Lecture 10: Core Principles and Best Practices for Software Design (Part I)

"Treat design as a wicked, sloppy, heuristic process. Don't settle for the first design that occurs to you. Collaborate. Strive for simplicity. Prototype when you need to. Iterate, iterate, and iterate again. You'll be happy with your designs."

-- Steve McConnell, *Code Complete* (*2nd ed.*), Ch. 5

Outline

- Best practices for software system design
- Time-tested design principles
  - With examples

Resources

- "*Code Complete*", 2nd ed., by Steve McConnell
  - Ch. 5: http://www.cc2e.com/docs/Chapter5-Design.pdf
- "*The Pragmatic Programmer*", by Andrew Hunt and David Thomas
  - Ch. 2 (section 7), Ch. 5 (section 26)
- "*On the Criteria to be Used in Decomposing Systems into Modules*", by David Parnas
- "*Design Patterns Explained*", by Alan Shalloway and James Trott

Why Learn How to Design?

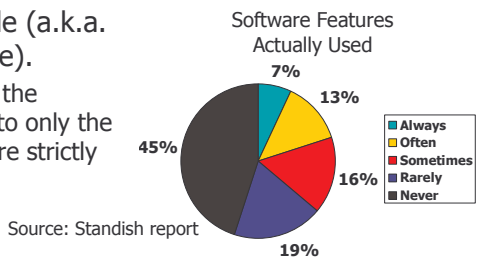
More Perspectives on How to Approach Design

"There are two ways of constructing a software design: one way is to make it so simple that there are obviously no deficiencies; the other is to make it so complicated that there are no obvious deficiencies."

-- C.A.R. Hoare (1985)

Best Practices for Software Design (1/5)

- Create at least three independent designs and choose the best one among them.
- Keep it simple (a.k.a. KISS principle).
  - Scale down the feature set to only the parts that are strictly necessary



Source: Standish report

Best Practices for Software Design (2/5)

- Ask yourself how you may test your components.
  - Testability is correlated with good design quality.
    - If you need to do extra work to test, something is likely wrong.
    - The culprit is usually tight coupling between modules.
- Do not invest too much into visualizing early designs – they will change substantially.
  - Write on index cards, rearranging and redrawing.
  - Write on whiteboards; take camera snapshots.
  - Avoid CAD tools and even UML-editing software – they are heavyweight and discourage making changes, so your design documents will quickly become obsolete.

Best Practices for Software Design (3/5)

- Learn and use design patterns.
  - Represent distilled knowledge about good designs
    - MVC is one well-known example.
  - Define a language to more effectively describe designs (e.g., façade, visitor, bridge, strategy, etc.)
  - Source: *Design Patterns Explained*, Alan Shalloway
- Consider if there are single points of failure or bottlenecks in your designs.
  - Can those be avoided or compensated for?

Best Practices for Software Design (4/5)

- Use abstractions as much as possible.
  - Example (of what not to do):

```
class Square {
        double lower_left_x_coord;
        double lower_left_y_coord;
        double lower_right_x_coord;
        double lower_right_y_coord;
        ...
}
```

- Encapsulate changing components; fix the interfaces between them.
  - Why not allow interfaces to change in order to enable the addition of new components later on (as needed)?!
  - Reference: *On the Criteria to be Used in Decomposing Systems into Modules*, David Parnas

Best Practices for Software Design (5/5)

- Favor composition over inheritance.
  - Example: