

An Architecture

- Didn't you think of how can we organise our components to work together?
- How our work is going to be **integrated**?
- We need to have **an overall view** at the design of the system
- In other words, we want to have an architecture for our system

What is an architecture?

- There are many definitions out there for an architecture.
- A refined version of Mary Shaw and David Garlan definition of the architecture
 - “ Software architecture encompasses the set of significant decisions about the organization of a software system”
- Decisions about what?

Software Architecture is

- Decisions about:
 - the selection of the **structural elements and their interfaces** by which the system is composed;
 - behavior as specified in collaboration among those elements
 - composition of these structural and behavioral elements into larger subsystems
 - and an **architectural style** that guides this organization
- architecture also involves functionality, usability, resilience, performance, reuse, comprehensibility, economic and technology constraints and tradeoffs

» Mary Shaw and David Garlan

Architectural Decisions Produce Varying Results



.Net



J2EE

<http://www.corej2eepatterns.com/presos/CoreJ2EEPatterns2.pdf>

Architecture styles

- You can think of architecture styles and patterns as sets of principles that shape an application.
- Architectural styles can be organized by their key focus area.

Category	Architectural Style
<i>Deployment</i>	Client/Server, N-Tier, 3-Tier
Structure	Component-Based, Object-Oriented, Layered Architecture
<i>Communication</i>	Service-Oriented Architecture (SOA), Message Bus

Architectural Styles Examples

Style	Description
<i>Client/Server</i>	<i>Segregates the system into two applications, where the client makes requests to the server. In many cases, the server is a database with application logic represented as stored procedures.</i>
Layered	Partitions the concerns of the application into stacked groups (layers).
N Tiers	Segregates functionality into separate segments in much the same way as the layered style, but with each segment being a tier located on a physically separate computer.
Service Oriented Architectures	Refers to applications that expose and consume functionality as a service using contracts and messages.
Component Based	Decomposes application design into reusable functional or logical components that expose well-defined communication interfaces.
MVC	Model View Controller

Modelling the domain

Display students with a certain major

The View

```
// Step 2: execute the query
Statement stmt = conn.createStatement();
String qry = "select sname, gradyear "
            + "from student, dept "
            + "where did = majorid "
            + "and dname = '" + major + "'";
ResultSet rs = stmt.executeQuery(qry);

// Step 3: loop through the result set
while (rs.next()) {
    String sname = rs.getString("sname");
    int gradyear = rs.getInt("gradyear");
    System.out.println(sname + "\t" + gradyear);
}
rs.close();
```

Much Better:

Separate the view from the data processing

The View

```
Collection<DBStudent> students=DBStudent.findMajors(major);  
for(DBStudent s : students){  
    System.out.println(s.getName()+"\t"+s.getGradYear());  
}
```

