Architectural Decomposition & Views
(based on materials from Mei Nagappan, Taylor et al., 2010)

CS 446/646/EC 452 – SOFTWARE DESIGN AND ARCHITECTURE
Software Architecture? – A deeper dive!

“The set of principal design decisions made about the software system.”
(Taylor et al., 2010)

Decisions related to system?
1) Structure (e.g., components should be organized and composed like …)
2) Behavior (e.g., data processing, visualization will be performed in strict sequence).
3) Communications (interactions among components will use only event notification).
4) Non-functional requirements.
5) Implementation.
Examples

Ordered Sequence of Actions

(2) Course notes posted

Course notes downloaded

Hierarchy of Objects

(1) Student

Graduate

Undergraduate

Opeyemi Adesina - Software Design and Architecture - Spring 2018
Key aspects of software architecture

- Software Components.
- Software Connectors.
- Software Configurations.
Software Components? – A deeper dive!

“Elements that **encapsulates processing and data** at an architectural level.”

(Taylor et. al, 2010)

Definition:

- An architectural entity that:
  - encapsulates a subset of functionality (e.g., `getAge()`);
  - restricts access via explicit interface (e.g., `private`); and
  - has explicit environmental dependencies (e.g., runs on OS).

```java
class Student {
    private int age;
    public int getAge();
}
```
Software Connectors? – A deeper dive!

**Definition:**
An architectural entity that are **tasked with effecting and regulating interactions** between components.

- the underlying protocols for communication.

**Examples**
- Inter – Node (via **RPC**, **Web services**, etc.).
- Intra – Node (method call, etc.).

(Zakariapour, 2018)
**Definition:** A set of specific associations between components and connectors of a software system’s architecture.

- This is also known as software topology.

(Zakariapour, 2018)
Topological Goals

- Minimize **coupling** between components.
  - The less components know about each other, the better (also known as information hiding).

- Maximize **cohesion** within each component.
  - Components should be **responsible for a logical service**; extraneous functionality should be eliminated.

(Easterbrook, 2012)
Topological Goals – How to?

- **Abstraction**
  - Complex problems can be solved by **abstracting away unnecessary details**.
  - **Focusing on key issues** while eluding extraneous detail (some of which may be important during more detailed design activities).
  - Two key methods: **control abstraction** (e.g., structured programming) and **data abstraction** (e.g., abstract data types).
Topological Goals – How to?

- **Decomposition (a.k.a. top-down abstraction)**
  - Complex problems can be broken into independent components.
  - **Describe each component.**
  - Criteria for decomposition:
    - Implementation teams;
    - Application domains; and
    - Parallelization.
  - Simplify typical cases and make them exceptional as much as possible.
"Organizations which design systems ... are constrained to produce designs which are copies of the communication structures of these organizations."

(Conway, 1967)
Facts about decomposition (Easterbrook, 2001)

- **Decomposition can work well:**
  - E.g. designing a restaurant menu
  - Choose style and theme
  - Design appetizers menu
  - Design entrees menu
  - Design desserts menu
  - Design drinks menu
  - Assemble and edit

- **Decomposition doesn’t always work:**
  - E.g. writing a play:
  - Choose a set of character parts
  - Write character 1’s part
  - Write character 2’s part
  - Write character 3’s part
  - Merge
  - ... etc ...

- **Decomposition isn’t always possible:**
  - For very complex problems (e.g. Managing the economy)
  - For impossible problems (e.g. Turning water into wine)
  - For atomic problems (e.g. Adding 1 and 1)
Facilitates communication between stakeholders.

Opaque architectures are useless — no one understands it.

Properties of representations:
- **Ambiguity**: open to more than one interpretations.
- **Accuracy**: correct within tolerances.
- **Precision**: consistent but not necessarily correct.
Architectural Views

- Architectural models can be overwhelming.
  - Different views focus on specific subsets of elements or relationships.
  - Views often focus on specific concerns or scenarios within a system.

- Views overlap; maintaining consistency between views is challenging.
- Architectural views (or representation) are means of communication.
  - Unified language is necessary (e.g., Unified Modeling Language - UML).
Architectural Views (contd.) - UML

- **UML diagram** – graphical view of a model of a system under design, implementation, or already in existence.
  - It contains graphical elements – Nodes and Edges.

- UML specification defines two broad categories of diagrams:
  - Structure diagrams.
  - Behavior diagrams.
Behavioral aspect (or view).

Use case diagrams are used to specify:
- what the system is supposed to do - requirements;
- what the system can do – functionality; and
- how environment interact with subjects (e.g., subsystem, component).
Structural aspect.

Class diagrams are used to specify:

- Classes and Interfaces;
  - their properties (e.g., attributes);
  - constraints (e.g., multiplicities, visibility, etc.); and
  - relationships (e.g., associations, generalizations, etc.).
Behavioral aspect.

State diagrams are used to specify:

- Discrete behavior of a part of designed system;
- Usage protocol of part of a system.

Kinds of state machines:

- Behavioral state machine, and
- Protocol state machine.
Behavioral aspect.

Sequence diagrams are used to specify:
- Message interchange between lifelines (or participating elements).
UML – Deployment Diagram

- Structural aspect.
- Deployment diagrams are used to specify:
  - System architecture as deployment of software artifacts (i.e., physical components resulting from a development process).

- Common types:
  - implementation of components by artifacts;
  - specification level deployment diagram;
  - instance level deployment diagram; and
  - network architecture of the system.
Kinds of architectures

- Prescriptive architecture.
  - dictates how the system **will be built** (i.e., as conceived).

- Descriptive architecture.
  - captures how the system **was built** (i.e., as implemented).
Architectural degradation

- **Architectural drift.**
  - when changes that are not captured in *the current architecture do not violate it.*

- **Architectural erosion.**
  - when changes to the architecture *violate current architecture.*
Managing degradation

» Architectural recovery key elements

» Conceptual architecture:
  » How developers think about the system.
  » Focuses on meaningful relationships.

» Concrete architecture:
  » How the system was actually built.
  » Necessary: the devil is in the details.
Managing degradation - Tool Support

- **Umple** - Model-oriented programming language:
  - Background article – [Merging modeling and programming using Umple](https://example.com).
  - UML diagrams facilitated – Class, state machine and composite structure diagrams.
  - Code generation – Java, Real-time C++, Ruby, nuXmv, etc.
  - Usage - Command-line, [web application](https://example.com) and Eclipse plugin.
  - ...
Tool Support

UMLet 14.2
Free UML Tool for Fast UML Diagrams

Follow @UMLet