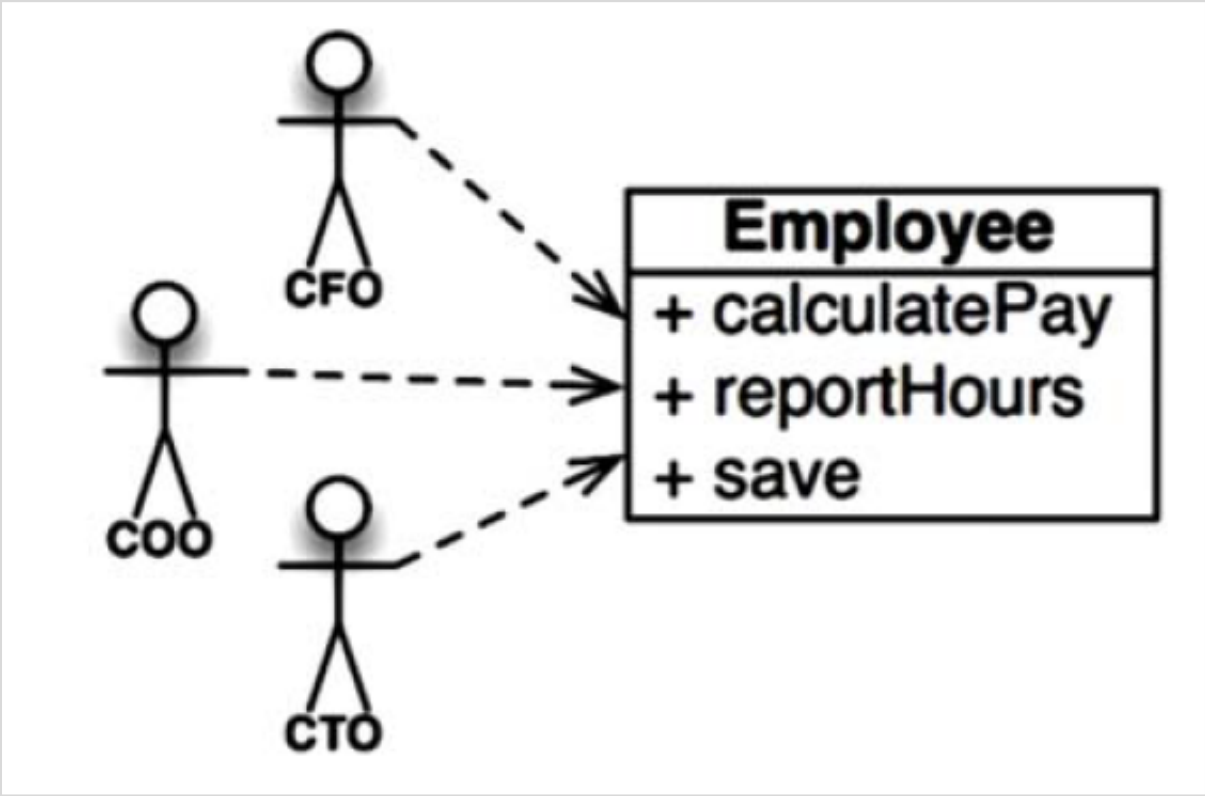
Chapter 7: The Single Responsibility Principle

- A module should be responsible to one, and only one, actor
  - Actor: a group of people who require a change
  - Module: a cohesive set of functions and data structures
- "Cohesion" is the force that binds together the code responsible to a single actor

