260CT
Software Engineering

Dr. Yih-Ling Hedley
Email: aa0817@coventry.ac.uk

Software Design

- **Architectural Level**
  - Designs are broken down, such as into layers, components etc. - a kind of conceptual module
- **System Level**
  - **System/sub-system design**
    - ✓ Concerned with **design of each of the components of the system**
  - **Detailed design**
    - ✓ Concerned with **design of objects and classes**
    - ✓ Designs **inputs**, **outputs**, **processes**, **file** or **database structures** into classes

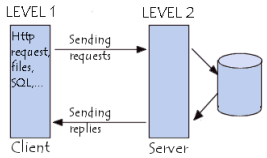
Software Application Architecture

- Software application architecture patterns that
  - Define a **structured solution** that meets all of the technical and operational requirements
  - Optimise common quality attributes such as **performance, security, and manageability**
  - Focus on how the components within an application are used by, or interact with, others within the application
  - Select **data structures and algorithms** or the implementation details of individual components

 Source: Microsoft Application Architecture Guide, 2nd Edition

Application Architectural Design 1

- **2-Tier Client/Server Architecture**: Segregates the system into two applications, where the client makes requests to the server whilst the server (containing a database with application logic with stored procedures) responds to the request from the client.



 Source: Microsoft Application Architecture Guide, 2nd Edition

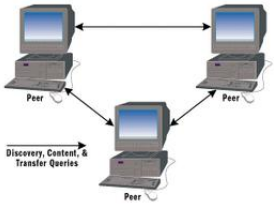
Application Architectural Design 1.1

- **2-Tier Client/Server Architecture**:
  - **Greater security**: data stored on the server with a greater control of security
  - **Centralized data access**: data is stored only on the server
  - **Ease of maintenance**: communicated via several servers, a client unaffected by a server repair, upgrade, or relocation.

 Source: Microsoft Application Architecture Guide, 2nd Edition

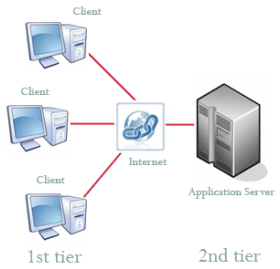
Application Architectural Design 1.2

- **2-Tier Client/Server: Peer-to-Peer (P2P)**
  - allows the client and server to change their roles to distribute and synchronize files and information across multiple clients.



Application Architectural Design 1.3

- **2-Tier Client/Server: Application servers**:
  - the server hosts and executes applications accessed by a **thin client** through a browser or specialized client installed software.



Activity 1

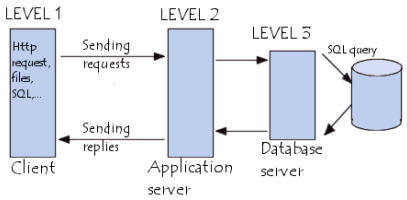
- Identify the issues associated with 2-Tier application architecture.

Activity 1: Feedback

- The issues with 2-Tier architecture:
  - close combined application data and business logic on the server – may have negative impact on system **extensibility and scalability**
  - dependence on a central server - may have negative impact on system **reliability**.

Application Architectural Design 2

- **3-Tier Client/Server Architecture**

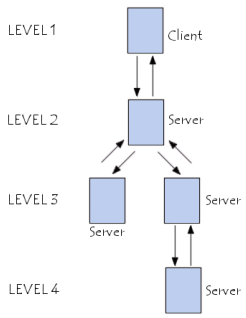


Application Architectural Design 2.1

- **3-Tier (or N-Tier) Client/Server Architecture**: evolved from 2-Tier architecture to address issues with better:
  - **Maintainability**: each tier independent of the other tiers, updates or changes can be carried out without affecting the application as a whole.
  - **Scalability**: tiers are based on the deployment of layers, thus easy to scale out an application
  - **Flexibility**: increases as each tier managed or scaled independently
  - **Availability**: exploit the modular architecture of enabling systems using easily scalable components, which increases availability.

