Outro

Insu Yun

What is a system?

- A **system** is a set of interconnected components that has an expected behavior observed at the interface with its environment.

- Examples
  - A personal computer
  - The onboard engine controller of an automobile
  - The Internet
  - …

4 common problems of systems

- Emergent properties
- Propagation of effects
- Incommensurate scaling
- Trade-offs

1. Emergent properties

- **Emergent properties** are properties that are not evident in the individual components of a system but show up when combining those components

- Example
  - Bottlenecks in large-scale distributed systems
  - Consistency issues in distributed databases
  - Unanticipated vulnerabilities in a security system
  - …

2. Propagating of Effects

- What looks at first to be a small disruption or a local change can have effects that reach from one end of a system to the other.

- Example)

  Change the tire size of a production model car from 15 to 15 inches
  -> redesigning the wheel wells, enlarging the spare tire space, rearranging the trunk that holds the spare tire, and moving the back seat forward slightly to accommodate the trunk redesign
  -> ...

3. Incommensurate scaling

- As a system increases in size or speed, not all parts of it follow the same scaling rules, so things stop working.

- Example) Mouse -> Elephant (Haldane 1928)
  - Mouse has a particular skeleton design
  - Scaling mouse to size of an elephant
    - Volume ~ O(n3)
    - Bone strength ~ cross section ~ O(n2)
    - Mouse design will collapse
  - Elephant needs different design than mouse

4. Trade-offs

- There is a limited amount of some form of goodness in the universe, and the design challenge is first to maximize that goodness, second to avoid wasting it, and third to allocate it to the places where it will help the most.

- Example)
  - A hardware circuit to run at a higher clock rate,
    -> increases both power consumption and the risk of timing errors.
  - Binary classifiers: Accuracy vs False positives

Complexity in system building

- Many goals / requirements
  - Example
    1975 Unix kernel: 10,5000 lines of code
    2024 Linux kernel: Millions lines of code

- Interaction of features

- Performance

Coping with complexity I

- Modularity
  - Split up system, consider separately
- Abstraction
  - Interfaces/hiding
  - Helps avoid propagation of effects
- Hierarchy
  - Reduce connections
  - Divide-and-conquer
- Layering
  - Gradually build up capabilities

Coping with complexity II

- Why modularity, abstraction, layering and hierarchy aren't enough?
  - One design should be chosen among the many design candidates

- Solution: Iteration
  – You won't get it right the first time, so make it easy to change!

Good programming practice

- Test-driven development
  - Good modularity + abstraction
  - Good code is easy to test!
- Refactoring: a change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior.
- Software design patterns: Known ways to build simple software