Lecture 02: Course Overview

Emad Shihab

Adapted from: Ahmed E. Hassan
Waterfall Development Process

1. Requirement Engineering
2. Architecture Analysis
3. Design & Implement
4. Testing

- Software Requirements Specification (SRS)
- Architecture Doc
- Source Code
Course Overview

- Requirements
- Architectural Styles
- Architecture Recovery
- Design Patterns
- Project Scheduling
- Software Estimation
Requirements
Where Do Requirements Come From?

- Requirements come from users and stakeholders who have demands/needs.

- An analyst/requirement engineer:
  - Elicits these demands/needs (raw requirements).
  - Analyzes them for consistency, feasibility, and completeness.
  - Formulates them as requirements and write down a specification.
  - Validates that the gathered requirements reflect the needs/demands of stakeholders:
    - Yes, this is what I am looking for.
    - This system will solve my problems.
Types of Requirements

■ Functional Requirements
  - Specify the function of the system
  - $F(input, \text{system state}) \rightarrow (output, \text{new state})$

■ Non-Functional Requirements (Constraints)
  - Quality Requirements
    - Specify how well the system performs its intended functions
    - Performance, Usability, Maintenance, Reliability, Portability
  - Managerial Requirements
    - When will it be delivered
    - Verification (how to check if everything is there)
    - What happens if things go wrong (legal responsibilities)
  - Context / Environment Requirements
    - Range of conditions in which the system should operate
Architectural Styles
Architectural Styles of Software Systems

- **Architectural Style**
  - Form of structure, e.g.,
    - "Pipes" between components, or
    - "Layered" system, or
    - "Bulletin board" system
  - Analogy: Style of a building

- It determines:
  - the vocabulary of components and connectors that can be used in instances of that style
  - a set of constraints on how they can be combined. For example, one might constrain:
    - the topology of the descriptions (e.g., no cycles).
    - execution semantics (e.g., processes execute in parallel).
Determining an Architectural Style

We can understand what a style is by answering the following questions:

- What is the **structural pattern**? (i.e., components, connectors, constraints)
- What are the **essential invariants** of the style?
- What are some **common examples of its use**?
- What are the **advantages** and **disadvantages** of using that style?
- What are some of the **common specializations** of that style?
Architecture Recovery
Architecture Terminology

■ Conceptual Software Architecture
  – Abstract structure: Large piece of software with many parts and interconnections
  – Analogy: Blueprint of house

■ Concrete Software Architecture
  – Actual structure: Large piece of software with many parts and interconnections
  – Analogy: Actual structure of house

■ Reference Architecture
  – General architecture for an application domain
  – Example: Common structure for compilers or for operating systems
  – Analogy: Typical architecture of a house
Linux Architecture

Conceptual Architecture

Concrete Architecture
Design
Architecture vs. Design

- **Architecture**
  - Structure of system (components and connectors)
  - High level and hard to change (better get it right!)
  - Concerned with technical and non technical requirements (e.g., Security, Legal, Outsourcing)
  - Makes sense for systems with MLOCs
  - Very early in life cycle

- **Design**
  - Inner structure of the components
  - Low level (information hiding and interfaces help it change)
  - Mostly technical concerns
  - Makes sense for systems with KLOCs
  - Late in life cycle
Design Patterns

- Good designers know not to solve every problem from first principles. They reuse solutions.
- Practitioners do not do a good job of recording experience in software design for others to use.
A Design Pattern systematically names, explains, and evaluates an important and recurring design.

We describe a set of well-engineered design patterns that practitioners can apply when crafting their applications.
Project Scheduling
A project is
- a temporary endeavour undertaken to create a "unique" product or service

A project is composed of
- a number of related activities that are directed to the accomplishment of a desired objective

A project starts when
- at least one of its activities is ready to start

A project is completed when
- all of its activities have been completed
A project plan is a schedule of activities indicating
- The start and stop for each activity. The start and stop of each activity should be visible and easy to measure
- When a resource is required
- Amount of required project resources
Project Planning

- Managers should consider:
  - Resource availability
  - Resource allocation
  - Staff responsibility
  - Cash flow forecasting

- Managers need to monitor and re-plan as the project progresses towards its pre-defined goal
Cost Estimation
Software cost estimation

- Predicting the resources required for a software development process
Topics covered

- Productivity
- Estimation techniques
- Algorithmic cost modelling
- Project duration and staffing
Course Webpage

- Schedule
- Project Deliverables
  - Assignment 0 (last year’s projects)
  - Assignments 1,2,3 (marking scheme)
  - Peer evaluation
Next Class…

- Tuesday Sept 14, BIOSCI 1120
- Will cover:
  - Requirements
  - Quality Attributes