CSE 301
Software Design and Architecture

Introduction to SDA

Dr. Salisu Garba
Salisu.garba@slu.edu.ng

Objective

- To understand **principles of good design**, and techniques for the evaluation of software design quality.
- To cover the principal architectural issues associated with the design and construction of software systems including **architectural design** and documentation, component models and technologies, and frameworks.
- To introduce the students to a number of **design patterns** and their applications.
- To introduce the students to Basics of **Software Evolution**, Re-engineering & **Reverse Engineering**.

Course Contents

	Topics
1	Introduction Software Design, Architecture & Principles of good software design
2	Design Concepts & Software Quality (Correctness, Robustness Flexibility, reusability and efficiency)
3	Introduction to software architecture
4	Software architectural design (Software Architectural attributes, Attribute types, Trade-off of attributes & choices)
5	Software Architectural Styles (Data flow architectures, Layered, Event-based, Data-centered, MVC, Service Oriented)
6	OBJECT ORIENTED DESIGN Functional Modeling (Activity Diagrams, Use case diagrams)
7	OBJECT ORIENTED DESIGN Structural Modeling (Class Diagrams)
8	OBJECT ORIENTED DESIGN Behavioural Modeling (Sequence Diagrams, Behavioural State Machines)
9	OBJECT ORIENTED DESIGN Evolving the analysis models into design models
10	Introduction to Components and Component Oriented Design
11	Pattern Oriented Design (Creational patterns, Structural patterns, Behavioral pattern)
12	Basics of Software Evolution, Re-engineering & Reverse Engineering

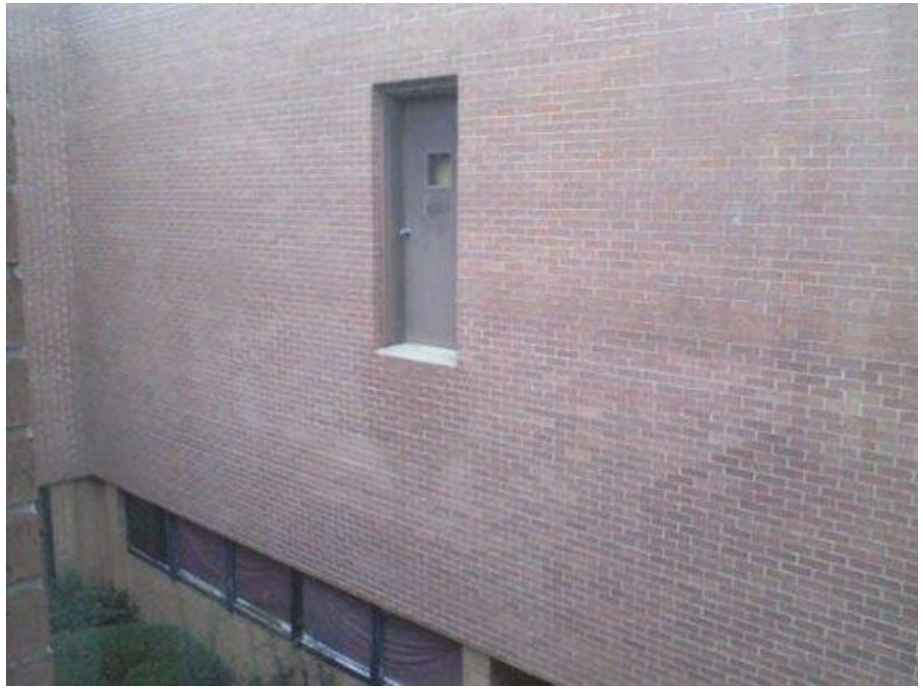
Course Administration

- Recommended Textbooks
  - Software Architecture And Design Illuminated By Kai Qian, Xiang Fu, Lixin Tao, Chong-wei Xu, Jones And Bartlett Publishers.
  - Object-Oriented Systems Analysis and Design using UML, by Simon Bennett, McRobb & Farmer McGraw-Hill.
  - Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development, By Craig Larman
- Assessment
  - The course shall be comprised of various assignments and quizzes that shall be spread through out the course.
  - Group Project.
  - Final Exams.
  - The final grade shall depend on your performance in EACH of the above mentioned KPIs.

