Architecture and Design Patterns

Focus on What is Being Designed and Built

OO Has Goal of Design and Software Re-Use
  – Encapsulation of Data and Operations
  – Class Hierarchy and Object Instances
  – Well Understand Use Cases
  – Well Understand Interaction Between Objects

Study 4 Key System Types

1. Interactive – E.g. GUI, CLI
2. Event Driven – E.g. Anit-lock Breaking System Software
4. Transaction Oriented – E.g. DBMS
Four Common Types of Systems

(a) Interactive subsystem

(b) Event-driven subsystem

(c) Transformational subsystem

(d) Database subsystem
Architectural Design

- Transition from Requirements, Supports Requirements Analysis, Supports Design Activity
- Focus on Domain Analysis, Use Cases and Higher Levels of System Rather than Details

- UML
  - Use Cases
  - Interaction Diagrams
  - Class Diagrams
  - Basic Class Hierarchy (Draft of Class Definitions)

- SA/SD – Dataflow, ER/EER Diagrams, High Level State Machines (Flowcharts are Typically Too Detailed)
Traditional SA/SD – Useful, But Not OO

- **Data Flow Diagrams** – Data [Messages] Between Processes and is Transformed

- **Entity Relationship Diagrams** – Lacks Operations, but Defines Entities [Objects] and Relationships

- **State Machines** [in Common, but Typically for Each Process in DFD]

- **Flow-Charts** – Detailed Procedural Design [Interaction, Logic]
Domain Models with UML - CASE

- UML is Universal Modeling Language
- Use to Support Requirements Analysis

Start Here!

https://www.modelio.org/

http://argouml.tigris.org/

Helpful Validation and Verification Features for Design

- Integrated Models
- Checklists – Completeness
- CPP and Java Code Generation
Tool-Based Activities

- Bring Up Modelio and Start Entering ATM Design – Use Case and Class Diagram, Compare to UML Reference

- Versions of UML – 2.4.1 Current

- Minute Paper #2 – Do Design Tools Really Assist with Software Quality Assurance?
Key Takeaway Points

• Domain modeling is a conceptualization process to help the development team understand the application domain.

• Five easy steps: collecting information about the application domain; brainstorming; classifying brainstorming results; visualizing the domain model using a UML class diagram; and performing inspection and review.
Domain Modeling in the Methodology Context

(a) Planning Phase
- Business goals & needs
- Current situation
  → Acquiring Requirements (Domain Modeling)

- Preliminary requirements
  → Deriving Use Cases from Requirements

- Abstract & high level use cases, use case diagrams
  → Allocating Use Cases & Subsystems to Iterations

- Use case-iteration allocation matrix
  → Producing an Architecture Design

(b) Iterative Phase – activities during each iteration

- Customer feedback
  → Accommodating Requirements Change

- Iteration use cases
  → Domain Modeling

- Domain model
  → Actor-System Interaction Modeling

- Expanded use cases & UI design
  → Behavior Modeling & Responsibility Assignment

- Behavior models
  → Deriving Design Class Diagram

- Design class diagram
  → Test Driven Development, Integration, & Deployment

- Domain model

- Test Driven Development, Integration, & Deployment

control flow

data flow

control flow & data flow
Architectural Design Process

- Determine Design Objectives
- Determine Type of System
  - Types of System & Characteristics
- Determine Design Objectives
- Apply an Architectural Style
  - [architectural style available]
  - [architectural style not available]
- Specify Subsystem Functions, Interfaces & Interaction Behavior
- Perform Custom Architectural Design
- Partial Design
- Review the Architectural Design
- Feedback
Assignment #2 – Domain Models

Learning Objectives – Value of UML for Requirements Analysis

– Completeness
– Design Walk-throughs
– Validation [Are We Building the Right Thing?]
– Verification [Are We Building it Right?]

New Design, Client-Server [Cloud] Architectural Pattern

Storage-as-a-Service

– The Competition, E.g. Drop-Box, Amazon S3, Google Cloud Storage
– The Concept – Archive for Unstructured Files (Photos, Documents, Bits…), Not Code, Not DBMS
The Requirements – Capabilities Focus

1. Store Any Number of Files (name space) Up to N Gbytes in an Archive, Browse on Web or Windows, Mac, Linux File Explorer

2. Protect with RAID Against Single Erasure (Covered in CS317, SE420)

3. Submit and Retrieve Any File by Name, Time and Date Archived (In Case of Duplicate Names)
Assignment #2 Goals

- Consistent and Complete UML Design – Domain Model Focus - Ready for Validation and Verification Walkthrough

- Ideally Capable of C++ or Java Class Code Generation

- We Will Walk-through Design for Assignment #3

- More In Depth Use of Modelio or ArgoUML

- Better Understanding of UML 2.4 – Use Case, Component, Class, Interaction Diagrams [2 From Behavior Side, 2 From Structure Side of UML]
Domain Modeling Steps

1. Collecting application domain information
2. Brainstorming
3. Classifying brainstorming result
4. Visualizing the domain model
5. Reviewing the domain model.
Tip for Domain Modeling

Do not do brainstorming and drawing at the same time. The result could be very poor.

