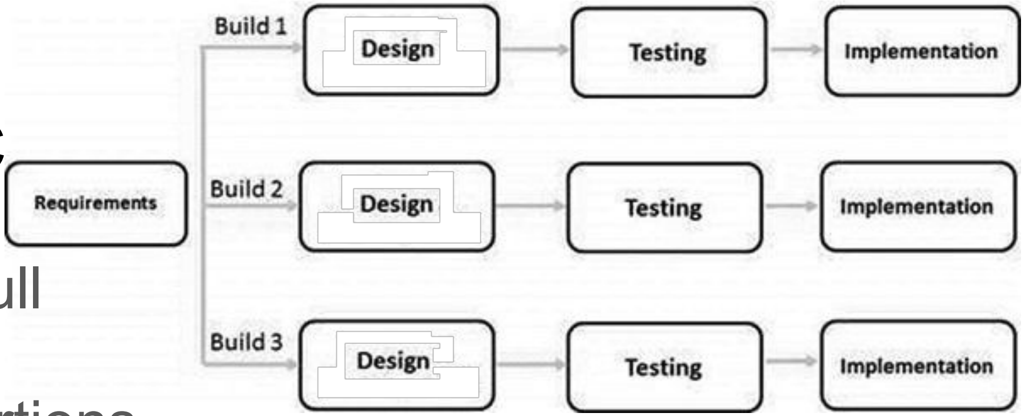


Iterative/Incremental SDLC



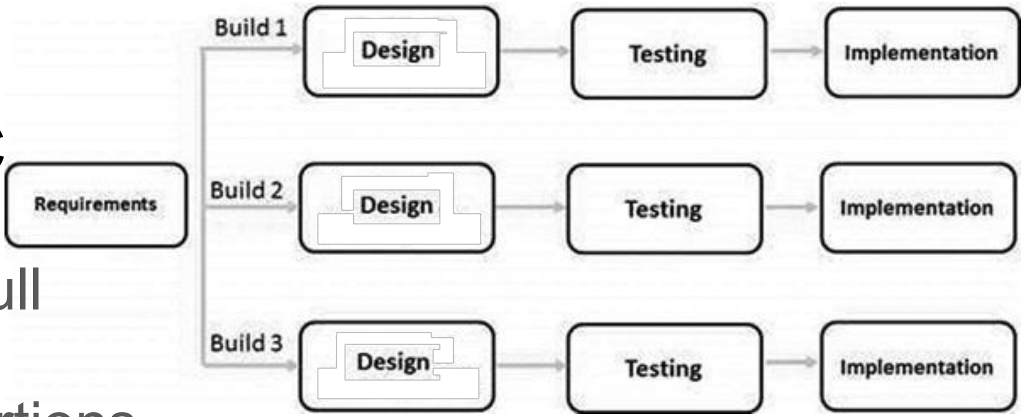
- Does not attempt to have full spec of requirements
- Incrementally add small portions

Iterative/Incremental SDLC



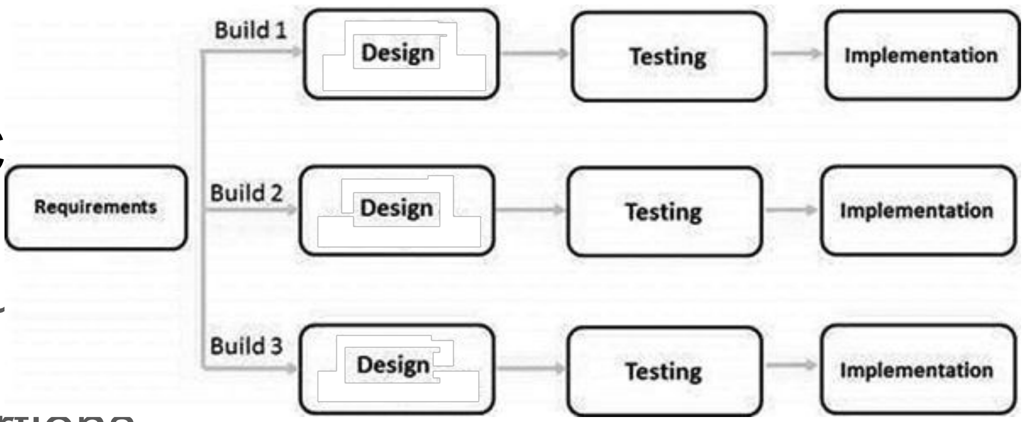
- Does not attempt to have full spec of requirements
- Incrementally add small portions
- At each iteration:
  - Modify design as needed
  - Add new features (implement small parts) to the system
  - Test, integration, and regression testing