Why SDA?

A staircase that leads right into a wall!

A door that would drop you 10 feet down!



Why SDA?

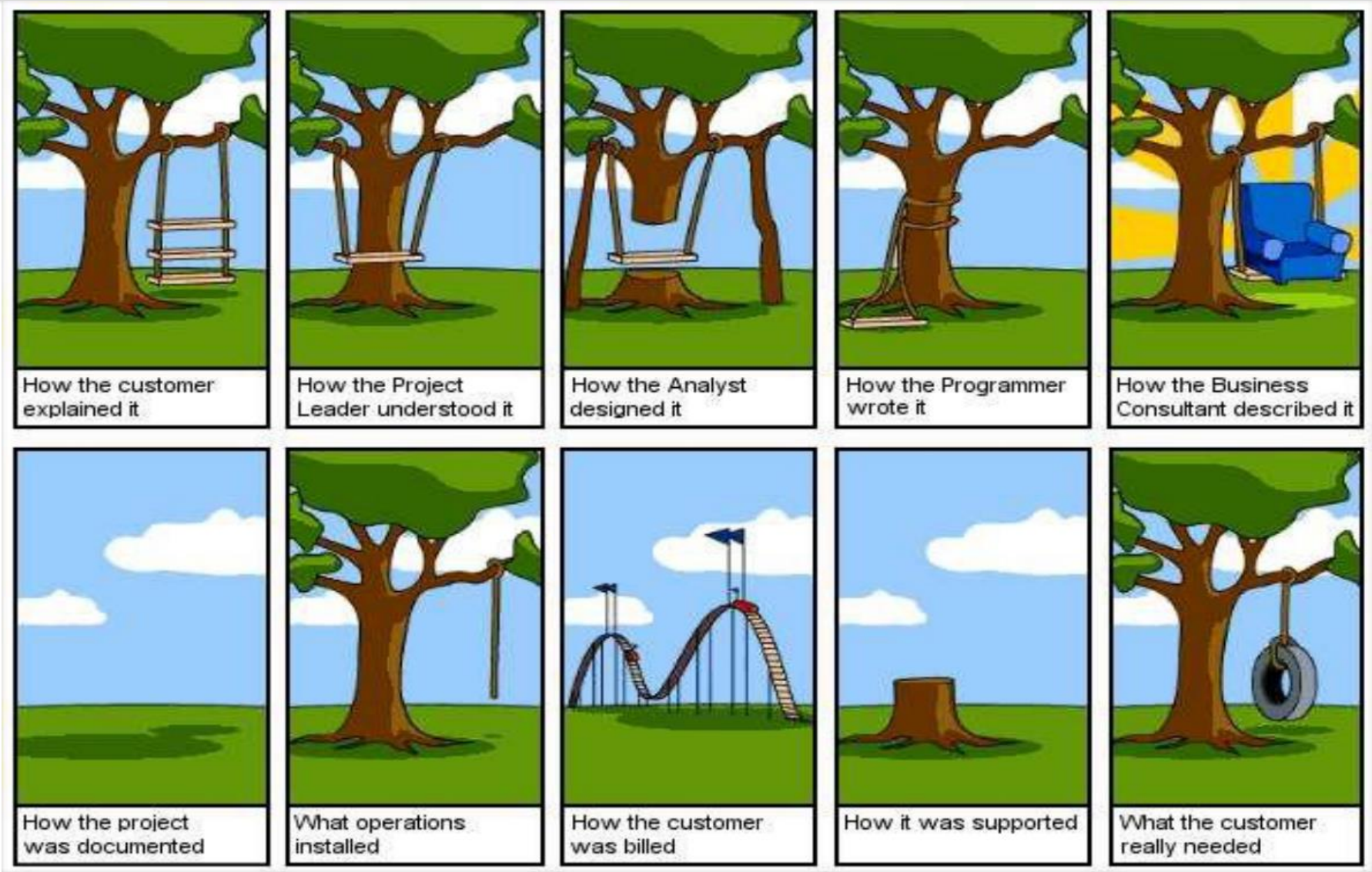
A difficult to use ATM machine

An impossible to use Bridge

Why SDA?