1) Team brainstorming: List the concepts, and then classify them on a whiteboard.

2) Take a picture(s) of the whiteboard using a digital camera.

3) Email the digital images to team members.

4) Have a member or two to convert the result to a UML class diagram.

5) Email the UML class diagram to all members to review.

6) Modify the diagram to reflect corrections and comments.
RAID Archive Systems - Multiple Disk Drives

- Disk Drives Fail – Like a Light-bulb
  - MTBF of 100’s of Thousands of Hours [3 to 5 Years at Duty Cycle]
  - Difficult to Determine When Failure Might Occur
  - The Larger the Population, the More Often Failures will be Seen

- Disk Drives Have Low Random Access [100 to 200 I/Os per Second]

- Idea – Write to them in Parallel and Mirror Data to Protect Against HDD Failures (Erasures)
RAID-10

A1, A2, A3, … A12

RAID-0 Striping Over RAID-1 Mirrors

RAID-1 Mirror

A1
A4
A7
A10

A1
A4
A7
A10

A2
A5
A8
A11

A2
A5
A8
A11

A3
A6
A9
A12

A3
A6
A9
A12

© Sam Siewert
RAID Operates on LBAs/Sectors (Sometimes Files)

- SAN/DAS RAID
- NAS – Filesystem on top of RAID
- RAID-10, RAID-50, RAID-60
  - Stripe Over Mirror Sets
  - Stripe Over RAID-5 XOR Parity Sets
  - Stripe Over RAID-6 Reed-Soloman or Double-Parity Encoded Sets
- EVEN/ODD
- Row Diagonal Parity
- Minimum Density Codes (Liberation)
- Reed-Solomon Codes
RAID5,6 XOR Parity Encoding

- MDS Encoding, Can Achieve High Storage Efficiency with N+1: N/(N+1) and N+2: N/(N+2)

For example:

Can tolerate any single erasure.

Storage Efficiency vs. Number of Data Devices for 1 XOR or 2 P,Q Encoded Devices

© Sam Siewert
RAID-50

A1,B1,C1,D1,A2,B2,C2,D2,E1,F1,G1,H1,…, Q2,R2,S2,T2

RAID-0 Striping Over RAID-5 Sets
RAID is an Erasure Code

- RAID-1 is an MDS EC (James Plank, U. of Tennessee)

![Diagram of RAID-1]

- MDS
- Extremely fast encoding/decoding/update.
- Rate: $R = 1/(m+1)$ - Very space inefficient
- There are many replication/based systems (*P2P especially*)

© Sam Siewert
Read, Modify Write Penalty

Any Update that is Less than the Full RAID5 or RAID6 Set, Requires
1. Read Old Data and Parity – 2 Reads
2. Compute New Parity (From Old & New Data)
3. Write New Parity and New Data – 2 Writes

Only Way to Remove Penalty is a Write-Back Cache to Coalesce Updates and Perform Full-Set Writes Always

Write A1

RAID-5 Set

P(ABCD)_{new} = A1_{new} \ xor A1 \ xor P(ABCD)

<table>
<thead>
<tr>
<th>A1</th>
<th>B1</th>
<th>C1</th>
<th>D1</th>
<th>P(ABCD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

...
What Is a Model?

• A conceptual representation of something.
• A schematic description of a system, theory, or phenomenon that accounts for its known or inferred properties and may be used for further study of its characteristics. (Dictionary Definition)

We perceive the world differently due to differences in backgrounds and viewpoints. Modeling facilitate collective understand of the application.
Represent Domain Model as UML Class Diagram

• UML class diagram is a structural diagram.
• It shows the classes, their attributes and operations, and relationships between the classes.
• Domain model is represented by a class diagram without showing the operations.
Representing Type in UML

**Employee**

- **SSN**: char*
- **name**: char*

**checkin**(time:float):void

**checkout**(time:float):void

**general syntax**

name : type
A staff is a user, a student is a user. Features (i.e., attributes and operations) defined for the super-class are automatically defined for the subclasses.
Two Tests for Inheritance