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

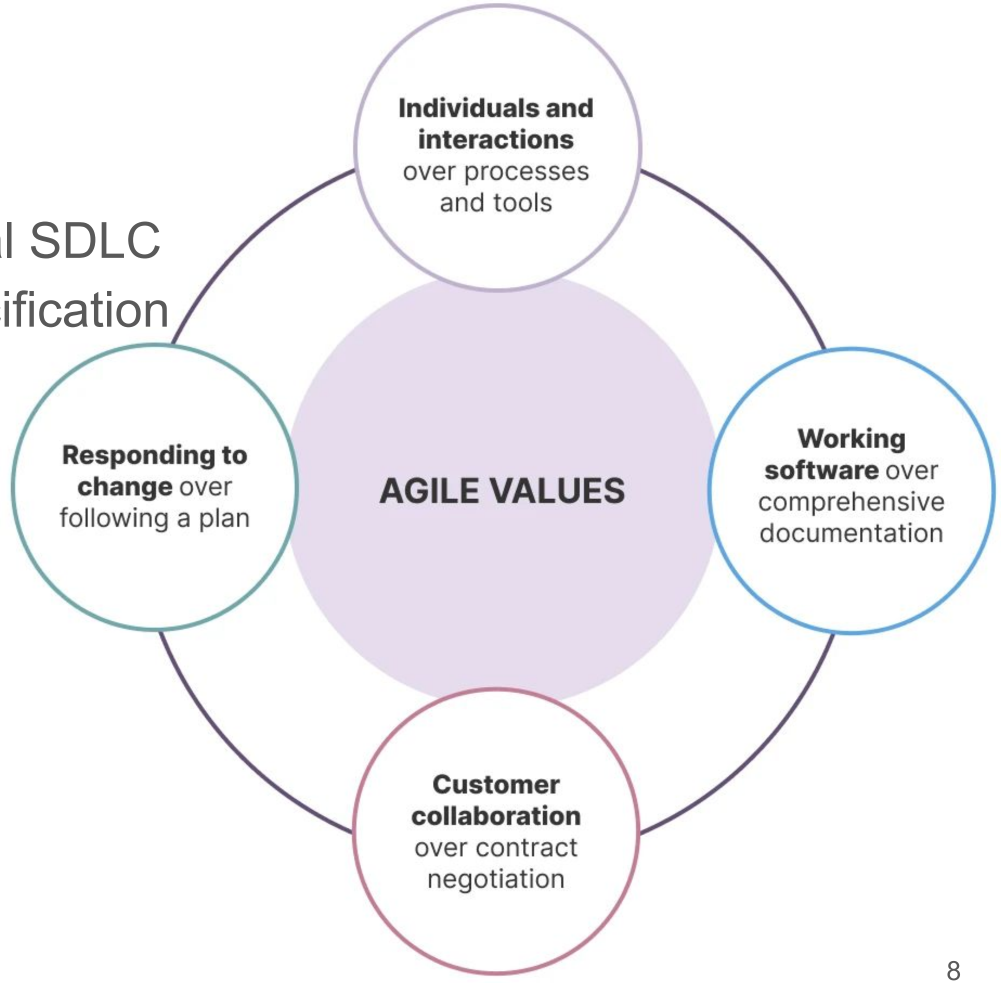


- Does not attempt to have fu
  spec of requirements
- Incrementally add small portions
- At each iteration:
  - Modify design as needed
  - Add new features (implement small parts) to the system
  - Test, integration, and regression testing
- Characteristics:
  - Allows for cycles
  - Observe working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