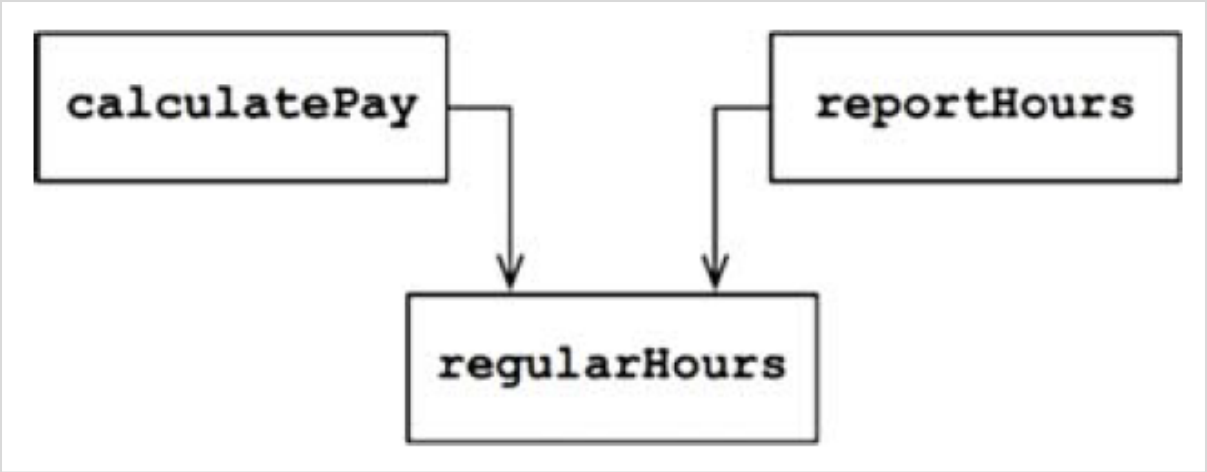
Symptom 1: Accidental Duplication

> Separate the code that different actors depend on



Payroll application's Employee class

- `calculatePay()` is specified by the accounting department, which reports to the CFO
- `reportHours()` is specified/used by HR, which reports to the COO
- `save()` is specified by DB admins, who reports to the CTO
- Developers have coupled each of these actors to the others
- Coupling can cause actions of the CFO's team to affect something that the COO's team depends on

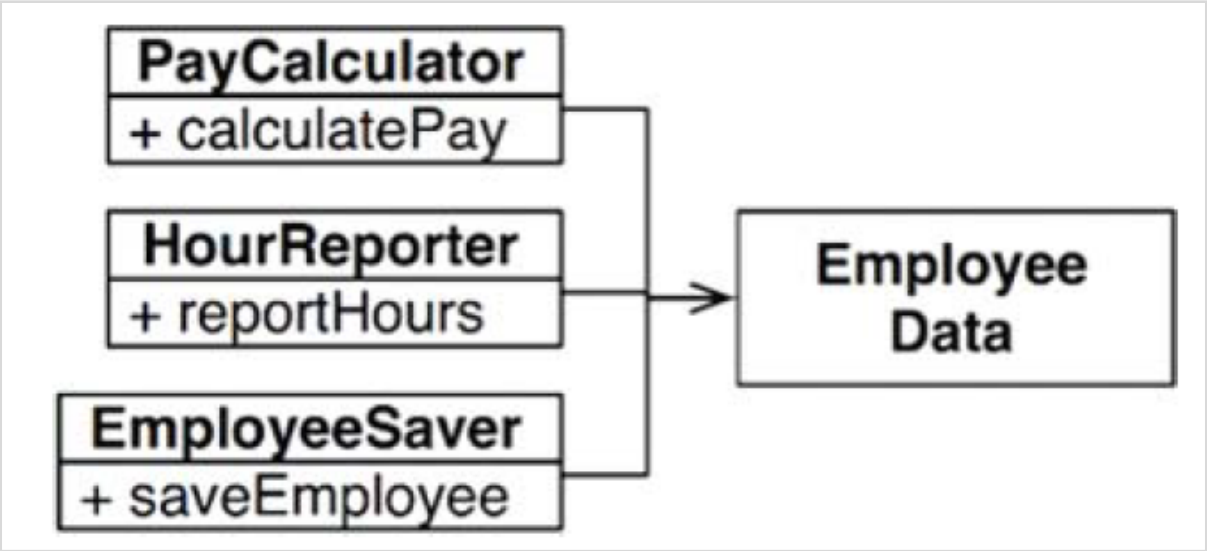


- `calculatePay` and `reportHours` both call `regularHours` to calculate OT hours.
- When `regularHours` is modified because of `calculatePay`, the CFO team knows, but `reportHours` is also affected, and COO's team is clueless

