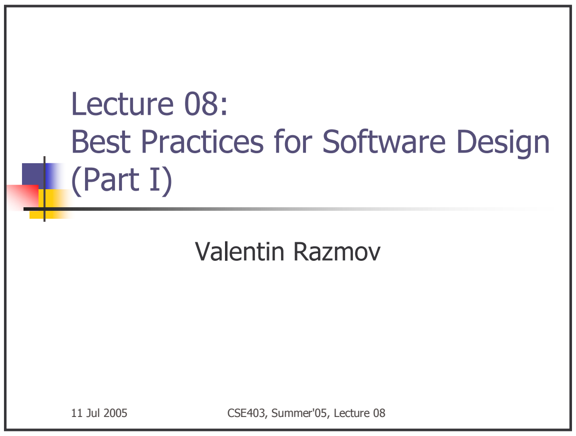
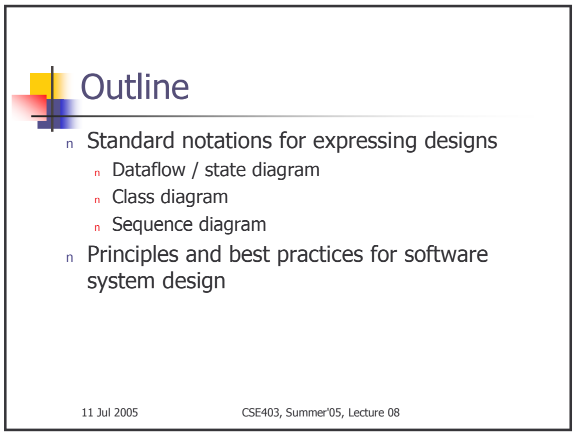


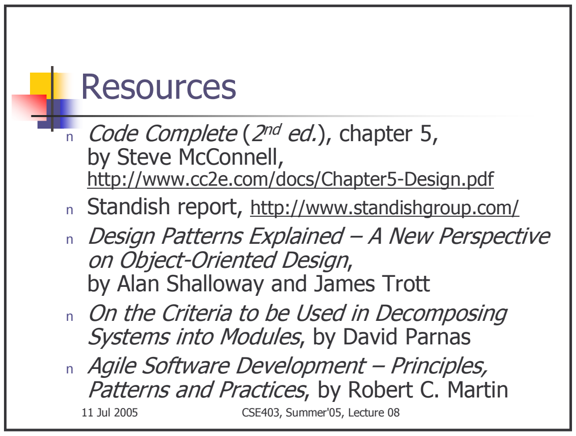
Lecture 08: Best Practices for Software Design (Part I)

Valentin Razmov



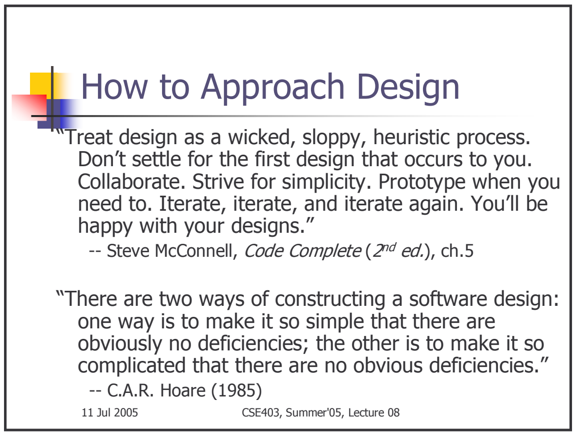
Outline

- Standard notations for expressing designs
  - Dataflow / state diagram
  - Class diagram
  - Sequence diagram
- Principles and best practices for software system design



Resources

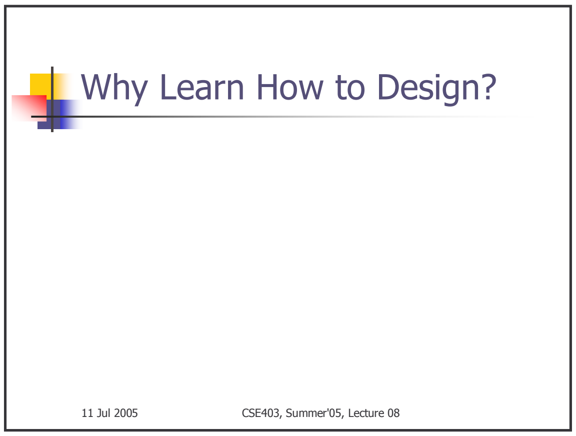
- *Code Complete* (*2nd ed.*), chapter 5, by Steve McConnell, http://www.cc2e.com/docs/Chapter5-Design.pdf
- Standish report, http://www.standishgroup.com/
- *Design Patterns Explained – A New Perspective on Object-Oriented Design*, by Alan Shalloway and James Trott
- *On the Criteria to be Used in Decomposing Systems into Modules*, by David Parnas
- *Agile Software Development – Principles, Patterns and Practices*, by Robert C. Martin



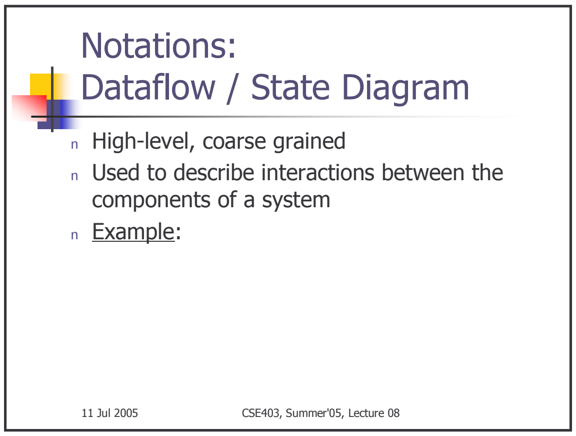
How to Approach Design

"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* (*2nd ed.*), ch.5

"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)



Why Learn How to Design?



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:

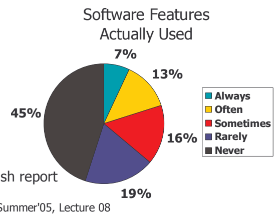
Notations: Sequence Diagram

- 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 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



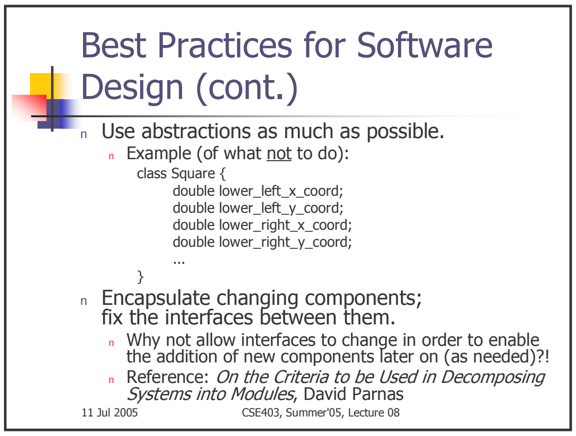
Source: Standish report

Best Practices for Software Design (cont.)

- 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 drawing early designs – they will change substantially.
  - Write on index cards, rearranging and redrawing.
  - Write on whiteboards; take camera snapshots.
  - Don't use UML or other CAD tools – they discourage making changes, so your design documents will quickly become obsolete.

Best Practices for Software Design (cont.)

- 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 (cont.)

- 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

11 Jul 2005 CSE403, Summer'05, Lecture 08