Note: test *before code*

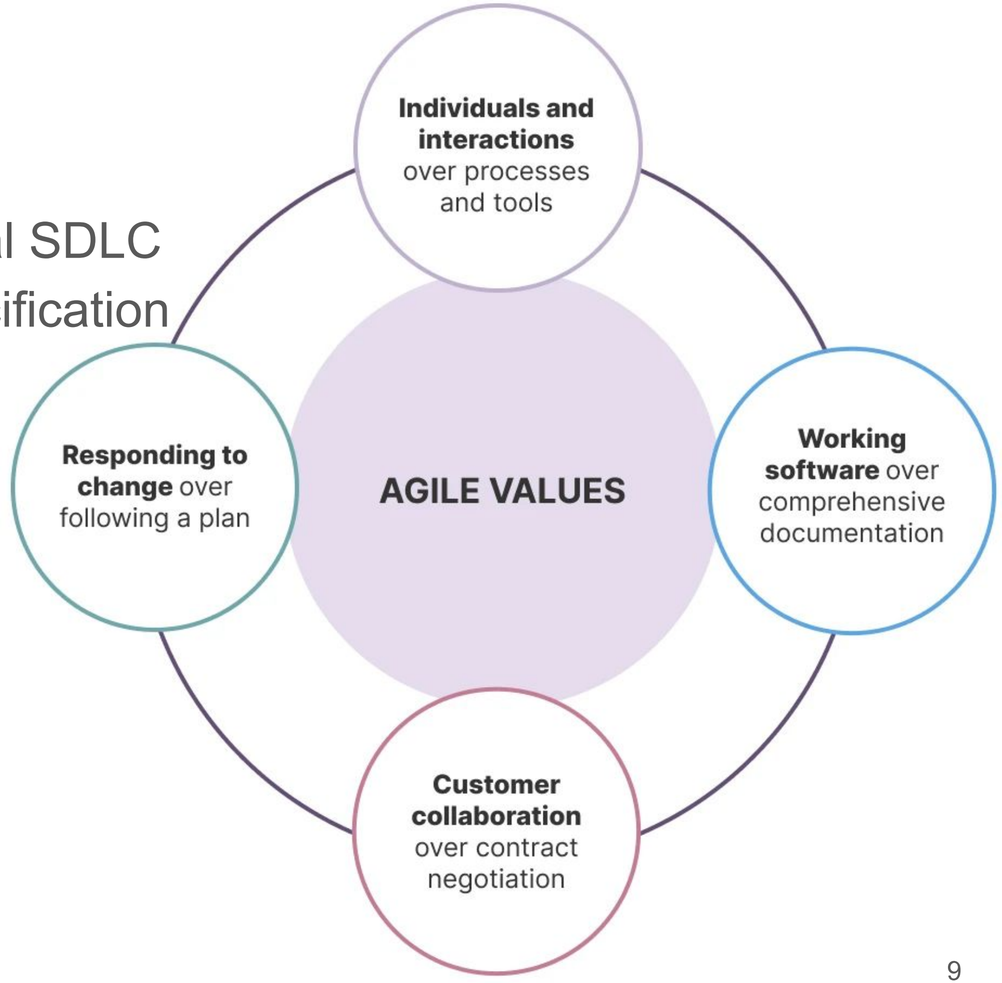
Agile SDLC

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


Individuals and interactions over processes and tools
Working software over comprehensive documentation
AGILE VALUES
Responding to change over following a plan
Customer collaboration over contract negotiation