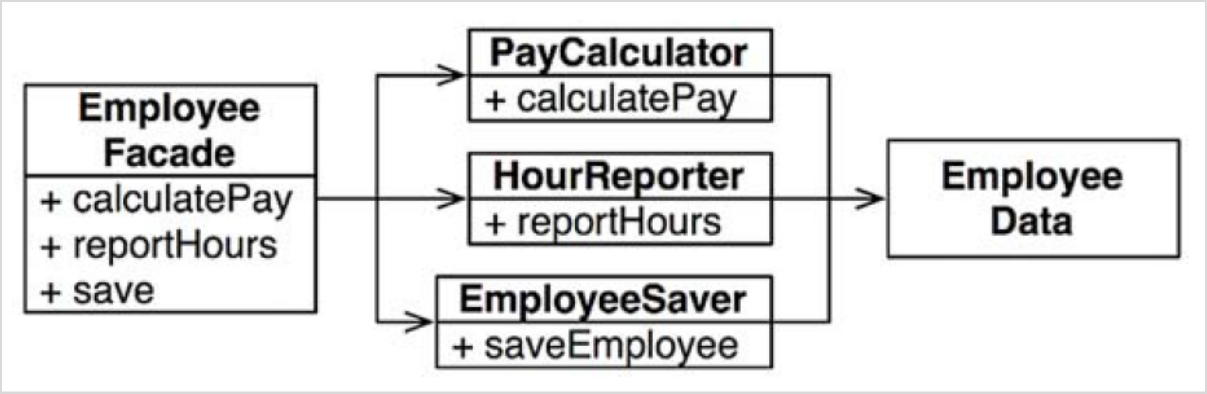
Symptom 2: Merges

- Merging code asks by two different actors in the same file
  - CTO wants schema changes to `Employee` table
  - COO wants change in the format of the hours report

Solution 1: Separate data from functions

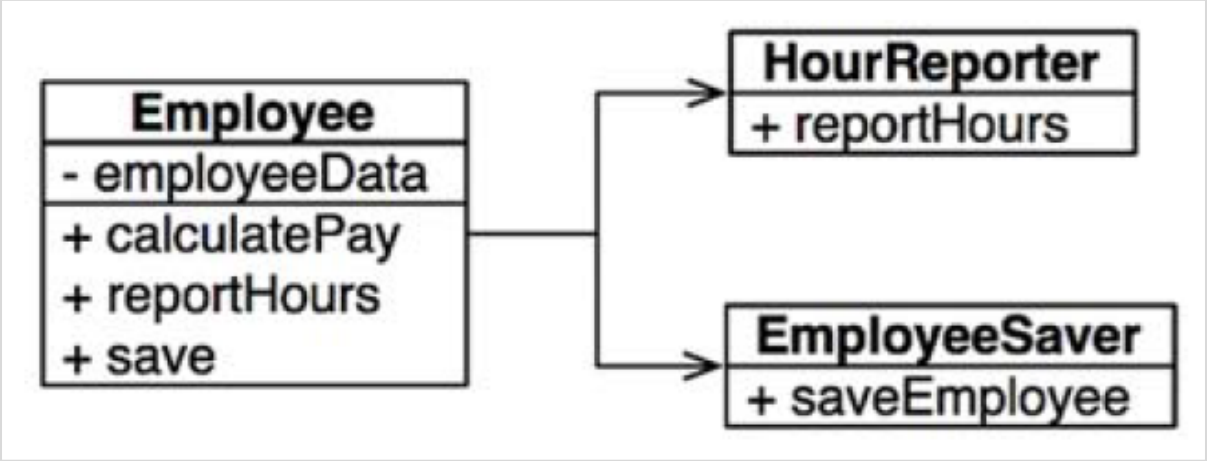


- `EmployeeData` only holds data structure
- Pros
  - The 3 classes do not know about each other
  - No more accidental duplication
- Cons
  - Devs now have three classes to instantiate and track
    - Solution: `Facade` pattern



`EmployeeFacade` is responsible for instantiating and delegating to the classes with the functions

Solution 2: Using the original class as a Facade class



Keep the most important method in the original `Employee` and using that class as a Facade for the lesser functions (`saveEmployee`)