- What do you think is wrong in these real life scenarios?
- The **requirements are correct**!
  - A staircase next to the outer wall
  - A door on the first floor
  - An ATM outside the bank branch
  - The bridge
- The **design is flawed**!
  - The execution based on the design **results in disaster**.
- Software are no different.
  - For a useful software, it has to be ‘engineered’; which involves giving specific attention to every phase of software development.

Problems in software development



Why Design is important?

- Without a proper design, we risk building an **unstable system**
  - one that will **fail** when **small changes** are made
  - one that may be **difficult to test**
  - one whose **quality cannot be assessed**

- Design team should **not do too much**
  - Detailed design should not become code
- Design team should **not do too little**
  - It is essential for the design team to produce a complete detailed design

Sources of Problems

- Most bugs are not because of mistakes in the code…
- Specification (~= 55%), **Design (~= 25%)**, Code (~= 15%), Other (~= 5%)
  - **Requirements Definition:** Erroneous, incomplete, inconsistent requirements. “if you can’t say it, you can’t do it”
  - **Design:** Fundamental design flaws in the software.
  - **Implementation:** Mistakes in chip fabrication, wiring, programming faults, malicious code.
  - **Support Systems:** Poor programming languages, faulty compilers and debuggers, misleading development tools.
  - **Inadequate Testing of Software:** Incomplete testing, poor verification, mistakes in debugging.
  - **Evolution:** Sloppy redeployment or maintenance, introduction of new flaws in attempts to fix old flaws etc

Adverse Effects of Faulty Software Design

- Ariane 5 Flight 501 self-destructing 37 seconds after launch due to design errors in the software (data conversion from **64-bit to 16-bit**)
- An F-18 crashed because of a missing exception condition: **if … then … without the else** clause that was thought could not possibly arise.
- In simulation, an F-16 program bug caused the virtual plane to flip over whenever it crossed the equator, as a result of a **missing minus sign to indicate south latitude**.
- Mr. Blodgett's auto insurance rate tripled when he turned 101. He was the computer program's first driver over 100, and **his age was interpreted as 1**.
- A Norwegian bank ATM consistently dispersed **10 times the amount** required.
- A software flaw caused a UK bank to **duplicate every transfer payment** request for half an hour. The bank lost 2 billion British pounds!

Introduction to SD

- **Software design** is the first of three technical activities (design, code generation, and testing) that are required to build and verify the software.
- Each of these activities transforms information in a manner that ultimately results in validated computer software.
- The Software Design process itself can be defined as a process through which requirements are translated into a representation of software.

The Role of Design

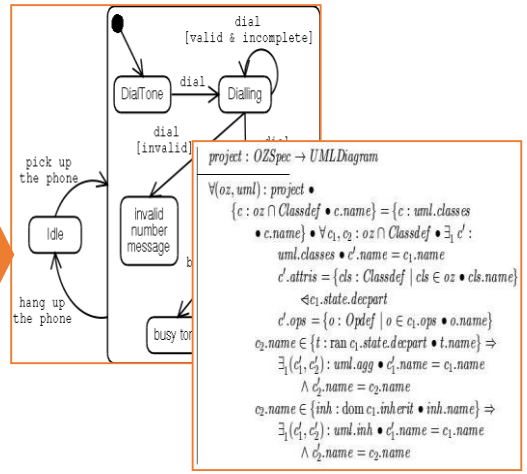
- Design play two pivotal role:

1. Clarify and refine requirements
2. Early defect detection and elimination

Requirements

Activity	Sub Activity	Description
Prioritize requirements	Review candidate requirement	Results in the concept CANDIDATE REQUIREMENT
Prioritize requirements	Assess relative value of candidate requirement	Results in a value for the attribute *relative value* in the concept CANDIDATE REQUIREMENT
Prioritize requirements	Estimate relative cost of implementing each candidate requirement	Results in a value for the attribute *relative cost* in the concept CANDIDATE REQUIREMENT
Prioritize requirements	Calculate value and cost, plot these on a cost-value diagram	Results in the concept COST-VALUE DIAGRAM
Prioritize requirements	Analyze, discuss and prioritize candidate requirements	Results in a value for the attribute *prioritization* in the concept CANDIDATE REQUIREMENT
Select requirement		Concept SELECTED REQUIREMENT is created, which is a CANDIDATE REQUIREMENT
Define release requirement document		Concept RELEASE REQUIREMENTS DOCUMENT is created, which contains a.o. at least one SELECTED REQUIREMENT
Validate release requirement		Results in a modification of the concept RELEASE REQUIREMENTS DOCUMENT, because the requirements in this document are validated.
Prepare Launch		Concept LAUNCH PREPARATION PACKAGE is created, which contains a.o. the RELEASE REQUIREMENTS DOCUMENT.

Design Models



Code

Software Design - Simplified

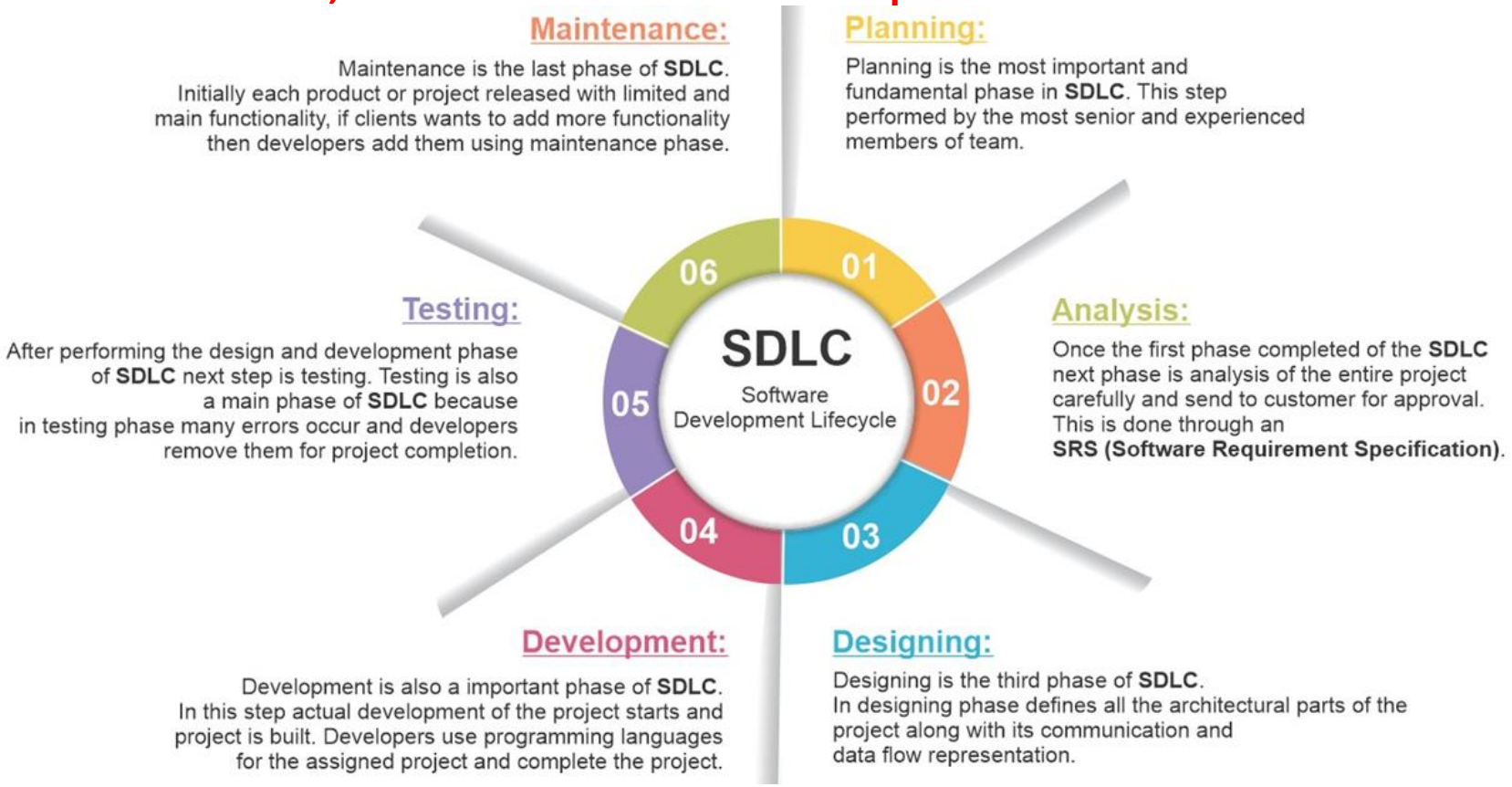
Requirements specification was about the WHAT the system will do