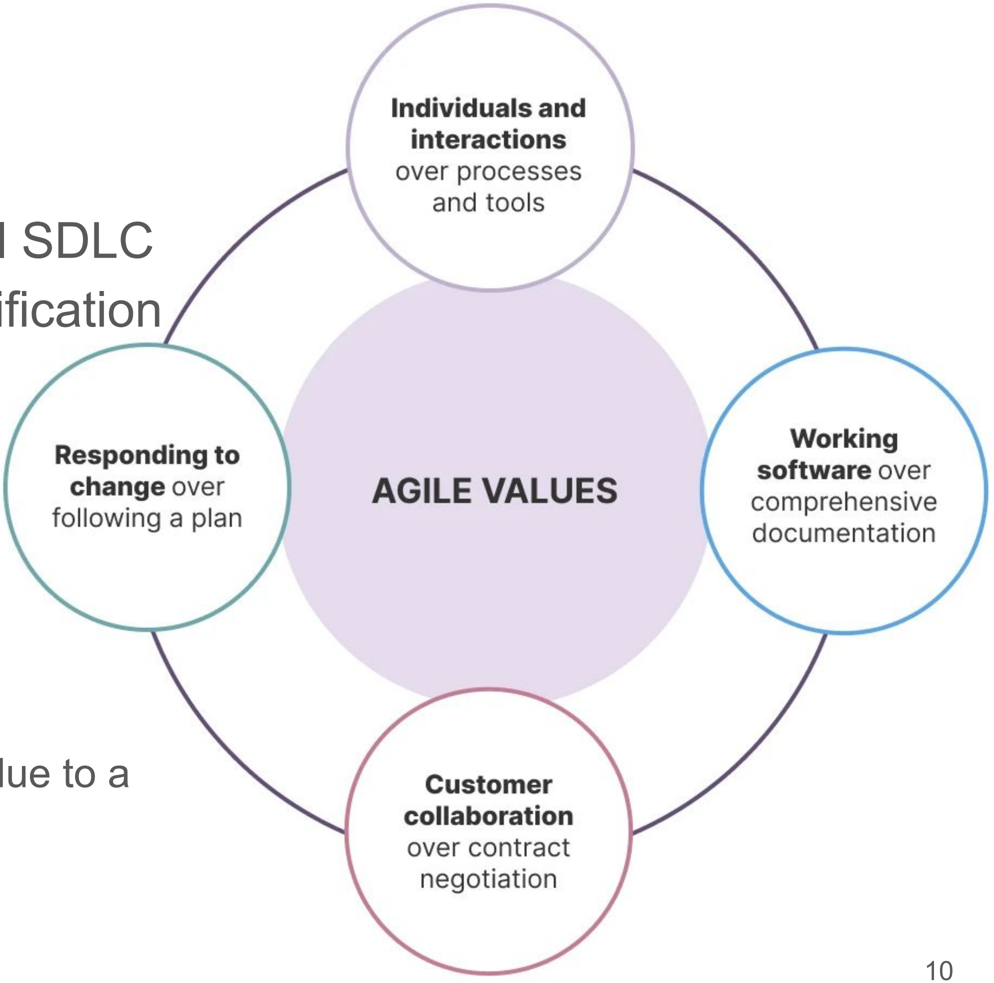
Agile SDLC

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

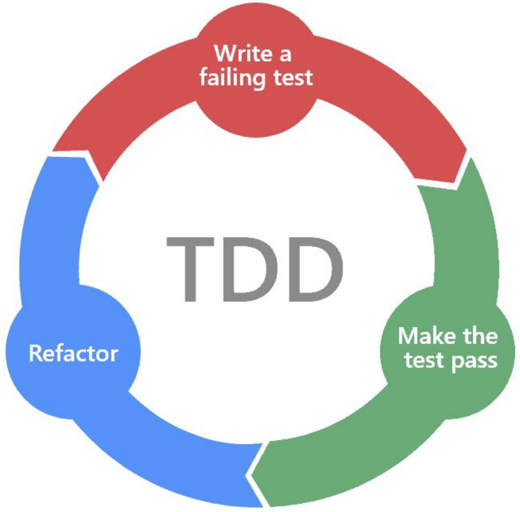

Individuals and interactions over processes and tools
Responding to change over following a plan
AGILE VALUES
Working software over comprehensive documentation
Customer collaboration over contract negotiation

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/
- Characteristics:
  - Hard to budget and manage
  - Challenging to transfer technology due to a lack of documentation


Individuals and interactions over processes and tools
Responding to change over following a plan
AGILE VALUES
Working software over comprehensive documentation
Customer collaboration over contract negotiation

Test Driven Development (TDD)



- Given a feature:

