

CSE403  
Lecture 6:

---


**Best Practices for  
Software System Design**

Valentin Razmov, CSE403, Sp'05




## Techniques for Design

- "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, 2<sup>nd</sup> ed.*, chap 5,  
<http://www.cc2e.com/docs/Chapter5-Design.pdf>
- Best practices for software system design
- Standard notations for expressing designs
  - Dataflow / state diagram
  - Class diagram
  - Sequence diagram




## Notations: Dataflow / State Diagram

- High-level, coarse grained
- Used to describe interactions between the components of a system
- Example:




## Notations: Class Diagram

- Medium-level
- Used to describe the relationships between classes (modules) in the system
- Example:



## Notations: Sequence Diagram

- Low-level, fine-grained
- Used to describe sequences of invocations between the objects that comprise the system
- Example:



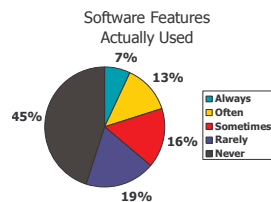
## Your Questions on Notations for Software Design

## Best Practices for Software Design

- Create at least three (3) independent designs and choose the best among them.

- Keep it simple (a.k.a. KISS principle).

- Scale down the feature set to only the parts that are strictly necessary



Reference: Standish report, <http://www.standishgroup.com/>

## Best Practices for Software Design (cont.)

- Ask yourself how you may test your design / components easily.
  - Testability is correlated with good design quality.
    - If you need to do extra work to test, something is wrong.
    - The culprit is usually tight coupling between modules.
- Do not invest too much into drawing early designs - they will change substantially.
  - Write on index cards, rearranging and redrawing.
  - Write on whiteboards and take snapshots with a camera.
  - Don't use UML or other CAD tools - they discourage changes and your design documents will quickly become obsolete.

## Best Practices for Software Design (cont.)

- Learn and use design patterns as much as you can.
  - They are distilled knowledge about good designs.
  - Reference: *Design Patterns Explained (2<sup>nd</sup> ed.)*, Alan Shalloway
- Establish and maintain a clear audit trail.
  - It requires little investment upfront but is invaluable for debugging purposes.

## Best Practices for Software Design (cont.)

- Consider if there are any single points of failure or bottlenecks in your designs.
  - Can those be avoided or compensated for?
- Make the common case fast and the uncommon case correct.
  - But do not spend time on optimizing code early on.
  - "*It is easier to optimize correct code than to correct optimized code.*" --Donald Knuth


## Best Practices for Software Design (cont.)

- 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
- Use abstractions as much as possible.
  - Example (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;
    ...
}
```

## Best Practices for Software Design (cont.)

- Favor composition over inheritance.
  - Example:
- A module should have a single responsibility.
  - Principle of strong cohesion
  - "God-object" metaphor



Your Questions on Best Practices for Software Design