Introduction to Computer Systems

Review–What is a computer systems?

- “A system is a set of **interconnected computing components** that has an expected behavior observed at the interface with its environment”.
- Essentially defined by **complex**
- Today, and throughout this course, we’re going to study the symptoms of this complexity, and the techniques we can use to minimize their impact.

Complexity In Systems

How do we know if a system is complex?

1. Large number of components
2. Large number of interconnections
3. Lack of regularity–many different components
4. A long description
5. A team of designers

The Problems with Complexity

1. Emergent Properties–Properties that only show up when combining individual components.
2. Propagation of Effects–Problems in one component affect other components
3. Incommensurate Scaling–Not all parts of a system scale at the same rate.
4. Tradeoffs–constraints mean that increasing one characteristic often harms another

Why are computers so complex?

1. Cascading and interacting requirements–each new requirement interacts with all existing requirements
2. Maintain High Utilization
3. Nearly no bounds on composition
4. Technology growth is unprecedented!
5. Fundamentally limited by human cognition rather than by physical laws.

Solutions–So, what do we do about complexity?

Modularization: Divide-and-Conquer

1. **Bottom-up Modularity**: when you create modules as you go
2. **Top-Down Modularity**: sometimes, you plan ahead!
3. Why?
    a. Hides complexity
    b. Isolates failure
    c. Improves debugging
    d. Improves interchangeability
4. Allows us to use what we already know (how do we write small programs) to solve really big problems

Abstraction: The key to making modularity work

1. How do you know that you drew a good module?
    a. If you've limited interactions!
    b. Critical for all of the prior advantages!
    c. If you did that, you've built a **powerful abstraction**.
2. Separation of interface from internals. The "how to" of modularization.
3. Involves committing to a contract with the user of the module. The user can think of the module as a **name** instead of as an implementation.
4. **Robustness Principle**: Be tolerant of inputs (try to do the right thing) be strict on outputs (always produce the right output).

How do we arrange our modules?

1. Layering: Arrange modules "in a straight line".
2. Hierarchical: arrange the modules "in a tree".

What else do we do with fast paced technology?

1. Iteration: you won't get it right the first time, so make it easy to change
2. Keep it Simple: