CS 370
REVIEW:
Introduction to Design

DR. MICHAEL J. REALE
FALL 2015
Challenges of Design

• Design is:
  ○ A wicked problem
  ○ A sloppy process (even if it produces a tidy result)
  ○ About tradeoffs and priorities
  ○ Involves restrictions
  ○ Nondeterministic
  ○ A heuristic process

• In other words, design is “emergent”
  ○ Don’t spring fully formed from designer’s brain
  ○ Designs evolve and improve
Challenges: Wicked Problem

- **Wicked problem**
  - Can only be clearly defined by solving all or part of it
  - Implies:
    - Define problem → “solve” it → define it better → “solve” better → etc.

https://upload.wikimedia.org/wikipedia/commons/4/4a/Tacoma-narrows-bridge-collapse.jpg
Software’s Primary Technical Imperative

• Software’s Primary Technical Imperative → managing complexity!
How to Attack Complexity

- Two-prong approach:
  1) Minimize amount of essential complexity that anyone’s brain has to deal with at any one time
  2) Keep accidental complexity from needlessly proliferating

- ALL OTHER TECHNICAL GOALS SECONDARY to MANAGING COMPLEXITY.
Desirable Characteristics of a Design

- **Minimal complexity**
  - Avoid “clever” designs → make “simple” and “easy-to-understand” designs
  - Can safely ignore other parts while working on specific part

- **Ease of maintenance**
  - Make design self-explanatory

- **Loose coupling**
  - As few interconnections as possible
  - Minimizes work during integration, testing, and maintenance

- **Extensibility**
  - Can enhance a system without causing violence to underlying structure

- **Reusability**
  - Design for reuse of parts in other systems (and same system)
Desirable Characteristics of a Design

- **High fan-in**
  - Refers to having a high number of classes that use a given class → e.g., good utility classes

- **Low-to-medium fan-out**
  - Having a given class use a low-to-medium number of other classes

- **Portability**
  - Can move easily to another environment

- **Leanness**
  - Designing system so that it has no extra parts

- **Stratification**
  - Can view system at same level and get consistent view

- **Standard techniques**
  - Use Design Patterns where reasonable and useful
1. Software System

2. Division into subsystems/packages

3. Division into classes within packages

4. Division into data and routines within classes

5. Internal routine design
Subsystems and Communication

- Communication between subsystems
  - Need-to-know basis
    - Can always relax later on
  - Avoid cycles
Common Subsystems

- **Business Rules**
  - Laws, regulations, policies, and procedures of system

- **User Interface**
  - Classes for GUI, menu, windows, help systems, etc.

- **Database Access / Data Storage**
  - Change storage method → doesn’t break everything

- **Graphics**
  - Wrap graphics-specific code

- **System Dependencies**
  - Wrap OS-specific functionality

- **Application Level Classes**
  - App-specific classes that do not include any of the above
Design Practices

- Iterate
- Divide and Conquer
  - Break the problem/project into manageable, self-contained chunks
  - Design each chunk
- Top-down vs. Bottom-up
  - Top-down
    - Start at highest-level → work down
    - Easier, defers construction details
    - *Problem*: low-level complexity can rise to top
  - Bottom-up
    - Start with what system needs to do (low-level) → generalize/work up
    - More tangible, encourages reuse, design more compact
    - *Problems*: more difficult, pieces may not work as system
  - Both
    - Complimentary
- Experimental Prototyping
  - Writing the **ABSOLUTE MINIMUM** amount of **throwaway** code needed to answer a *very specific question!*