Software Engineering - CSC4350/6350 - Rao Casturi
System Design – System Decomposition

- Migration from Analysis to System Design.
- Define *Design Goals*
- Design Initial *Sub System* Decomposition.
- *Refine* the Sub System to address the Design Goals.
Activities of System Design

This phase will produce the following

1. Design Goals
2. Software Architecture
3. Boundary Use Cases, Exceptions, hardware configurations
Design Goal:

- Design Goals come from Non-Functional Requirements.
- Trade Off decisions made.
- Sub System Decomposition is the bulk of the System Design.
- Developers divide the system into manageable parts to reduce complexity.
- Each Sub-System is assigned to a smaller teams.
<table>
<thead>
<tr>
<th>Item</th>
<th>Architectural Concept</th>
<th>SW Engineering Concept</th>
</tr>
</thead>
<tbody>
<tr>
<td>Components</td>
<td>Room</td>
<td>Subsystem</td>
</tr>
<tr>
<td>Interfaces</td>
<td>Doors</td>
<td>Services</td>
</tr>
<tr>
<td>Non-Functional Requirements</td>
<td>Living Area</td>
<td>Response time</td>
</tr>
<tr>
<td>Functional Requirements</td>
<td>Residential House</td>
<td>Use Cases</td>
</tr>
<tr>
<td>Costly Rework</td>
<td>Moving Walls</td>
<td>Change Subsystem Interfaces</td>
</tr>
</tbody>
</table>
Design Goals Continued

- Finalize the strategies for
  - Hardware / Software Strategy
  - Persistent Data Management
  - Control Flow
  - Access Control Policy
  - Handling Boundary Conditions

Source: OOSE-Bernd Bruegge & Allen H. Dutoit
What is a Subsystem?

• A subsystem is a replaceable part of the system with well-defined interfaces that encapsulates the state and behavior of its contained classes.

• To reduce the complexity of the Solution Domain, a System is divided into smaller systems called Subsystem.

• Subsystems are also Called Categories
More about Subsystems....

• **Service**: Is a related operations that share a common purpose.

• Define the Subsystem in terms of the **services** they provide.

• During this phase we **define the Services** and in the Object Design phase we **develop the operations** it will provide.
Services and Subsystem Interfaces

Subsystem Interface:

• A set of operations of a subsystem that are available for other subsystems.

• Subsystem Interface includes
  – Name of the operation
  – Pass through Parameters and data type
  – Return values
  – High level behavior

• Object Design focus on the API (Application Programmer Interface)

• Minimize the information provided about the implementation.
  – Should not reference about the internal data structure.
Services and Subsystem Interfaces

• **Coupling**
  – Number of dependencies between two systems.
  – Loosely coupled – relatively independent.
  – Impact analysis.
  – Assembly connectors or Ball-and-Socket Connector

• **Cohesion**
  – Number of dependencies within a subsystem.
  – Unrelated objects – low cohesion.
Coupling Example

Student Information Display

Class Registration

Graduation Tracker

Student Database
Modified Design to show loosely coupled systems

- Student Information Display
- Class Registration
- Graduation Tracker
- Request Translator
- Student Database
Cohesion Example

DecisionSubsystem

Criterion

Assesses

Option

DesignProblem

SolvableBy

SubTask

ResolvedBy

Decision

ImplementedBy

ActionItem

Task

Subtasks

Source: OOSE-Bernd Bruegge & Allen H. Dutoit
Layers and Partitions
Layers and Partitions

• A **hierarchical decomposition** of a system yields an ordered set of layers.

• A **layer** is a grouping of subsystems providing related services, possibly realized using services from another layer.

• **Closed Layer Architecture** – Each Layer can access only one layer immediately below.

Source: OOSE-Bernd Bruegge & Allen H. Dutoit
Example of Closed Layer

Example: the OSI (Open System Interconnection) closed architecture model.

1. The Physical layer - is the hardware interface with the network.
2. The DataLink layer – responsible for transmitting data frame using the services from the Physical layer.
3. The Network – transmitting and routing packages within the network.
4. The Transport – responsible for reliably transmitting data from end to end. This is the layer seen by UNIX programmers/users when transmitting data over TCP/IP (Transmission Connection Protocol/Internet Protocol) sockets between processes.
5. The Session – responsible for the initialization of a connection.
6. The Presentation – performs data transformation such as byte swapping or encryption.
7. The Application – the layer that is designed which may also consist of subsystem layers itself.

