Software Engineering 301: Software Analysis and Design

Software architecture

Agenda

- Basic concepts
- Partitioning software
- Architectural styles

Software-in-the-large

- Consider a system from a very high-level : What do we see?
  - collection of software elements
    - classes, modules, components, ...
  - interfaces of elements
    - public classes, sets of operations, contracts, ...
  - interactions among elements
    - method calls, remote procedure call, IP, ...

High-level vs. low-level design

- Low-level design involves
  - Details of individual objects
  - Abstraction into classes
  - Interactions amongst objects
- High-level design is useful
  - To abstract away details of individual classes
  - To subdivide a system into more manageable pieces
  - To consider how the pieces will fit onto pieces of hardware

Software architecture

... “the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them”

Defining an architecture

- Decide what big pieces your system should be divided into (*subsystems*)
- Decide how these pieces will be put into different machines (nodes) or processes (*deployment*)
- Decide how the pieces will communicate with each other

Agenda

- ~~Basic concepts~~
- Partitioning software
- Architectural styles

How to partition a system?

- Hardware considerations
  - Pre-existing applications
    - (e.g., database server)
  - Scalability and expected scale
- Computational costs
  - Look to distribute/parallelize expensive operations
- Divide up along logical divisions
  - (e.g., UI, business logic, database, …)

Communication

- Intraprocess
  - Within single method: CHEEEEEAPPPP!
  - Between methods in same thread: CHEAP
  - Between methods in different threads: cheapish
- Interprocess, same node (machine)
  - Potentially expensive
  - Custom protocol?
  - Security concerns?

Communication

- Interprocess, different nodes
  - Increased computational power
  - Increased scalability
  - Increased communication costs
  - Which protocols?
  - Security!

Kinds of partition

- Informal groupings
- Packages
  - Convenient to collect related classes
  - Minor visibility restrictions possible
- Components
  - Formal communication boundaries
  - Hide internal implementations
  - Need specialized environments
  - Require decisions about communication protocols, security, deployment, …

Agenda

- ~~Basic concepts~~
- ~~Partitioning software~~
- Architectural styles

Architectural style

- A way of describing a “standard” layout of pieces
- Note that an architectural style is not an architecture

Architectural style is to an *architecture* as
design pattern is to a *design*

- An architectural style has to be filled in with concrete details appropriate to your context, to form an architecture

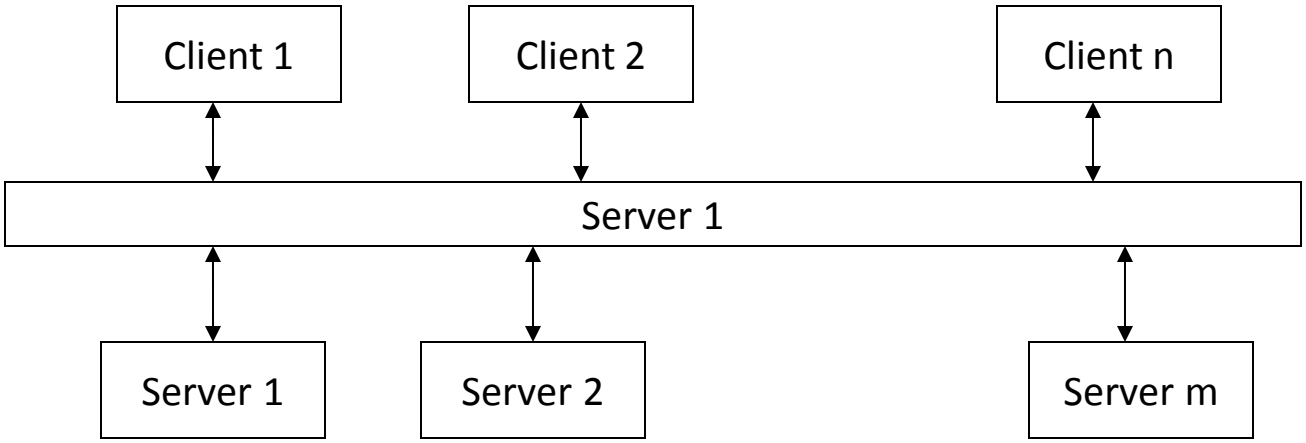
Common architectural styles

- **client/server**
- pipes-and-filters
- **repository**
- model-view-controller
- **layered (three-tier, four-tier)**
- peer-to-peer
- interpreter
- plugin
- component-based
- event-based

Sometimes, it's quite appropriate to apply more than one style

Client-server style

- Elements:
  - set of stand-alone servers (sub-systems)
  - set of clients (other sub-systems)
  - network that connects them



