Delivering the Backlog Using CI

Git Workflow

- Team works on origin
- Dev works on local
- Dev pushes to origin from local
- Merges happen on origin
    - Acceptance tests can be requested on merge on stage
- Origin eventually pushed to prod
- Start working on dev, push to stage, sent to prod
    - Delivered happens on stage
    - Accepted happens by customer prior to prod
- Team may work on a separate fork which is then merged into prod
- Work in priority order
    - Defines merge conflict resolution decisions
- Can specify a `PULL_REQUEST_TEMPLATE.md` file in `.github/PULL_REQUEST_TEMPLATE` folder

Bug Fixing

1. Report
2. Reproduce and or reclassify
    - Report may not necessarily be for a bug
3. Regression test
4. Repair
5. Release the fix

Reporting Bugs

- Add user / support message to tracker
    - Create a ticket
- Can use a fully featured bug tracker
- Use the simplest tool that works for your team and project scope

When to Reclassify

- Reclassify as “not a bug” or “won’t be fixed”
- Reproduce with the simplest possible test and add it to regression
    - Minimize preconditions (before blocks, given, or background steps)
- Repair $\triangleq$ test fails in presence of bug; make it pass

Patterns AntiPatterns and SOLID

Design Patterns

- A pattern language is an organized way of tackling an architectural problem using patterns
- Architectural (‘macroscale’) patterns (eg: MVC)
- Computation patterns (eg: FFT)
- Gang of Four (GoF) patterns $\triangleq$ structural, creational, behavior
    - Descriptive guide for how to structure OOP code
- Pattern is not an implementation
- Meta-Patterns $\triangleq$ separate out the things that change from those that stay the same
    - Both implementation and responsibilities
    - Program to an interface; not an implementation
    - Prefer composition and delegation over inheritance

        - Delegation is about interface sharing, inheritance is about implementation sharing
- Antipattern ≙ code that looks like it should probably follow some design pattern, but it doesn't
    - Often result of accumulated technical debt
    - Viscosity (easier to do hack than right thing)
    - Immobility (can't dry out functionality)
    - Needless repetition (comes from immobility)
    - Needless complexity from generality

SOLID OOP Principles

- Motivation: minimize cost of change
- Single Responsibility principle
- Open/Closed principle
- Liskov Substitution Principle
- Injection of Dependencies
    - Traditionally known as Interface Segregation principle
- Demeter principle

Single Responsibility Principle (SRP)

- Class should have one and only one reason to change
- Class's responsibility should be $\leq$ 25 words
- Models with many sets of behavior
    - Really big class files are a tip-off
- Lack of Cohesion Methods (LCOM)
    - Revised Henderson-Sellers := LCOM = 1 - $\sum_{MVi} MV$
        - MVi := # instance methods that access the ith instance variable (excluding 'trivial' getters and setters)
        - M := # instance methods
        - V := # instance variables
        - $0 \leq \text{LCOM} \leq 1$
    - LCOM-4 counts number of connected components in graph where related methods are connected by an edge
    - High LCOM suggests possible SRP violation
- Can avoid by looking for test seams and 'mock train-wrecks' as places to extract functionality to small classes

Universal Modeling Language (UML)

- UML := notation for describing various artifacts in OOP systems
- One type of UML diagram is a class diagram which shows class relationships and principal methods
- Care about relationship between different classes
- Open circle ≙ aggregation
- Diamond ≙ composition (relationship but not a dependency)
- Arrow ≙ inheritance
    - From the inherited class to the class being inherited
- Labels ≙ has one, has many, etc.
- Class-Responsibility-Collaborator (CRC) Cards
    - Class name (usually the nouns in a user story)
    - What the class is responsible for (usually the verbs in a user story)
        - Eg: knowing the name of a movie, computing ticket availability
    - What other classes share some part of the responsibility