Source: OOSE-Bernd Bruegge & Allen H. Dutoit
Example of Open Layer

- The lowest layer is provided by the operating system or by a windowing system, such as X11, and provides basic window management.
- AWT is an abstract window interface provided by Java to shield applications from specific window platforms.
- Swing is a library of user interface objects that provides a wide range of facilities, from buttons to geometry management.
- An Application usually accesses only the Swing interface. However, the Application layer may bypass the Swing layer and directly access AWT.
- Openness of the architecture allows developers to bypass the higher layers to address performance bottlenecks.

Source: OOSE-Bernd Bruegge & Allen H. Dutoit
Goal:
To Identify how the Categories interact or relate to implement a Use Case

• **Category Interaction Diagram (CID)**
  
  – A graphical representation of the interactions between Categories required to satisfy, or implement a Use Case using the notation of CASE Tool
  
  – UML world Interaction Diagram is any diagram that depicts interaction at the Instance Level, and can be either a Sequence Diagram or a Collaboration Diagram

• **UML Interaction Diagram:** A UML Interaction Diagram refers to both UML Sequence Diagram and UML Collaboration Diagram. They represent general behavior, specifically interactions between objects

• **UML Sequence Diagram:**
  
  – Shows the collaboration required between objects (instances)

Work Product: Produce a CID
Steps to create a CID

Step 1: Establish Naming convention
- CID will be appended to the USE CASE Name
- UC16_Operator_Updates_Notional_Values_In_DB_CID
- For Category Sequence Diagram it will be CSD

Step 2: Agree upon a Representation Mechanism (to produce)
- OMT Event Traces
- Jacobson Interaction Diagrams
- Booch Object Scenario Diagrams
- UML Sequence Diagram
- UML Collaboration Diagrams
Steps to create a CID

Step 3: Agree upon Diagram Contents
- Include the Title of the Diagram
- Textual overview of the Use Case
- Exceptions
- GUI Classes

Step 4: Number each Message
- Each Message on a CID needs a sequence Number as a part of the message or as a comment attached to the message

Step 5: Comment each CID

Step 6: Agree to Notation for Use Cases Using/Referencing other Use Cases
- Unique Name ($Use_Case)
- Use the Name of the Use Case as the name of the Message to the category
UC_16_Operator_Updates_Notional_Values_in_DB_ID

Steps in the Use Case

1. Operator selects “Update all” button
2. Display the Update_All_View
3. Disable the Alarm
4. Operator enters values
5. If the OK button is selected then:
   6. Start a transaction:
      7-12 steps Enter values
   13. Commit the transaction and
   14. Enable the alarm and
   15. Destroy the Update_All_View;
   16. Else if the Cancel button selected, then
      17. Enable the alarm and
      18. Destroy the Update_All_View:
1. Operator selects "Update all" button.
2. Display the Update_All_View.
3. Disable the Alarm.
4. Operator enters values.
5. If the OK button is selected, then;
6. start a transaction;
7-12. enter values into the Database;
13. commit the transaction and
14. enable the alarm and
15. Destroy the Update_All_View;
16. else
if the Cancel button is selected, then
17. Enable the alarm and
18. Destroy the Update_All_View.

*NOTE: The parent Database Class is shown on this ID, but at the time the IDs are drawn it may not be known what database implementation (e.g., which Subclass) will be used. In the solution, the SNAP implementation uses the MSAccess_Database Class; while the C++ implementation uses the Flat_File_Database Class.

Figure 6-6. Interaction Diagram: UC16_Operator_Updates_Nominal_Values_In_DB

Source: Use Cases Combined with Booch OMT UML Process and Products by Putnam and Charles
Architecture Styles
Repository Architectural Style:

• In the repository architectural style, subsystems access and modify a single data structure called the central repository.

• Subsystems are relatively independent and interact only through the repository. Control flow can be dictated either by the central repository or by subsystems.

• Example – Database triggers on the data invoke peripheral systems or by the subsystems.
Repository Architectural Style

Source: OOSE-Bernd Bruegge & Allen H. Dutoit
### Model View Controller

Classified into 3 different types:

1. **Model subsystems** – responsible for maintaining domain knowledge.

2. **View subsystems** – responsible for display to user.

3. **Controller subsystems** – responsible for managing the sequence of interactions with user.

This is also a type of repository architecture. It is used mostly in interactive systems, especially when multiple views of the same model are needed.
Model – View – Controller Design Pattern

• **Model** holds the domain knowledge (Business Rules).
• **View** holds the User input and display model.
• **Controller** holds the sequencing of the user inputs and acts as intermediate between Model and View.
Model – View – Controller Design Pattern

- Smart - Thin- DUMMY approach
- Initiator, Subscriber & Notifier
- Observer Design Pattern
Client Server Architecture

- In the **client/server architectural style** a subsystem, the **server**, provides services to instances of other subsystems called the **clients**, which are responsible for interacting with the user.
- The request for a service is usually done via a remote procedure call mechanism or a common object.
- Control flow in the clients and the servers is independent except for synchronization to manage requests or to receive results.

Source: OOSE-Bernd Bruegge & Allen H. Dutoit
Peer-to-Peer Architecture Style

- A **peer-to-peer architectural style** is a generalization of the client/server architectural style in which subsystems can act both as client or as servers, in the sense that each subsystem can request and provide services.
- The control flow within each subsystem is independent from the others except for synchronizations on requests.

![Diagram of peer-to-peer architecture](image_url)
Three-Tier & Four Tier Architecture

In this style the subsystems are organized into three layers

- The **interface layer** includes all boundary objects which deal with end users. Example windows, forms, web pages etc.
- The **application logic layer** includes all control and entity objects, realizing the processing, rule checking, and notification required by the application.
- The **storage layer** realizes the storage, retrieval, and query of persistent objects.

Source: OOSE-Bernd Bruegge & Allen H. Dutoit
Pipe and filter Architecture

- **Pipe and filter architectural style** the subsystems process data received from a set of inputs and send results to other subsystems via a set of outputs.
- The subsystems are called “filters,” and the associations between the subsystems are called “pipes.”
- Each filter knows only the content and the format of the data received on the input pipes, not the filters that...
System Design Activities: Addressing Design Goals
Addressing Design Goals

1. Mapping Subsystems to Processors and Components
2. Identifying and Storing Persistent Data
3. Providing Access Control
4. Designing the Global Control Flow
5. Identifying Services
6. Identifying Boundary Conditions
7. Reviewing the System Design Model
Addressing Design Goals

Source: OOSE-Bernd Bruegge & Allen H. Dutoit
1. Mapping Subsystem to Processes and Components

- **UML Deployment Diagrams**
- DDs are used to show the relationship among run-time components and nodes.
- **Components**: Self-contained entities that provide services to other components. (Eg: Web browsers, Web Servers etc)
- **Node**: Is a physical device or an execution environment in which components are executed.
- **System**: Composed of interacting run-time components that can be distributed among several nodes. A node can contain another node.
1. Mapping Subsystem to Processes and Components

• Many Systems run on more than one computer.
• High-Performance needs
• Multiple uses
• Selection of virtual machine
• Hardware mapping activity has significant impact on the performance and complexity of the system.
• Suggested approach is to perform it early in system design.
2. Persistent Data Store

• **Flat Files:** Storage abstractions provided by operating system. Data stored as a sequence of bytes. This is relatively low level and could enhance speed. However, data can be corrupted or/and lost easily. Should be used for very small applications that may not grow in size.

• **Relational Database:** Data stored in tables – each column is an attribute and each row represents a data item as a tuple of attribute values.

• **OO database:** Provide similar services to relational databases, but stores data as objects and associations. It also provide inheritance and abstract datatypes. However, it is slower than relational database.
When to use flat files, relational databases, and object-oriented databases

When should you choose flat files?
- Voluminous data (e.g., images)
- Temporary data (e.g., core file)
- Low information density (e.g., archival files, history logs)

When should you choose a relational or an object-oriented database?
- Concurrent accesses
- Access at finer levels of detail
- Multiple platforms or applications for the same data

When should you choose a relational database?
- Complex queries over attributes
- Large data set

When should you choose an object-oriented database?
- Extensive use of associations to retrieve data
- Medium-sized data set
- Irregular associations among objects
3. Designing the Global Control Flow

**Control flow** is the sequencing of actions in a system.
- Control flow is a design problem.

There are 3 possible control mechanisms:

- **Procedure-driven control**: Operations wait for input whenever data is needed. Used mainly in systems written using procedural-languages.