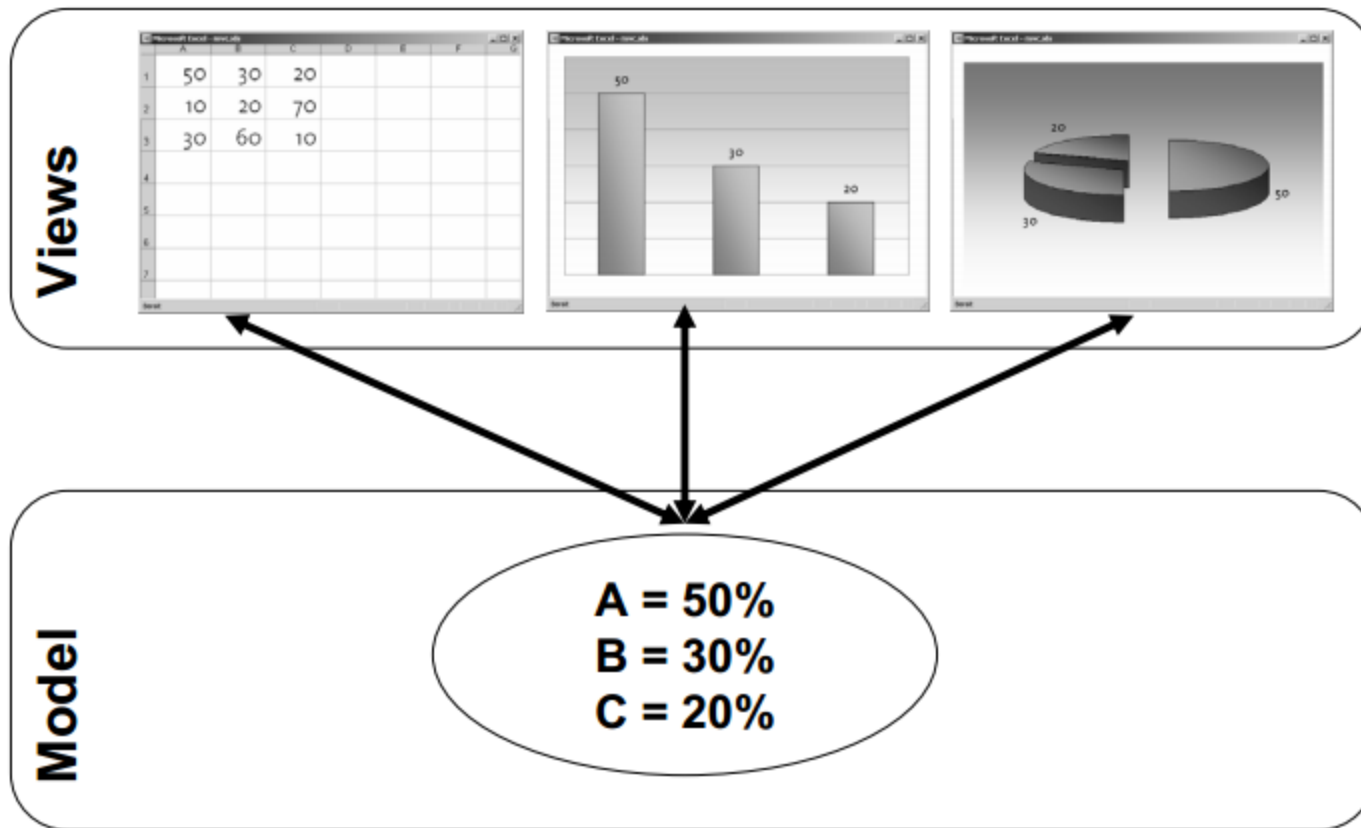
The view code doesn't see any of the JDBC code
Totally separate from database type or schema
config