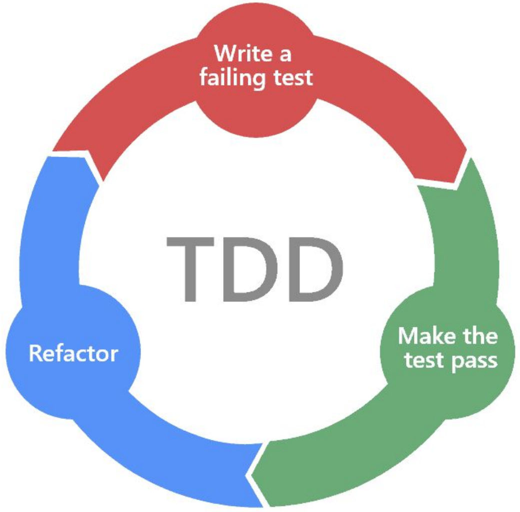
$\Big\}$ *test before code*

- Easy to do for unit testing or component testing

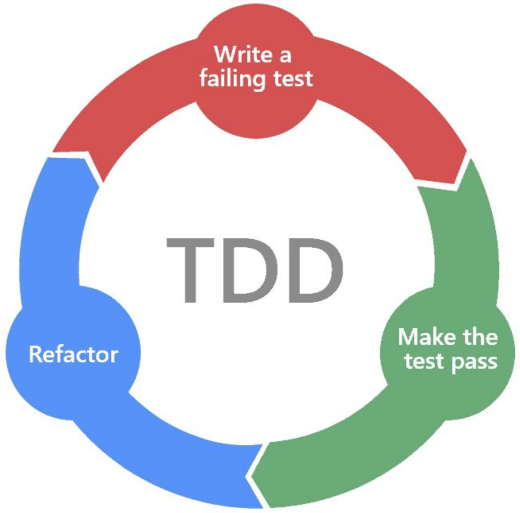
Test Driven Development (TDD)

- Given a feature:
    - What does it need to do to meet the requirement?
    - Translate this to:
      Which tests does this feature need to pass?

- Easy to do for unit testing or component testing



Test Driven Development (TDD)



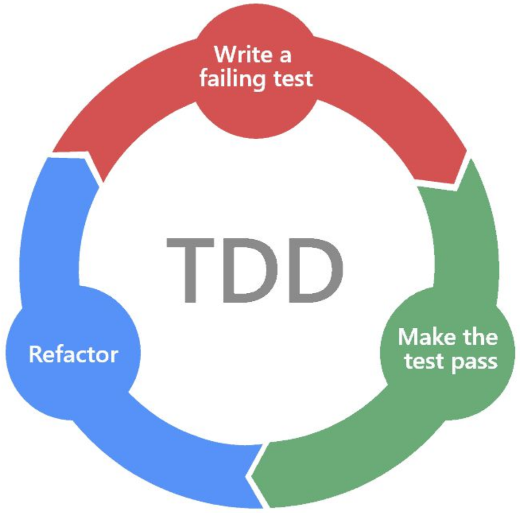
- Given a feature:
  - What does it need to do to meet the requirement?
  - Translate this to:
    Which tests does this feature need to pass?
  - Write the tests:
    - Include positive and negative cases
    - Include boundary cases

- Easy to do for unit testing or component testing

Test Driven Development (TDD)

- Given a feature:
  - What does it need to do to meet the requirement?
  - Translate this to:
    Which tests does this feature need to pass?
  - Write the tests:
    - Include positive and negative cases
    - Include boundary cases
  - Let the tests fail

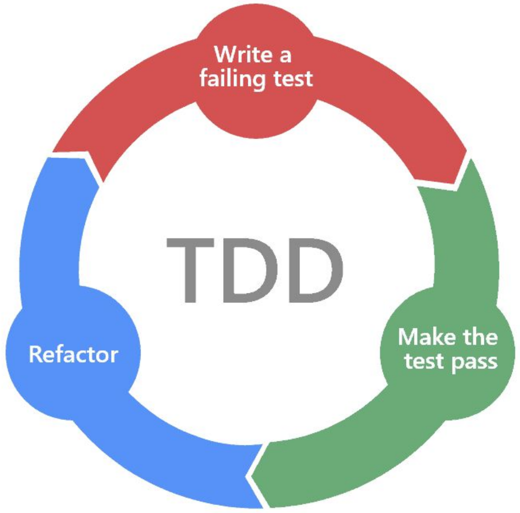
- Easy to do for unit testing or component testing