- **Event-driven control**: A main loop, also waits for an external event. A main loop used. Useful for testing subsystems. Also the most commonly used.

- **Threads**: The system creates an arbitrary number of threads and run them concurrently. Each thread is run on the basis of procedure-driven.
1. **Procedure-driven control:**

Operations wait for input whenever data is needed. Used mainly in systems written using procedural-languages.

```java
Stream in, out;
String userid, passwd;

/* Initialization omitted */
out.println("Login:");
in.readln(userid);
out.println("Password:");
in.readln(passwd);
if (!security.check(userid, passwd)) {
    out.println("Login failed.");
    system.exit(-1);
}

/* ...*/
```

Source: OOSE-Bernd Bruegge & Allen H. Dutoit
2. Event-driven control:
A main loop, also waits for an external event. A main loop used. Useful for testing subsystems. Also the most commonly used.

```java
Iterator subscribers, eventStream;
Subscriber subscriber;
Event event;
EventStream eventStream;
/* ... */
while (eventStream.hasNext()) {
    event = eventStream.next();
    subscribers = dispatchInfo.getSubscribers(event);
    while (subscribers.hasNext()) {
        subscriber = subscribers.next();
        subscriber.process(event);
    }
} /* ... */
```

Source: OOSE-Bernd Bruegge & Allen H. Dutoit
3. Threads-driven control: Concurrent variation of the procedure driven control flow

```java
Thread thread;
Event event;
EventHandler eventHandler;
boolean done;
/*@ ...*/
while (!done) {
    event = eventStream.getNextEvent();
    eventHandler = new EventHandler(event);
    thread = new Thread(eventHandler);
    thread.start();
}
/*@ ...*/
```

Source: OOSE-Bernd Bruegge & Allen H. Dutoit
Identifying Services & Boundary Conditions

• Configuration
  – Creation of the object and destroying of the object

• Start-up and Shutdown
  – For each component add 3 uses cases – Start, Shutdown and Configuration of the component

• Exception handling
  – Notify the user when there is any error
System Design Document
1. Introduction
   1.1 Purpose of the system
   1.2 Design goals
   1.3 Definitions, acronyms, and abbreviations
   1.4 References
   1.5 Overview
2. Current software architecture
3. Proposed software architecture
   3.1 Overview
   3.2 Subsystem decomposition
   3.3 Hardware/software mapping
   3.4 Persistent data management
   3.5 Access control and security
   3.6 Global software control
   3.7 Boundary conditions
4. Subsystem services
Glossary
System Design and Decomposition summary

- **Subsystem decomposition** describes the decomposition into subsystems and the responsibilities of each. This is the main product of system design.
- **Hardware/software mapping** describes how subsystems are assigned to hardware and off-the-shelf components. It also lists the issues introduced by multiple nodes and software reuse.
- **Persistent data management** describes the persistent data stored by the system and the data management infrastructure required for it. This section typically includes the description of data schemes, the selection of a database, and the description of the encapsulation of the database.
- **Access control and security** describes the user model of the system in terms of an access matrix. This section also describes security issues, such as the selection of an authentication mechanism, the use of encryption, and the management of keys.
- **Global software control** describes how the global software control is implemented. In particular, this section should describe how requests are initiated and how subsystems synchronize. This section should list and address synchronization and concurrency issues.
- **Boundary conditions** describes the start-up, shutdown, and error behavior of the system. (If new use cases are discovered for system administration, these should be included in the requirements analysis document, not in this section.)
Questions?
References

• Use Cases Combined with BOOCH, OMT UML Process and Products
  - Putnam P Texel, Charles B Williams

• Object-Oriented Software Engineering Using UML Patterns, and JAVA
  - Bernd Bruegge & Allen H. Dutoit

• Software Engineering 9th Edition
  - Ian Sommerville