Key Concepts

- Overall goal: reduce complexity
  - Dependencies
  - Obscurity
- Good design is an investment
  - Tactical vs. strategic programming
- Complexity is incremental: zero tolerance
- Abstraction: find simple ways to think about complicated things
- Information hiding
  - Interface vs. implementation
- Classes should be deep

- General-purpose classes are deeper
- Different layers should have different abstractions
- Pull complexity downward, push specialization upward
- Comments should describe things that aren’t obvious from the code
- Comments are at a different level of precision than code
- Names matter!
- Define errors out of existence
- Code should be obvious
Red Flags

- Shallow classes
- Unnecessary specialization
- Information leakage
  - Dependencies
  - Conjoined methods/classes
- Temporal decomposition
- Pass-through methods
- Code duplication
- Special cases
- Inconsistencies
- Comment duplicates code
- Implementation contaminates interface documentation
- Documentation has to be long to be complete
- Vague names
- Code is not obvious
Workloads

- If Project 1 was 1.0 unit of work, how many units were

- Project 2: 2.0, .75, 1.34, 1.2, .8, .85, 1.4, 1.2, .9, .7, .8, .85, 0.2, .8, 1.1, 1.8

- Project 3: .75, .7, .52, .8, .6, .75, .9, .8, .7, .9, .85, .75, 0.8, .7, .7, .5