DESIGN: ARCHITECTURE AND METHODOLOGY

Software Engineering
CS 130
Donald J. Patterson

Content adapted from Essentials of Software Engineering 3rd edition by Tsui, Karam, Bernal Jones and Bartlett Learning
Design Topics Covered

1. Architectural vs. Detailed design

2. “Common” architectural styles, tactics and reference architectures

3. Basic techniques for detailed design

4. Basic issues with user-interface design
Design

- **Starts** mostly from/with *requirements* (evolving mostly from *functionalities* and other non-functional characteristics)

- How is the software solution going to be structured?
  - What are the *main components* --- *(functional comp)*
    - *Often* directly from *Requirements’ Functionalities (use Cases)*
  
    - How are these *components related*?
      - possibly re-organize the components *(composition/decomposition)*

- Two main levels of design:
  - Architectural *(high-level)*
  - Detailed design

- How should we depict design--notation/language?
DESIGN: ARCHITECTURE & METHODOLOGY

Relationship between Architecture and Design

Requirements → Architecture → Detailed Design

Detailed Design come from Requirements & Architecture
Software Architecture

• **Structure(s) of the solution**, comprising:
  1. Major Software **Elements**
  2. Their externally visible **properties**
  3. **Relationships** among elements

• Every software system has an architecture

• **May have Multiple structures**!
  – multiple ways of organizing elements, *depending on the perspective*

• **External properties of components (& modules)**
  – **Component (Module) interfaces**
  – **Component (Module) interactions**, rather than internals of components and modules
Views and Viewpoints

- **View** – Representation of a system structure
- **4+1 views** (by Kruchten)
  - Logical (OO decomposition – key abstractions)
  - Process (run-time, concurrency/distribution of functions)
  - Subsystem decomposition
  - Physical architecture
  - +1: use cases
- **Other classification** (Bass, Clements, Kazman)
  - Module
  - Run-Time
  - Allocation (mapping to development environment)
- Different views for different people
We discuss Architectural Styles/Patterns as “reusable” starting point for Design activities.

1. Pipes-and-Filters
2. Event-Driven
3. Client-Server
4. Model-View-Controller (MVC)
5. Layered
6. Database Centric
7. Three tier
Pipe-Filter architecture style

• The high level design solution is decomposed into 2 “generic” parts (filters and pipes):
  – **Filter** is a service that transforms a stream of input data into a stream of output data
  – **Pipe** is a mechanism or conduit through which the data flows from one filter to another

**Reminds one of DFD without the data store or source sink**

Problems that require **batch file processing** seem to fit this architecture: e.g. payroll, compilers, month-end accounting
The high level design solution is based on an event dispatcher which manages events and the functionalities which depends on those events. These have the following characteristics:

- Events may be a simple notification or may include associated data.
- Events may be prioritized or be based on constraints such as time.
- Events may require synchronous or asynchronous processing.
- Events may be “registered” or “unregistered” by components.

Problems that fit this architecture includes real-time systems such as: airplane control; medical equipment monitor; home monitor; embedded device controller; game; etc.

- - - try a commercial flight control system - - -
Basic Client-Server Style

Application split into client component and server component
Client-Server Style

- Client may connect to more than one server (servers are usually independent)
Model View Control (MVC) Style

- Separates **model (data)** from **view**
- **Controller** often integrated with **view** nowadays

Most of the internet web applications fall under this style
The “outer” layer can only ask for service from the “inner” layer or “upper” layer can only ask for service from “lower” layer.

- **strict layering**----- only directly inside or below layers
- **relaxed layering**---- any inside or below layers
DESIGN: ARCHITECTURE & METHODOLOGY

Shared Data (DB) centric style

Hospital DB

- Patient processing
- Room Scheduling
- Purchasing
- Nurses Scheduling

Very popular within the business applications community
Three tier style (mixture)

- Clients do not access DB directly
- Better **Flexibility, integrity** (why?)

Reminds one of Client-Server or MVC?
Architectural Tactics

- **Tactics** (in contrast to architectural style) are for **solving “smaller, specific” problems**

- Do not affect overall structure of system

- Example: we add specific functionalities or component (e.g. to **increase reliability**) in the design for fault detection ---- especially for distributed systems:
  - **heartbeat**
  - **ping / echo**
Reference Architectures

- Full-fledged architectures
- Serves as "templates" or as "a reference" for a class of systems
- Example: J2EE Reference Architecture (MVC2)

There also are "application domain specific" reference architectures
Detailed Design

- Further Refine Architecture and match with Requirements

- How detailed?

- How formal?

- Maybe of different levels of detail for different views
Functional Decomposition Technique

• Dates back to “structured programming” [now (non-OO)Web apps with PHP tool]

• Start with: main (task/requirements) -> module

• Refine into sub-modules

• There are alternative decompositions
Possible Decomposition of (student-course management app)
“Alternative” Decomposition/Composition

0. Main

1. Students
2. Courses
3. Sections
4. Registration
5. Database
  5.1 Add
  5.2 Modify
  5.3 Delete
Relational Database Design

• Most databases use relational technology

• Relations (tables)
  – Two-dimensional sets
  – **Rows** (tuples), **Columns** (attributes)
    • A Row may be an entity, Columns may be relationship or attributes
  – **Primary key** (unique identifier) – for search
  – **Foreign keys** (connects tables)
Database Design

- **Conceptual modeling** (done during analysis/requirement phase) produces ER diagram
- ** Logical design** (to relational)
- **Physical design** (decide data types, etc.)
- **Deployment/maintenance**
  - Low-level physical (which hard-drive etc)
  - Adjustment of indexes
Entity-Relationship diagrams

