Managing Complexity: Systems Design

March 2, 2001

What makes Computer Systems Special?

- Complexity not limited by physical laws
  - Noise does not limit computer system complexity computers are digital
- Technology changes at an unprecedented rate
  - The technology we build helps us to build better technology

➡ Computer systems can be more complex than planes, trains, or automobiles

Managing Complexity

- Modularity
- Abstraction
- Hierarchy
- Layers

➡ Knowing when and how to most effectively use these will take practice

Modularity

- Also known as "Divide and Conquer"
  - This approach that words best for the Bank Vault Problem
- Idea - Group the system components into a smaller set of interacting subsystems
  - Which dimension of complexity does this reduce?
- Example - compilation of a large program

Modularity, cont.

- Enforce boundaries between modules
- E.g., assume input can be wrong (outside of specs), but ensure output is always within specs
- Examples:
  - Tolerances in components (Henry Ford, Rifles for the U.S. Army),
  - Digital logic gates

Abstraction

- Hide implementation details behind well-specified interfaces
- Choose abstractions to minimize inter-module interactions
- Align modules (or groups of modules) and abstractions
- Careful with performance tradeoffs

Hierarchy

- Organize modules into a tree-like structure
  - each node represents a set of modules
  - Modules can interact only along the links
- Examples: Businesses
- Reduces interactions from $O(n^2)$ to $O(n)$

Layers

- Organizing technique that uses modularity and abstraction
- Create a different way of looking at an existing system, without adding more functionality
- Example 1: logic gates + memory cells: microprocessor: machine language processor: Java processor

Managing Complexity - what else?

- How do we pick the right form of modularity, abstraction, hierarchy, or layers?
  - E.g., Architecture of Complexity paper illustrates many ways hierarchies are used
- We need more help to manage complexity:
  - Iteration
  - KIS

Iteration

- Start with a simple, working system that meets a few key requirements (sound familiar?)
- Use what you learn to evolve the system
- Learn from your mistakes early on (study failures!)
- Import new technology as it arrives
- Tradeoffs?
  - "The kitchen sink"
  - Harder to change early decisions as time goes on

KIS

- Keep It Simple
- "If in doubt, leave it out" - anonymous
- "Everything should be made as simple as possible, but no simpler" - Albert Einstein
- Problem - Convincing someone that leaving something out will help in the long term
  - "we need more features to beat competitors!"
  - "we've already tested each of the 100 features on its own - all we have to do is integrate them"

Putting it all together

- Frameworks for analyzing systems
  - What are we dealing with?
    - System - Components, Interface, Environment
      - Purpose and granularity
    - Problem, Approach, Experiment, Results, Conclusions, Next Steps
    - Assumptions, Objectives, Tradeoffs
      - Objectives: Speed, Robustness, etc.
  - Where do we look for issues?
    - Emergent effects, propagation of effects, incommensurate scaling

Putting it all together, cont.

- Frameworks for designing systems:
  - How do we detect complexity?
    - Number of components, number of interactions, etc.
  - How do we control complexity?
    - Modularity, Abstraction, Hierarchy, Layers
    - Iteration, KIS

Can airplanes really fly?

- Weight grows with cube of size, whereas lift (based on cross-section) grows w/the square
- At small dimensions, (e.g., a kite, a bird), the lift can support the weight
- Larger flying machines based on winged-lift, however, will not work

Help an airline design its traffic grid

- Hub & spoke or multi-stop traffic?
- Objectives:
  - Market share
  - Low cost
- Assumptions:
  - Fixed delay between airplane landing and taking off (1 hour)
  - Average flight - 2 hours long
  - What else?