Architectural Styles

Reid Holmes
Objectives

- What are the benefits / pitfalls of different architectural approaches?
- What are the phases of the design process?
- What are some alternative design strategies? When are they necessary?
- Define: abstraction, reification, and SoC
- Identify key architectural style categories
Architectural approaches

- Creative
  - Engaging
  - Potentially unnecessary
  - Dangerous

- Methodical
  - Efficient when domain is familiar
  - Predictable outcome
  - Not always successful
Design process

1. Feasibility stage:
   - Identify set of feasible concepts

2. Preliminary design stage:
   - Select and develop best concept

3. Detailed design stage:
   - Develop engineering descriptions of concept

4. Planning stage:
   - Evaluate / alter concept to fit requirements, also team allocation / budgeting
Design strategies

- Standard
- Cyclic
  - Revisit earlier stages
- Parallel
  - Split off #2 or #1 in parallel
- Adaptive
  - Plan next stage with insights from current
- Incremental
  - Update all stages as experience is gained
Abstraction

Definition:

“A concept or idea not associated with a specific instance”

Top down

Specify ‘down’ to details from concepts

Bottom up

Generalize ‘up’ to concepts from details

Reification:

“The conversion of a concept into a thing”
Level of discourse

- Consider application as a whole
  - e.g., stepwise refinement
- Start with sub-problems
  - Combine solutions as they are ready
- Start with level above desired application
  - e.g., consider simple input as general parsing
Separation of Concerns

- Decomposition of problem into independent parts
- In arch, separating components and connectors
- Complicated by:
  - Scattering:
    - Concern spread across many parts
      - e.g., logging
  - Tangling:
    - Concern interacts with many parts
      - e.g., performance
Architectural patterns

A set of architectural design decisions that are applicable to a recurring design problem, and parameterized to account for different software development contexts in which that problem appears.

e.g., Three-tier architectural pattern:
Architectural styles

- Some design choices are better than others
  - Experience can guide us towards beneficial sets of choices (patterns) that have positive properties
    - Such as?
  - An architectural style is a named collection of architectural design decisions that:
    - Are applicable to a given context
    - Constrain design decisions
    - Elicit beneficial qualities in resulting systems
Architectural Styles

- Language Based
  - Object-oriented
    - Main program & Subroutines
  - Virtual Machine
    - Client Server
- Layered
  - Virtual Machine
    - Client Server
- Dataflow
  - Pipe-and-Filter
- Peer-to-Peer
  - Peer-to-Peer

Shared Memory
- Rule-based
  - Blackboard
- Mobile code
  - Event-based

Implicit Invocation
- Publish-subscribe
  - Event-based

Interpreter
- Rule-based
  - Interpreter
- Mobile code
  - Interpreter
  - Interpreter
- Event-based
Lunar lander example
Style: Main program & subroutine
Style: Main program & subroutine

- Decomposition of functional elements.
- Components:
  - Main program and subroutines.
- Connections:
  - Function / procedure calls.
- Data elements:
  - Values passed in / out of subroutines.
- Topology:
  - Directed graph between subroutines and main program.
Style: Main program & subroutine

Additional constraints:
- None.

Qualities:
- Modularity, as long as interfaces are maintained.

Typical uses:
- Small programs.

Cautions:
- Poor scalability. Data structures are ill-defined.

Relations to languages and environments:
- BASIC, Pascal, or C.
Style: Object-oriented
Style: Object-oriented

- Encapsulation of state and actions.

- Components:
  - Objects or ADTs.

- Connections:
  - Method calls.

- Data elements:
  - Method arguments.

- Topology:
  - Varies. Data shared through calls and inheritance.
Style: Object-oriented

- Additional constraints:
  - Commonly used with shared memory (pointers). Object preserves identity of representation.

- Qualities:

- Typical uses:
  - With complex, dynamic data. Correlation to real-world entities.

- Cautions:
  - Distributed applications hard. Often inefficient for sci. computing. Potential for high coupling via constructors. Understanding can be difficult.

- Relations to languages and environments:
  - C++, Java.
Dataflow

- A data flow system is one in which:
  - The availability of data controls computation.
  - The structure of the design is determined by the orderly motion of data between components.
  - The pattern of data flow is explicit.
- Variations:
  - Push vs. pull.
  - Degree of concurrency.
  - Topology.

[CZARNECKI]
Style: Batch-sequential
Style: Batch-sequential

- Separate programs executed in order passed, each step proceeding after the previous finishes.

- Components:
  - Independent programs.

- Connections:
  - Sneaker-net.

- Data elements:
  - Explicit output of complete program from preceding step.

- Topology:
  - Linear.
Style: Batch-sequential

- Additional constraints:
  - One program runs at a time (to completion).

- Qualities:
  - Interruptible execution.

- Typical uses:
  - Transaction processing in financial systems.

- Cautions:
  - Programs cannot easily feed back in to one another.
Style: Pipe-and-filter
Style: Pipe-and-filter

- Streams of data are passed concurrently from one program to another.

- Components:
  - Independent programs (called filters).

- Connections:
  - Explicitly routed by OS.

- Data elements:
  - Linear data streams, often text.

- Topology:
  - Typically pipeline.
Style: Pipe-and-filter

- Qualities:
  - Filters are independent and can be composed in novel sequences.

- Typical uses:
  - Very common in OS utilities.

- Cautions:
  - Not optimal for interactive programs or for complex data structures.
Style: Blackboard
Style: Blackboard

- Independent programs communicate exclusively through shared global data repository.

- Components:
  - Independent programs (knowledge sources), blackboard.

- Connections:
  - Varies: memory reference, procedure call, DB query.

- Data elements:
  - Data stored on blackboard.

- Topology:
  - Star; knowledge sources surround blackboard.
Style: Blackboard

‣ Variants:
  ‣ Pull: clients check for blackboard updates.
  ‣ Push: blackboard notifies clients of updates.

‣ Qualities:
  ‣ Efficient sharing of large amounts of data. Strategies to complex problems do not need to be pre-planned.

‣ Typical uses:
  ‣ Heuristic problem solving.

‣ Cautions:
  ‣ Not optimal if regulation of data is needed or the data frequently changes and must be updated on all clients.