Complexity in Computer Systems

Monday, November 19, 2007
5:12 PM

- We're going to talk about "system perspective" in computer system design.
  - Think about system in a global way.
  - Abstract details.

- Common problems about systems.
  - Emergent properties
    - Properties of a system that you cannot infer from properties of individual components.
    - "Surprises"
  - Propagation effects
    - A local change can have wide-ranging effects.
  - Incommensurate scaling
    - Not all part of a system follows the same scaling rules.
  - Trade-offs
    - You cannot have all you want.
    - Waterbed effect = solving one problem causes another problem elsewhere.

- System vocabs
  - System = a set of interconnected components that has a specified behavior observed at the interface with its environment.
  - What are system components?
    - Depends on your point of view.
      - Purposes
      - Granularity
  - **Computer system** or **information system** = system built to store, process, and/or communicate information under automatic control. We are interested in digital system.

- Complexity
  - Signs
    - Large number of components.
    - Large number of interconnections
    - Irregularity.
    - Lack of methodical description.
    - A team of size N>1 to understand, construct, and maintain the system.
  - Sources of complexity
    - Cascading and interacting requirements.
      - Generality = meeting many requirements with a single design.
      - Generality contributes to complexity directly and indirectly.
        - Directly: Well, more interacting requirements.
        - Indirectly: Users of a very general system will use the system differently so that they can suppress that part that they do not want. As such, they cannot communicate with others well about the same system.
      - Requirements also change.
      - As a system ages, it will accumulate requirements and become more complex. The lifetime of a system is until its complexity is too much to understand.
    - Maintaining high utilization
      - You have resource. Needs to use resource efficiently. Performance always gets in the way of everything.

- Coping with Complexity
  - Modularity
    - Design systems as a collection of interacting subsystems or modules.
    - Advantages

        - Can think about interactions between components of a module without thinking about their interactions with things outside the module.
        - Can replace modules with better ones.
    - Abstraction
        - Make sure that modules can treat others knowing only the interface.
        - Software is particularly prone to abstraction violation because the boundary between modules are soft. There's always dangling pointers and things like that when you program with C or C++.
        - Robustness principle
            - Be tolerant on inputs and strict on outputs.
            - Namely, accept inputs that is slightly out of the scope you intend, and make your output more precise than it needs to be.
            - Robustness principle suppresses noise.
            - Very successful in digital system: see static discipline.
    - Hierachy
        - Make a bigger module of a small group of modules. Make even bigger modules out of the bigger modules. Repeat until you get the whole system.
        - Hierarchy helps limit interaction. From O(n^2) to O(n), in particular.
    - Layers
        - Take a set of mechanisms that is already complete, and use them to create a different complete set of mechanisms.
        - Layers does not provide new functionality, but it recasts the functionality so that it becomes easier to use and compose. Namely, it helps with understanding.
    - Names
        - Modules should refer to one another by names, so that we can replace the underlying modules easily.
        - **Binding** = choose among various implementations of a module.
        - Delayed binding = name a functionality, not implement it. Sometimes called **indirection**.
        - "Every problem in computer science can be solved by adding a layer of indirection."

- Computer systems are different!
    - Computer systems have no nearby bounds on composition.
        - Computer systems are digital.
            - Thanks to static discipline.
                - The set of analog value that a device accepts as ONE (or ZERO) is a lot larger than the value that the device will output as ONE (or ZERO).
            - Noise do not propagate in digital systems.
            - So you can compose them until you cannot understand them any more.
        - Computer systems are controlled by software.
            - Software has no physical limit on compositions.
            - Abstraction should help control software complexity, but software abstractions are always leaky.
            - Developers are always tempted to add more features.
    - Unprecedented rate of change in technology.
        - Moore's law!
        - By the time a system is finished, the technology changes so much that the design may become obsolete or no longer work. Incommensurate scaling is like to occur if the designers are to incorporate the new change to the system. A new design is needed altogether.
        - There's no time to weed mistakes out of the old designs. No time to find tuning. No time for thorough analysis.
        - Brute-force solutions might be the right approach for a lot of problems.
        - Usability and "human engineering" factors are always neglected.
        - Logical and judicial processes are always behind.
            - Think Bittorrent, or the Computer Crime Bill.

- Coping with complexity in computer systems
    - The previous tools are not enough!

        - It's hard to figure out the right modularity, abstraction, hierarchy, and layers.
    - Iteration
        - Design the system so that you can change it easily.
        - Take small steps so that you can discover mistakes easily and quickly.
        - Don't rush. Plan every step well.
        - Plan for feedback.
        - Study failures and learn from it.
        - Don't lose conceptual integrity after iterations.
    - Keep it simple
        - So that designers can make compelling arguments about correctness.
        - So that it's clear to everyone what's going on.