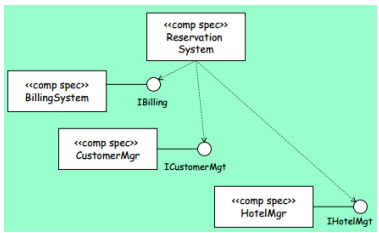
Application Architectural Design 2.2

- **N-Tier Client/Server Architecture**



Application Architectural Design 3

- **Component-Based Architecture:** Decomposes application design into reusable functional or logical components that expose well-defined communication interfaces.



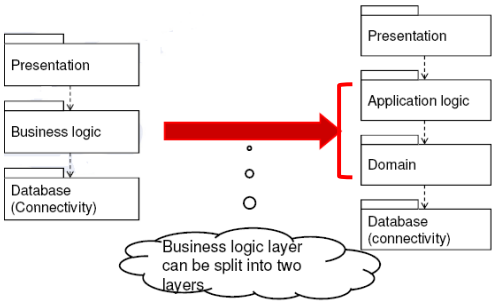
Application Architectural Design 4

- **Layered Architecture**: Partitions the concerns of the application into stacked groups (layers).
- **Layered vs. Tiered architecture**: the layers of an application may reside on the same physical computer (the same tier) or may be **distributed** over separate computers (**n-tier**).
- Example:

Application Architectural Design 4.1

- **Layered Architecture**: Example
  - **Client layer**: involved with users with several different types of clients, e.g. HTML web page.
  - **Client presenter layer**: contains the presentation logic, i.e. how business objects are displayed to users, e.g. the pop-up screens or drop-down menus
  - **Business layer**: also domain layer, handles and encapsulates all of business domains and rules.
  - **Persistence layer**: handles the read/write of the business data to the data access layer (DAL).
  - **Data access layer**: an external data source, e.g. a database.

Basic Layered Models:
Three & Four Layered Architectures



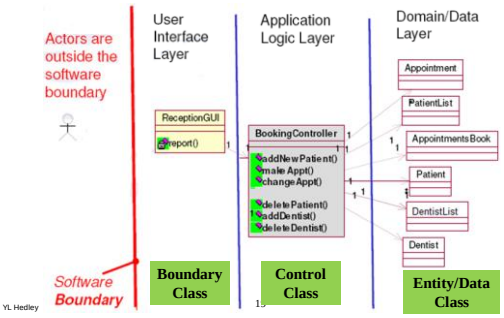
Layered Architecture: Four Layers

- **User Interface Layer (Presentation Layer)**:
  - contains **boundary classes** (e.g. GUI objects) to handle **input** and **output** with the external actors.
- **Application Logic Layer**:
  - new 'invented' **control classes** are added to **coordinate the application logic** for each use case.
- **Domain Layer**:
  - data objects contained in **data classes** from business domain form the basis of the layer or those not identified at analysis stage.
- **Data Access (Database) Layer**

Software Design: Static Model

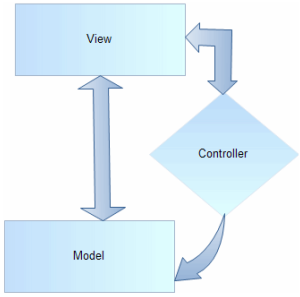
- **Static model** of the design, to include the following **stereotypes** that differentiate the roles objects can play
  - **Boundary classes**: to handle communication with actors. To identify the main **interfaces** with users and other systems, software and hardware devices
  - **Entity/data classes:** to represent something (from the application domain and external to the system) about which the system must **store information.**
  - **Control classes:** control objects **co-ordinate and control** other objects
    - Deal with business/application logic, such as register new patients

Layered Architecture: PDS Example
(The data access layer is omitted)



Architectural Pattern: MVC 1

- MVC, **Model-View-Controller** Pattern
  - the Controller handles events from the View and inform the Model on behalf of View