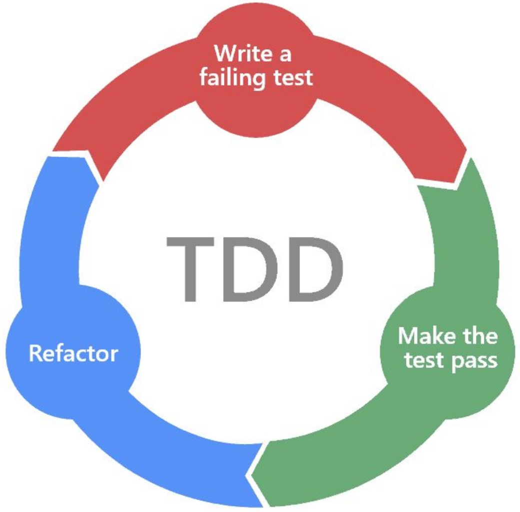
Test Driven Development (TDD)

- Given a feature:
  - What does it need to do to meet the requirement?
  - Translate this to:
    Which tests does this feature need to pass?
  - Write the tests:
    - Include positive and negative cases
    - Include boundary cases
  - Let the tests fail
  - Write the code to pass each test

- Easy to do for unit testing or component testing



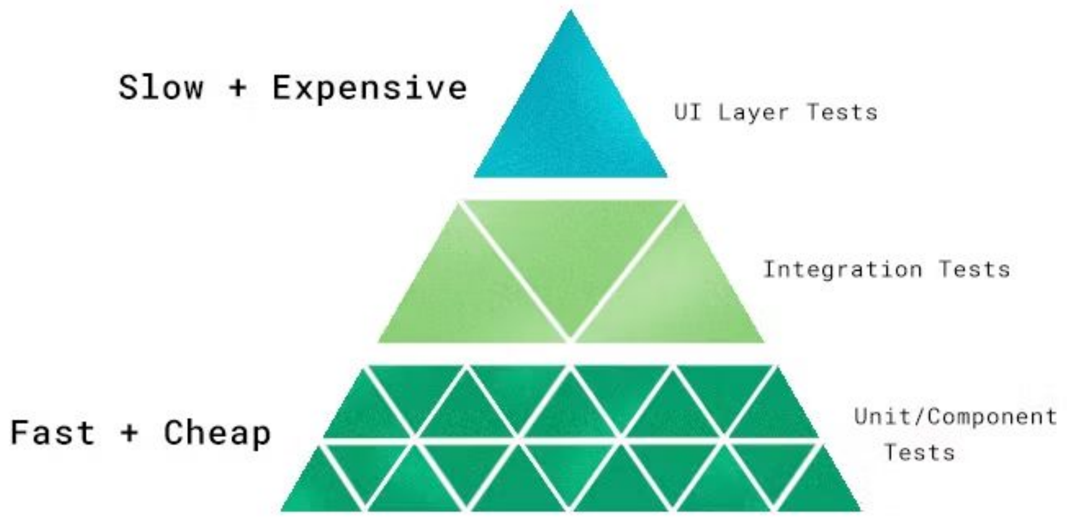
Test Driven Development (TDD)



- Given a feature:
  - What does it need to do to meet the requirement?
  - Translate this to:
    Which tests does this feature need to pass?
  - Write the tests:
    - Include positive and negative cases
    - Include boundary cases
  - Let the tests fail
  - Write the code to pass each test
  - Refactor the code (clean up potential redundancies)
- Easy to do for unit testing or component testing

