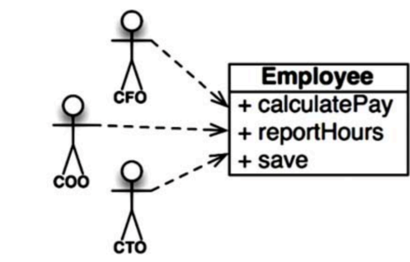
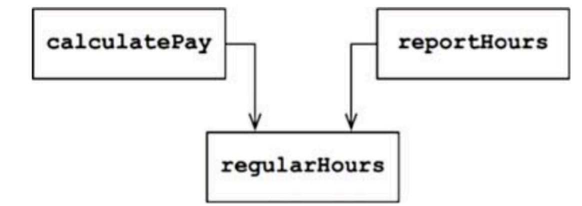
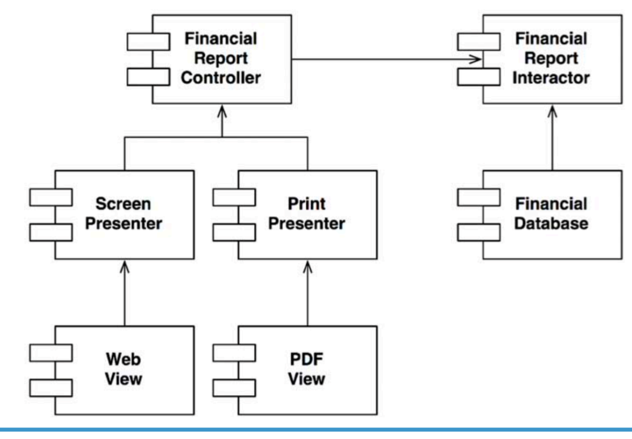
Part III - Design Principles (SOLID)

- **Intro**
  - Principles for mid-level software
  - Goal of principles
    - the creation of structures that
      - Tolerate change
      - Are easy to understand
      - Are the basis of components that can be used in many software systems
- **7. SRP: The Single Responsibility Principle**
  - Principle
    - A module should have one, and only one, reason to change
      - A module should be responsible to one, and only one, actor.
  - Violations
    - Symptom 1: Accidental Duplication

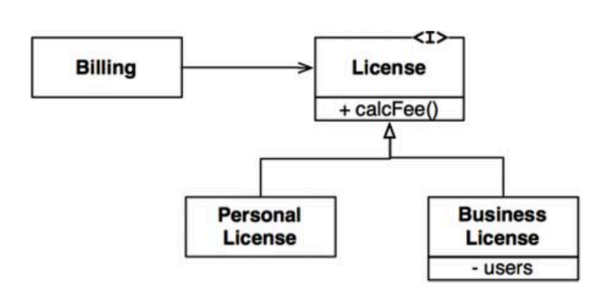
      

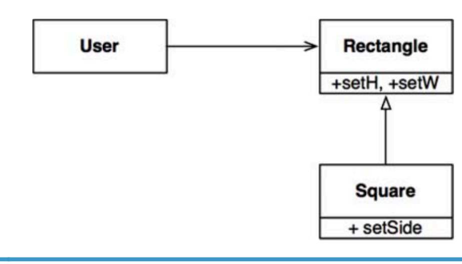
      - ⚠️ Separate the code that different actors depend on!
    - Symptom 2: Merges
      - Two different teams, check out the Employee class and begin to make changes
        - their changes collide & merge!
      - ⚠️ Separate code that supports different actors!
  - Solutions
    - The three classes do not know about each other
      - +The Facade pattern
    - Employee is used as a Facade for the lesser functions
- **8. OCP: The Open-Closed Principle**
  - Principle (by Bertrand Meyer in 1988)
    - A software artifact should be
      - open for extension
      - but closed for modification
  - Let easily to add new Views to display existing reports in different format

- **9. LSP: The Liskov Substitution Principle**
  - Principle (by Barbara Liskov In 1988)
    - What is wanted here is something like the following substitution property: If for each object o1 of type S there is an object o2 of type T such that for all programs P defined in terms of T, the behavior of P is unchanged when o1 is substituted for o2 then S is a subtype of T.
  - Guiding the Use of Inheritance

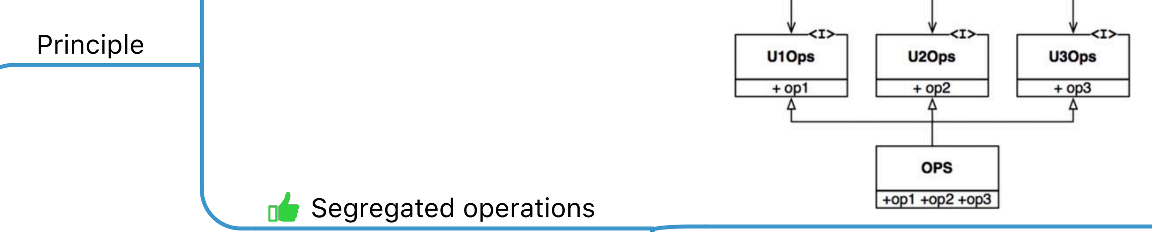
  - Violation
    - The Square/Rectangle Problem

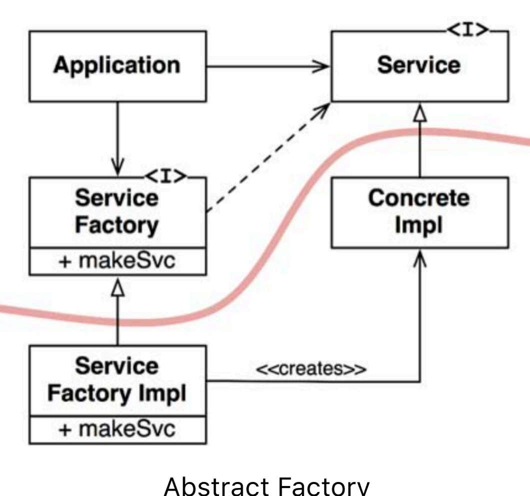
    - Example with different taxi services interfaces

URI	Dispatch Format
Acme.com	/pickupAddress/%s/pickupTime/%s/dest/%s
.	/pickupAddress/%s/pickupTime/%s/destination/%s

- **10. ISP: The Interface Segregation Principle**
  - Principle

    - User1 uses only op1 but when op2/op3 are changed then User1 should also be recompiled!
    - Segregated operations
  - ISP and Language
    - dynamically typed languages create systems that are more flexible and less tightly coupled than statically typed languages
  - ISP and Architecture
    - System S → Framework F → Database D
      - If no interfaces then changes in D will force changes in F and will force changes in S
- **11. DIP: The Dependency Inversion Principle**
  - Principle
    - The most flexible systems are those in which source code dependencies refer only to abstractions, not to concretions
      - We want to avoid dependencies on VOLATILE concrete elements
  - Stable Abstractions
    - Don't refer to volatile concrete classes
    - Don't derive from volatile concrete classes
    - Don't override concrete functions
    - Never mention the name of anything concrete and volatile
  - Factories

    - Abstract Factory
    - Dependency Inversion
      - The source code dependencies are inverted against the flow of control
  - DIP violations cannot be entirely removed
    - but they can be gathered into a small number of concrete components and kept separate from the rest of the system
  - Dependency Rule
    - the dependencies cross that curved line in one direction, and toward more abstract entities