Today: **Introduction to Software Architecture**

**Software “Architecture”?**

- We’re leaning on the construction metaphor
  - What do people refer to when they talk about “architecture”?

**Software Architecture Reminder**

- Simple definition: A software architecture is the structure of a system
  - Consider:
    - Software components
    - Their relationships
    - Interfaces to the external world

- Dealing with components is an abstraction
  - Ignore lower level details
    - What is the color of the pillars?
    - How does the sorting algorithm work?
    - Who cares at this point?

- Note that there is more than one structure in a system
  - Module structure
    - Which module uses which? who calls whom?
  - Process execution structure
    - What is the chain of events that occurs when we receive input?
  - File structure
    - Databases? Libraries?
  - Physical structure
    - Network layout? Types of computers?

- Types of structures crosscut each other, so they need to be considered and handled simultaneously
Architecture and Design

• Do not confuse them!
  – When dealing with the system level we do architecture
  – When dealing with code, classes, etc., we do design

• Architecture...
  – Deals with the high-level construction of a system
    • Technology choices (language, platform, database)
    • System construction (overall structure – monolithic, 3-tiered?)
    • Modules and programs

• Design...
  – Deals with how and where to "lay down" code
    • Classes, methods, and attributes
    • Design patterns
    • Dependencies among classes
    • Subsystems, (Java) packages

• But note that the boundary between architecture and design is blurry

Why is architecture important?

• Set out the key elements and aspects of the software system
  – The most difficult to correct, the hardest to change
  – The ones that defines implementation constraints
  – The ones that enables or inhibit quality attributes (e.g. security, performance)

• Treating a system as components allows for narrower focus
  – Divide and conquer
  – Easier organization (team A works on module X, team B works on module Y)
  – Easier estimation

• Architecture documents enable early discussions on possible solutions

• Architecture documents allow for review
  – Training tools
  – Progress indicators

A word on documentation

• Does not have to be extremely detailed
  – Most of the times, annotated boxes and arrows will do

• But it does have to be extremely clear
  – For yourself
  – For your future self
  – For software designers
  – For new developers learning about your system
  – For the technical documentation

• Lack of detail ≠ Lack of clarity
  – For every component
    • State its nature and main tasks
  – For every relationship among components
    • State who depends on whom, what sort of information is passed
  – For every external interface
    • Standards and protocols used, or at least a high-level description of the kinds of interactions

OK, OK, architecture is important. How do I do it?

• Well...
  – There’s really no structured way to do it
  – All I can say is, if:
    • You know and understand the requirements, AND
    • Have some domain experience, AND
    • Have paid attention to other systems’ architectures...
    • …then the shape of the system will start to form on your mind
    • Yes, it’s a little bit of a black art

• But I can give you some tips
  – Reading and studying other architectures is essential
  – Iterations are good
  – Documenting the iterations is better
  – Getting feedback on each iteration is even better
  – Trying out or simulating your latest iteration “in the small” is best
“Non-Functional” Requirements must be satisfied

- Non-Functional Requirements (NFRs): All those system qualities that can’t really be expressed as features
  - Performance
  - Usability
  - Security
  - Availability
  - Robustness
  - ...
- The (browser, operating system, IM client) with the most features won’t take off if
  - It’s slow
  - We can’t make sense of it
  - Has some glaring security holes
  - ...
- Architectural work is the key time to address these problems

There are plenty of NFRs to consider

- The Usual Suspects
  - Performance
  - Usability
  - ...
- Those that facilitate production and maintenance
  - Conceptual integrity
  - Modifiability
  - Reusability
  - Testability
  - ...
- Those needed to keep the business running
  - Cost
  - Time
  - Projected lifetime
  - ...

Successful architectures address two issues

- First issue: What is the best structure to satisfy the system’s functional and behavioral requirements?
  - That is, so that it does everything it is supposed to do
  - ...at the level of quality that we require?
- Second issue: What is the best structure to ensure that we can build the system given
  - Our skills and assets
  - Our business constraints (or assignment deadlines!)
  - Our competitors’ offerings
- And the most important question, what is the best structure to satisfy both problems?