- **Entities** (rectangles)
  - Weak – double lines
- **Relationships** (diamonds)
- **Attributes** (ovals)
  - Multi-valued - double lines
  - Identifying - underlined
Logical DB Design - Entities

- Table per entity
- Flatten composite attributes
- For **weak entities**, add the primary key of the **strong entity**
Logical DB Design – **Multi-valued**

- New table needed for multi-valued attributes
Logical DB Design - Relationships

- If one side related to just one entity, add foreign key to that side
- For many-to-many, need new table
- For ternary, need new table

STUDENT

<table>
<thead>
<tr>
<th>Id</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>Gender</td>
</tr>
</tbody>
</table>

TAKEs

<table>
<thead>
<tr>
<th>CourseNumber</th>
</tr>
</thead>
<tbody>
<tr>
<td>SectionNumber</td>
</tr>
<tr>
<td>Semester</td>
</tr>
<tr>
<td>Year</td>
</tr>
<tr>
<td>Student_id</td>
</tr>
<tr>
<td>Grade</td>
</tr>
</tbody>
</table>

SECTION

<table>
<thead>
<tr>
<th>CourseNumber</th>
</tr>
</thead>
<tbody>
<tr>
<td>SectionNumber</td>
</tr>
<tr>
<td>Semester</td>
</tr>
<tr>
<td>Year</td>
</tr>
<tr>
<td>Time</td>
</tr>
<tr>
<td>Location</td>
</tr>
</tbody>
</table>
Physical DB Design

• **Data types for each attribute**
  – Check which ones your DBMS support
  – Encoding

• **Decide on Indexes**
  – Searches are faster, updates are slower
  – Indexes consume space
  – Can always adjust during deployment

• **Denormalization done sometimes (avoid)**
DESIGN: ARCHITECTURE & METHODOLOGY

OO Design

• **First step:** Review & Refine use cases

• Decide
  – Which classes to create
  – How are the classes related

• Use UML as the Design Language
Use case diagram

- Register For Section
  - Choose Section
  - Add Course
  - Add Section
- Add Student
- Registrar

Student
Class Design

- **Classes** represent real-world entities or system concepts
- Organized into **classes**: objects in a class have similar characteristics
- **Classes** have properties (attributes or data)
- **Classes** also have methods (performs functions)

<table>
<thead>
<tr>
<th>Student</th>
</tr>
</thead>
<tbody>
<tr>
<td>dateOfBirth : Date</td>
</tr>
<tr>
<td>name : String</td>
</tr>
<tr>
<td>getAgeInYears() : int</td>
</tr>
<tr>
<td>getAgeInDays() : int</td>
</tr>
</tbody>
</table>
UML Class diagrams

- **Association**

  Student 0..* Is Enrolled 1..1 School

- **Composition**

  School

  Use “no-fill” diamond for aggregation
UML Class diagrams - Inheritance
UML State diagram

depicting a student’s “status” in school

Accepted → enroll: → Active → graduate: → Alumni

Inactive → enroll: → Expelled

enroll: → fails to enroll:

expel:
DESIGN: ARCHITECTURE & METHODOLOGY

UML “Sequence Diagram”

used to depict a flow of interactions

Diagram:
- RegistrationGui: register(aStudent : Student)
- aSection: prerequisites(), prerequisites *
- course: hasTaken( prerequisite ), boolean
- student: Add(aStudent)
- success: void
User Interface Design

- Most apparent to the user
- Two main issues
  - i) Flow of interactions
  - ii) Look and feel
- Types of interfaces
  - Command-Line
  - Text menus
  - Graphical (GUI)
Flow of interactions

Prototype Screens

1. Registration:
   Select term

2. Registration: shows term
   Select first course

3. Registration: shows term, course(s) with schedule and cost
   Select *Additional course; *Delete course; *Finish registration

4. Registration: shows final schedule
   Select Confirm or Cancel
DESIGN: ARCHITECTURE & METHODOLOGY

High Fidelity Prototype
User: aStudent

Screens:

Process:
Student selects course and clicks “Add Course”

Welcome aStudent

Registration
Desired School term to register - Spring 2012
Select course to add ALL Courses
Add Course Cancel Help

Welcome aStudent

Registration
Desired School term to register - Spring 2012
Desired Schedule: SWE 2313 Intro to Software Engineering
Add another course Confirm Schedule Cancel Help
User interaction added to the sequence diagram
Norman’s 7 Stage Model

1. User forms goal
2. User forms intention
3. Specifies action
4. Executes action
5. User perceives feedback
6. Interprets feedback
7. Evaluates
The GOMS Model
(an “advanced” topic for UI)

• Consider different kinds of users

• Four factors (for the kind of user)
  – Goals of the user
  – Operations provided by the system
  – Methods or the sequence of operations
  – Selection Rules for the methods
Other UI Issues

- Kinds of users
- Heuristics
- UI Guidelines
- Multicultural issues
- Metaphors
- Multiplatform software
- Accessibility
- Multimedia Interfaces
### Sample HTML

```html
<form method="GET" action="something.php">
    <p>
        Username:<br>
        <input type="text" name="username">
    </p>
    <p>
        Password:<br>
        <input type="password" name="password">
    </p>
    <input type="submit" value="Login">
</form>
```

### Visual result (possible)

Username: 
Password: 

Login
Model-View-Controller (MVC) software project
Object-Relational Impedance Mismatch
(an “advanced” topic)

How do we handle mismatches between object-oriented concepts and Relational DB such as:

- typing
- private and public
- inheritance and polymorphism
- nested structure versus table structure