Processing

Includes the JDBC code

```
public class  
    DBStudent
```

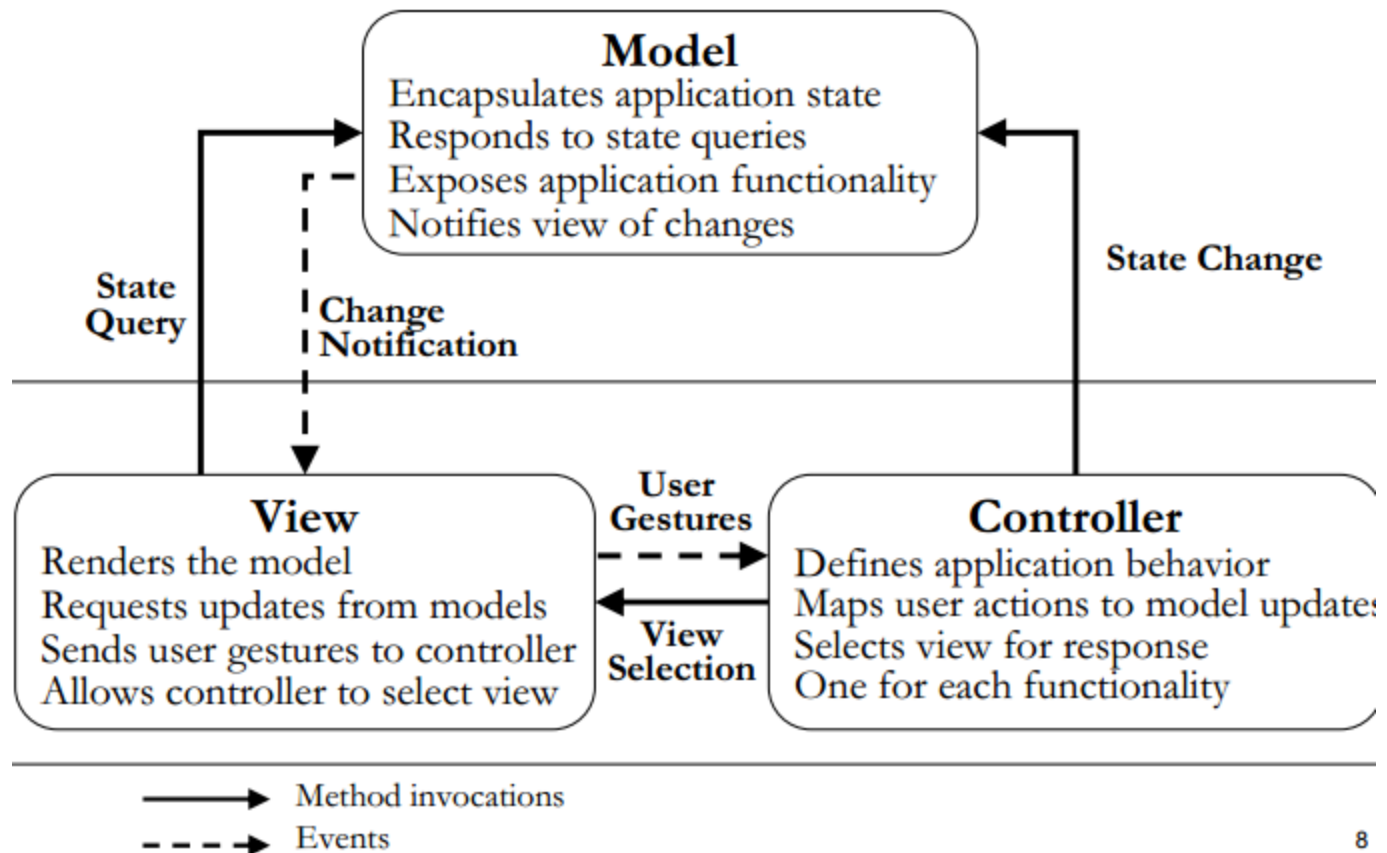

Separating Concerns and Reusability



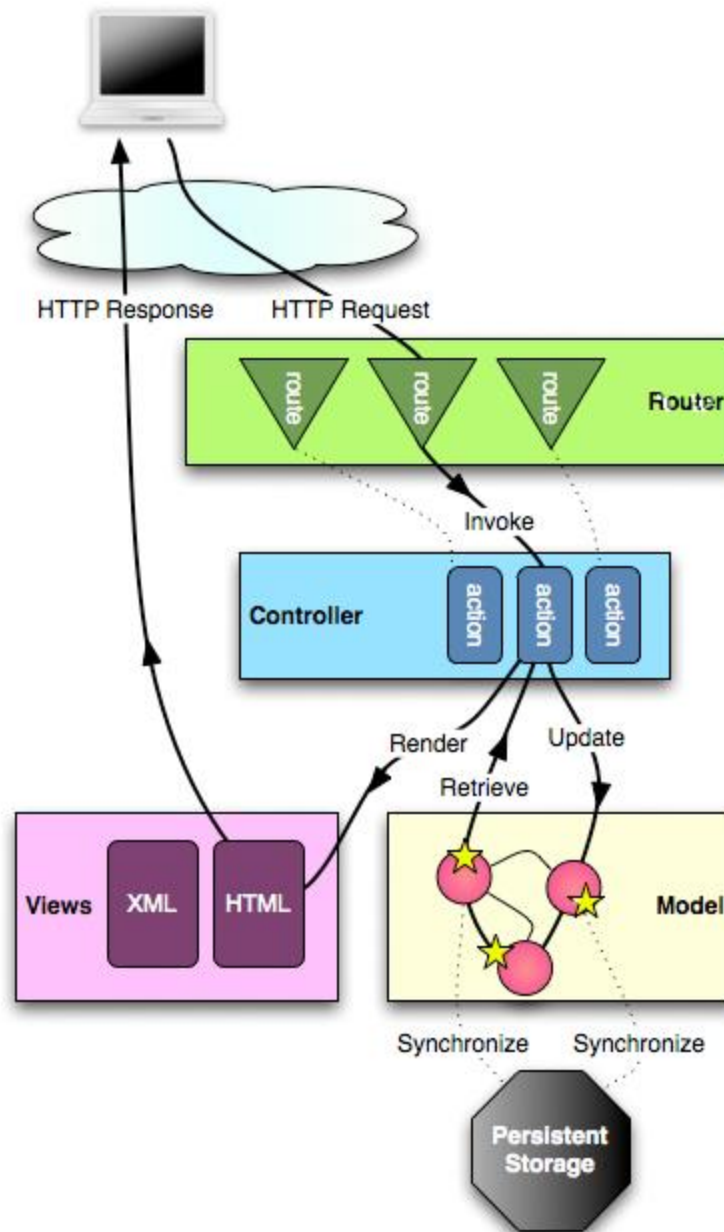
View and Model classes

- An objected oriented program should separate the view from the model
- **View classes:**
 - Exclusively for the interface
 - Take input from user
 - Display output to user
 - Have no idea how the data is processed
- **Model classes:**
 - They retrieve and store data in database
 - Have no idea of how data is displayed