- IS-A test: every instance of a subclass is also an instance of the superclass.
- Conformance test: relationships of a superclass are also relationships of subclasses.

```
<table>
<thead>
<tr>
<th>Engine</th>
</tr>
</thead>
<tbody>
<tr>
<td>model#</td>
</tr>
<tr>
<td>horse power</td>
</tr>
<tr>
<td>manufacturer name</td>
</tr>
<tr>
<td>start()</td>
</tr>
<tr>
<td>stop()</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Car</th>
</tr>
</thead>
<tbody>
<tr>
<td>drive()</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Professor</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
</tr>
<tr>
<td>phone</td>
</tr>
<tr>
<td>teach(...)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Visiting Professor</th>
</tr>
</thead>
<tbody>
<tr>
<td>signContract()</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Retirement Plan</th>
</tr>
</thead>
<tbody>
<tr>
<td>enroll</td>
</tr>
</tbody>
</table>

Visiting professors cannot enroll in a retirement plan at the host university.

Car is not an engine.
Aggregation Relationship

- It expresses the fact that one object is part of another object.
- Example: engine is part of a car.
- It is also called part-of relationship.
Association Relationship

- It expresses a general relationship other than inheritance and aggregation.
- These can be application specific relationships between two concepts.
- Example: "instructor teach course," "user has account."

```
Professor
name
phone
teach(...)  enroll  Retirement Plan
```

Enroll is not an inheritance or aggregation relationship.
Role and Association Direction

Student

enroll

Course

Student enrolls in Course

Student

association name

Course

association direction

Faculty teaches Course

Faculty

teach

role name

Copyright © 2014 by the McGraw-Hill Companies, Inc. All rights Reserved.
Role and Multiplicity

Another employee is the worker.

Employee supervises other employees.

An employee is the supervisor.

A supervisor supervises zero or more employees.

An employee is the worker.
Key Takeaway Points

• The software architecture of a system or subsystem refers to the style of design of the structure of the system including the interfacing and interaction among its subsystems and components.

• Different types of systems require different design methods and architectural styles.
Architectural Design in Methodology Context

(a) Planning Phase
- Business goals & needs
- Current situation
- Acquiring Requirements
  - Preliminary requirements
  - Deriving Use Cases from Requirements
  - Abstract & high level use cases, use case diagrams
  - Allocating Use Cases & Subsystems to Iterations
  - Use case-iteration allocation matrix
  - Producing an Architectural Design

(b) Iterative Phase – activities during each iteration
- Use case-iteration allocation matrix
- Customer feedback
- Allocating Use Cases & Subsystems to Iterations
- Domain Modeling
- Actor-System Interaction Modeling
  - Expanded use cases & UI design
- Behavior Modeling & Responsibility Assignment
  - Behavior models
- Deriving Design Class Diagram
  - Design class diagram
- Test Driven Development, Integration, & Deployment

Copyright (c) 2014 by the McGraw-Hill Companies, Inc. All rights Reserved.
What Is Software Architectural Design

• The *software architecture* of a system or subsystem refers to the style of design of the structure of the system including the interfacing and interaction among its major components.

• Software architectural design is a decision-making process to determine the software architecture for the system under development.
Characteristics of Interactive Systems

- The interaction between system and actor consists of a relatively fixed sequence of actor requests and system responses.
- The system has to process and respond to each request.
- Often, the system interacts with only one actor during the process of a use case.
- The actor is often a human being although it can also be a device or another subsystem.
- The interaction begins and ends with the actor.
- The actor and the system exhibit a “client-server” relationship.
- System state reflects the progress of the business process represented by the use case.
Characteristics of Event-Driven Systems

• It receives events from, and controls external entities.
• It does not have a fixed sequence of incoming requests; requests arrive at the system randomly.
• It does not need to respond to every incoming event. Its response is state dependent—the same event may result in different responses depending on system state.
• It interacts with more than one external entity at the same time.
• External entities are often hardware devices or software components rather than human beings.
• Its state may not reflect the progress of a computation.
• It may need to meet timing constraints, temporal constraints, and timed temporal constraints.
Characteristics of Transformational Systems

• Transformational systems consist of a network of information-processing activities, transforming activity input to activity output.

• Activities may involve control flows that exhibit sequencing, conditional branching, parallel threads, synchronous and asynchronous behavior.

• During the transformation of the input into the output, there is little or no interaction between system and actor—it is a batch process.

• Transformational systems are usually stateless.

• Transformational systems may perform number crunching or computation intensive algorithms.

• The actors can be human beings, devices, or other systems.
Characteristics of Object-Persistence Systems

• It provides object storage and retrieval capabilities to other subsystems.
• It hides the implementation from the rest of the system.
• It is responsible only for storing and retrieving objects, and does little or no business processing except performance considerations.
• It is capable of efficient storage, retrieval, and updating of a huge amount of structured and complex data.