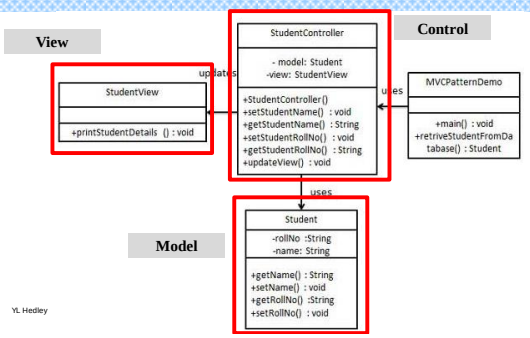
Source: http://www.codeproject.com

Architectural Pattern: MVC 2

- MVC Pattern
  - **Model** - represents an object or data and has logic to update controller if its data changes.
  - **View** - represents the visualization of the data that model contains.
  - **Controller** - acts on both Model and View, and controls the data flow into model object and updates the view whenever data changes. It separates View and Model.

Source: http://www.codeproject.com

Model-View-Controller: Example

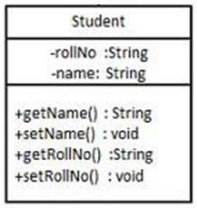


MVC Pattern: Code 1

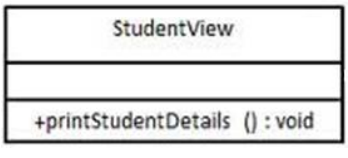
Create Model.

Student.java

```
public class Student {
   private String rollNo;
   private String name;
   public String getRollNo() {
      return rollNo;
   }
   public void setRollNo(String rollNo) {
      this.rollNo = rollNo;
   }
   public String getName() {
      return name;
   }
   public void setName(String name) {
      this.name = name;
   }
}
```



MVC Pattern: Code 2



Create View.

StudentView.java

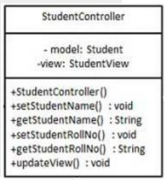
```
public class StudentView {
   public void printStudentDetails(String studentName, String studentRollNo){
      System.out.println("Student: ");
      System.out.println("Name: " + studentName);
      System.out.println("Roll No: " + studentRollNo);
   }
}
```

MVC Pattern: Code 3

Create Controller.

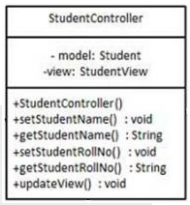
StudentController.java

```
public class StudentController {
   private Student model;
   private StudentView view;

   public StudentController(Student model, StudentView view){
      this.model = model;
      this.view = view;
   }

   public void setStudentName(String name){
      model.setName(name);
   }

   public String getStudentName(){
      return model.getName();
   }
```



MVC Pattern: Code 3.1

```
   public void setStudentRollNo(String rollNo){
      model.setRollNo(rollNo);
   }

   public String getStudentRollNo(){
      return model.getRollNo();
   }

   public void updateView(){
      view.printStudentDetails(model.getName(), model.getRollNo());
   }
}
```

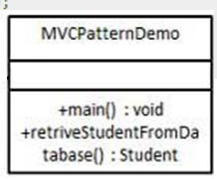


MVC Pattern: Code 4

Use the *StudentController* methods to demonstrate MVC design pattern usage.

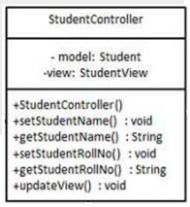
MVCPatternDemo.java

```
public class MVCPatternDemo {
   public static void main(String[] args) {

   private static Student retriveStudentFromDatabase(){
      Student student = new Student();
      student.setName("Robert");
      student.setRollNo("10");
      return student;
   }
```



MVC Pattern: Code 4.1

```
      //fetch student record based on his roll no from the database
      Student model  = retriveStudentFromDatabase();

      //Create a view : to write student details on console
      StudentView view = new StudentView();

      StudentController controller = new StudentController(model, view);

      controller.updateView();

      //update model data
      controller.setStudentName("John");

      controller.updateView();
   }
}
```



MVC Pattern: Exercise

- Implement the following in a programming language of your choice.