Model View Controller



Life Cycle of a HTTP request



**FOR OUR LOGIC, WE WILL FOLLOW
A COMPONENT BASED STRUCTURE**

Component Based Software Architecture

- Component-based architecture describes a software engineering approach to system design and development.
- It focuses on the decomposition of the design into individual functional or logical components that expose well-defined communication interfaces containing methods, events, and properties.
- This provides a higher level of abstraction than object-oriented design principles, and does not focus on issues such as communication protocols and shared state.

key principle of the component-based style

- **Reusable.**
 - Components are usually designed to be reused in different scenarios in different applications. However, some components may be designed for a specific task.
- **Replaceable.**
 - Components may be readily substituted with other similar components.
- **Not context specific.**
 - Components are designed to operate in different environments and contexts. Specific information, such as state data, should be passed to the component instead of being included in or accessed by the component.

Key Principles Cont'd

- **Extensible.**
 - A component can be extended from existing components to provide new behavior.
- **Encapsulated.**
 - Components expose interfaces that allow the caller to use its functionality, and do not reveal details of the internal processes or any internal variables or state.
- **Independent.**
 - Components are designed to have minimal dependencies on other components. Therefore components can be deployed into any appropriate environment without affecting other component or systems.

Inter-component requirements

- “An application’s architecture can be well observed by checking the dependencies among all its modules”
- Each component should be a black-box (opaque) for other components
 - Each component should aim to design for a level of abstraction through which other components are not involved in implementation by any means

Interfaces

- “interface is the definition of a collection of one or more methods, and zero or more attributes, ideally one that defines a cohesive set of behaviors. ”

Behaviors that you don't want their target users (programmers) to be considered with how they implemented.

The Challenge

- We already have our components defined (almost ;)
- We would like to move from our **product backlog** to **extracting dependencies** among our components.
- We can't have redundant logic or classes (We are not speaking UI here).
- **Our code has to be integrated from the start.**

The architecture Workshop

- It is known that agile techniques have a problem in taking care of an architecture as it is traditionally done.
- We can't have one **Genius guy** who takes a decision. Decisions are **collaborative**. **Developers** have to be involved.
- We need to get architecture into a **continuous inspection and adaptation** of SCRUM.

Steps to be done:

1. We will conduct an architecture workshop
2. A representative of each component should be in charge of decisions
3. Review of architecture artefacts will take place after each sprint
4. External review of the architecture at the end of the process (mine)

(only @GUC)

Sketch Your Components on the Whiteboard

User Management



Project Management



Budget Management



Steps to do


- Draw boxes that matches to different themes from your class diagrams ...it could be a complete match with your component or not.
- Take every story (starting by high priority) and think whether it belongs
 - Inside the theme box?
 - On the borderline with another theme (s)?
 - Requires adding another story Inside another theme(Interface)?
Depends on specific data or story needed from another theme?
 - (represent this on the white board drawing with different colors)
- Keep doing the above steps till all your stories are on the board (either inside a component box or on the border line)

Example

- Story:
 - Add users from different groups to a project
 - Success:
 - Next to a project, you click on the button 'Add members' which will open a filtration page that allow choosing a **community, a group showing their members in a multiple select list**. Upon selection and submission, members are added and instantly appear highlighted in the list. The members can be de-selected from there as well to undo the selection. An email will be sent to the members who were added for notification.
 - Failure:
 - Members who already exist will appear highlighted to avoid adding members twice.
 - The add button won't appear or will be dimmed in case there are no other members in the system to invite


Sketch Your Components on the Whiteboard

User Management



```
get Communities()  
getGroups(communityid)  
getMembers(groupid)  
getAllMembers(communityid)
```

Project Management



```
Add members to a  
project
```

Budget Management



Tips

- In order to decide whether a story should go in the component box or on the border, you should think in the following
 - Plug in the scenarios that you have with input and possible output
 - Are you waiting for input from another component? Do you need to provide input to the other component?

After the board is complete

- Refine the product backlog according to the discussion in the architecture
- Team of architects finalise conflicts about ownership of classes, stories or signatures
- Add a section called dependency stories to the product backlog
- Add the additional required stories to the product backlog with unique Id (can be derived from the original story)
 - get a list of all members in a group required for a user by providing the ID in order to add members (S9)
- The added story will get the same priority as the original story