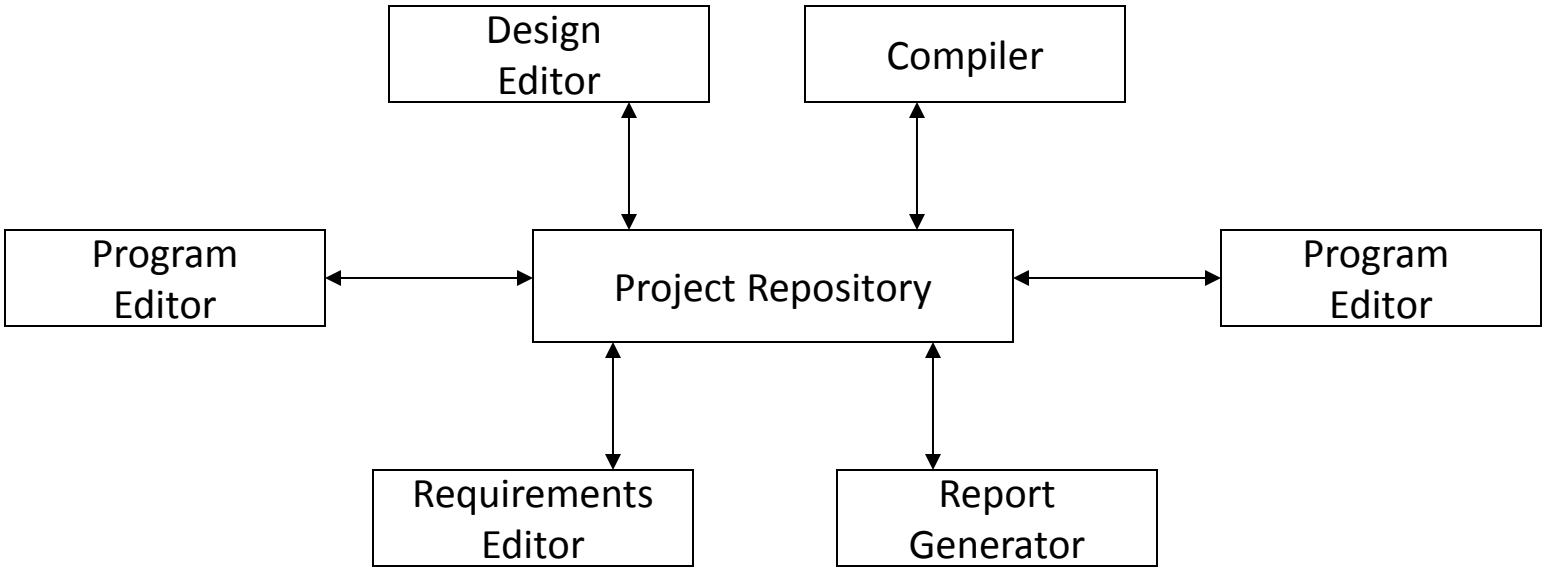
Client-server: pros and cons

+ Good for distributed applications
+ Easy to add new servers
+ Server performance can be optimized

– Each server must manage its data by itself
– Communication overhead
– Security concerns

Repository style

- Elements share data in a central database
- Data is communicated through message passing



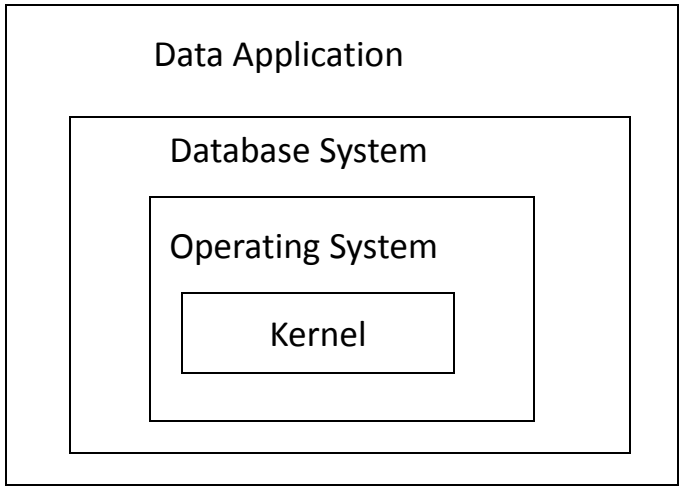
Repository: pros and cons

+ efficient for sharing of large data
+ security, access control, recovery, etc. are centralized
+ can add new tools
  $\Rightarrow$ each must agree with the data model

– sub-systems must agree on data model
– synchronization must be handled carefully for good performance
– difficult to distribute repository

Layered style

- Sub-systems are organized into layers
- Each layer:
  - uses the services of the layer below
  - provides services to layer above
- “Abstract machines”
- The terms “tier” and “layer” are interchangeable, for most people



Layered: pros and cons

+ Supports incremental development
+ Comprehension is easier

– Structuring can be difficult
– Performance *can* be affected

  $\Rightarrow$ to avoid this, systems allow bypassing of layers

    $\Rightarrow$ affects comprehensibility, security, functionality, etc.

Next time

- Design analysis