Software Architecture

- “Good software architecture makes the rest of the project easy.” —Steve McConnell, Survival Guide

- “There are two ways of constructing a software design: one way is to make it so simple that there are obviously no deficiencies; the other is to make it so complicated that there are no obvious deficiencies.” —C.A.R. Hoare (1985)

The basic problem

Requirements

How do you bridge the gap between requirements and code?

Code

One answer

Requirements

A miracle happens!

Code
An architecture is expressed by components and connectors

- **Components** define the basic computations comprising the system, and their behaviors
  - abstract data types, filters, etc.
- **Connectors** define the interconnections between components
  - procedure call, event announcement, asynchronous message sends, etc.
- The line between them may be fuzzy at times
  - Ex: A connector might (de)serialize data, but can it perform other, richer computations?

A good architecture

- Satisfies functional and performance requirements
- Manages complexity
- Accommodates future change
- Also: reliability, safety, understandability, compatibility, robustness, …
Divide and conquer

- Benefits of decomposition:
  - Decrease size of tasks
  - Support independent testing and analysis
  - Separate work assignments
  - Ease understanding
- Use of abstraction leads to modularity
  - Implementation techniques: information hiding, interfaces
- To achieve modularity, you need:
  - Strong cohesion within a component
  - Loose coupling between components
  - And these properties should be true at each level

Qualities of modular software

- decomposable
  - can be broken down into pieces
- composable
  - pieces are useful and can be combined
- understandable
  - one piece can be examined in isolation
- has continuity
  - reqs. change affects few modules
- protected / safe
  - an error affects few other modules

Interface and implementation

- public interface: data and behavior of the object that can be seen and executed externally by "client" code
- private implementation: internal data and methods in the object, used to help implement the public interface, but cannot be directly accessed
- client: code that uses your class/subsystem
- Example: radio
  - public interface is the speaker, volume buttons, station dial
  - private implementation is the guts of the radio; the transistors, capacitors, voltage readings, frequencies, etc. that user should not see

Loose coupling

- Coupling assesses the kind and quantity of interconnections among modules
- Modules that are loosely coupled (or uncoupled) are better than those that are tightly coupled
- The more tightly coupled are two modules, the harder it is to work with them separately, and thus the benefits become more limited
  - Think about testing … can you test two modules independently, or do you need to test them both together?
Tightly or loosely coupled?

- Ignore those "End" markers.
- What would happen if we removed one component?
- How many modules would be affected?

Strong cohesion

- Cohesion refers to how closely the operations in a module are related.
- Tight relationships improve clarity and understanding.
- Classes with good abstraction usually have strong cohesion.
- No schizophrenic classes!

Strong or weak cohesion?

```cpp
class Employee {
public:
    // Other methods...
    bool IsJobClassificationValid(JobClassification jobClass);
    bool IsZipCodeValid(Address address);
    bool IsPhoneNumberValid(PhoneNumber phoneNumber);
    // Other methods...
}
```

- Any logical connection between employees and routines that check zip codes, etc.?
- SQL details at the same layer of abstraction?
An architecture helps with

- **System understanding** – describes the interactions between modules
- **Reuse** – by defining the high level components, we can see if there is opportunity for reuse
- **Construction** – breaks development down into work items; provides a path from requirements to code
- **Evolution** – see reuse
- **Management** – helps understand the work to do, and the work done; track progress
- **Communication** – gives you a vocabulary; pictures say 1000 words

Architectural style

- Defines the vocabulary of components and connectors for a family (style)
- Constraints on the elements and their combination
  - Topological constraints (no cycles, register/announce relationships, etc.)
  - Execution constraints (timing, etc.)
- By choosing a style, one gets all the known properties of that style (for any architecture in that style)
  - Ex: performance, lack of deadlock, ease of making particular classes of changes, etc.

Not just boxes and arrows

- Consider pipes & filters, for example (Garlan and Shaw)
  - Pipes must compute local transformations
  - Filters must not share state with other filters
  - There must be no cycles
- If these constraints are not satisfied, it’s not a pipe & filter system
  - One can’t tell this from a picture
  - One can formalize these constraints

Pipe and filter

Pipe – passes the data

Filter – computes on the data

Each stage of the pipeline acts independently of the others
**Model-View-Controller**

Separates the application object (model) from the way it is represented to the user (view) from the way in which the user controls it (controller).

**Blackboard architectures**

- The knowledge sources: separate, independent units of application dependent knowledge. No direct interaction among knowledge sources
- The blackboard data structure: problem-solving state data. Knowledge sources make changes to the blackboard that lead incrementally to a solution to the problem.
- Control: driven entirely by state of blackboard. Knowledge sources respond opportunistically to changes in the blackboard.

Blackboard systems have traditionally been used for applications requiring complex interpretations of signal processing, such as speech and pattern recognition.

**The design and the reality**

- The code is often less clean than the design
- The design is still useful
  - Communication among team members
  - Selected deviations can be explained more concisely and with clearer reasoning

**Hearsay-II: blackboard**

![Hearsay-II Instance of Blackboard](image)
Questions?