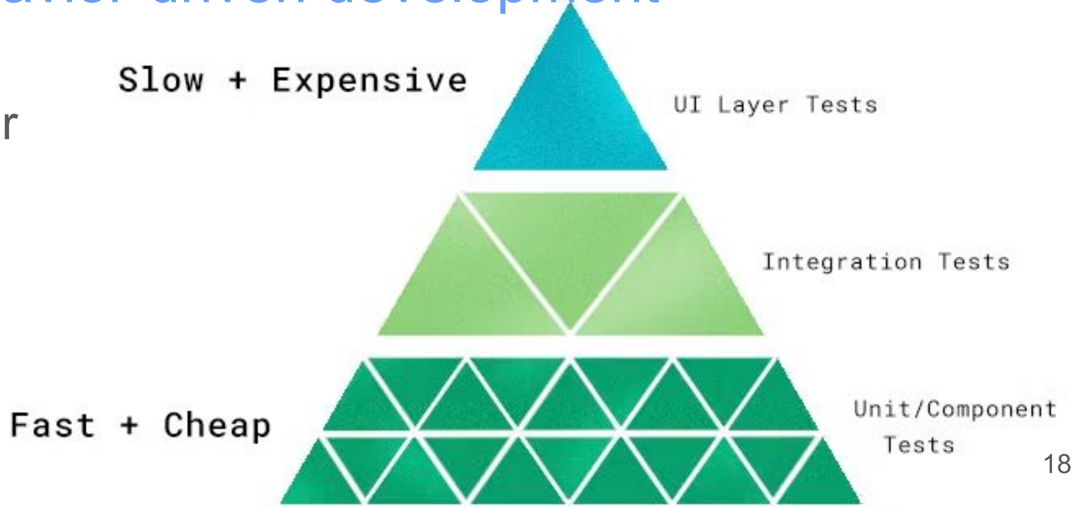
Types of Testing

- Integration and unit/component testing can be mocked and stubbed
- UI testing is often manual

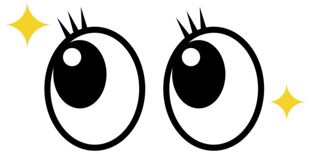


Types of Testing

- Integration and unit/component testing can be mocked and stubbed
- UI testing is often manual
- Alternative is to use behavior-driven development
    - Automated testing while anticipating user behavior
    - Check: Selenium



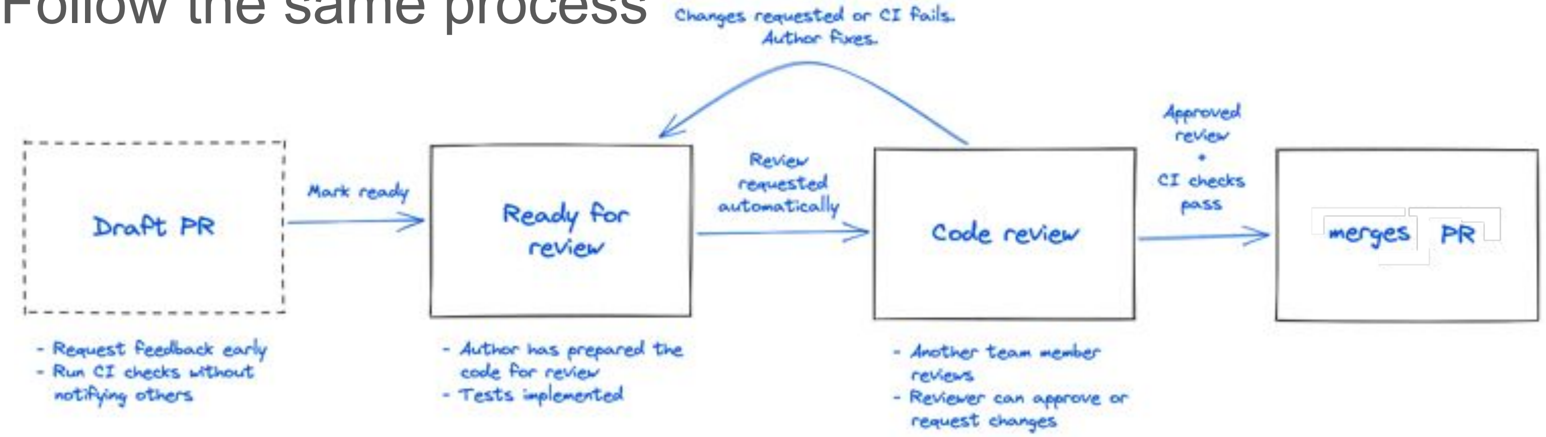
Writing Constructive Code Reviews



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

Writing Constructive Code Reviews

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



Changes requested or CI fails. Author fixes.
Draft PR
Mark ready
Ready for review
Review requested automatically
Code review
Approved review + CI checks pass
merges PR
- Request feedback early
- Run CI checks without notifying others
- Author has prepared the code for review
- Tests implemented
- Another team member reviews
- Reviewer can approve or request changes

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, commit messages, chat histories, descriptions for issue tracking

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, commit messages, chat histories, descriptions for issue tracking
- PR authors have feelings too

Next Steps

- Submit project plan with Google doc link on Canvas by Friday 11:59pm

- Next week: Open Topic
    - Focus on project checkin