Design is about the **HOW the system will perform its functions**

- provides the overall decomposition of the system
- allows to split the work among a team of developers
- also lays down the groundwork for achieving non-functional requirements (performance, maintainability, reusability, etc.)
- takes target technology into account (e.g., kind of middleware, database design, etc.)

Software Design in SDLC

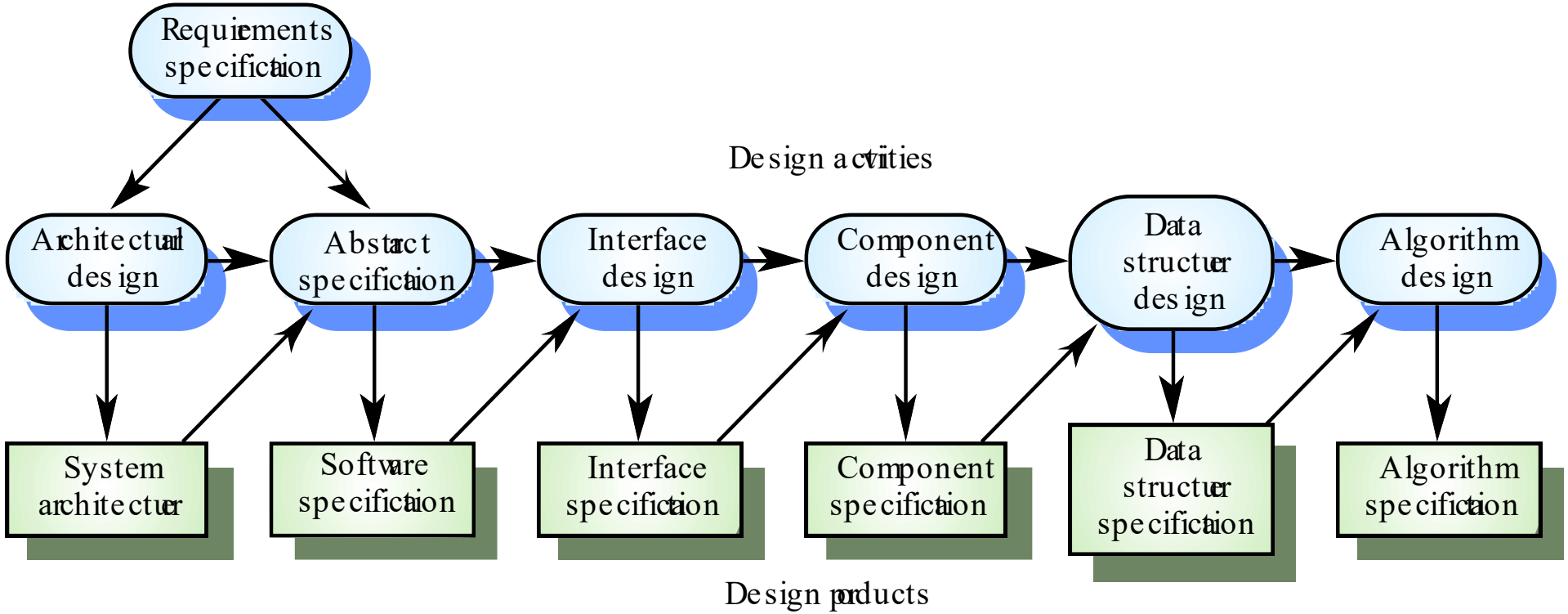
- In SDLC (Software Development Life Cycle), Design phase is one of the most important phases which focuses on four major areas of concern: data, architecture, interfaces and components.



Design Process Activities

- Architectural design
  - Modules, inter-relationships etc
- Abstract specification
  - Services of each sub-system, constraints etc
- Interface design
  - Interface to other sub-system or outside environment
- Component design
  - Services allocated to components and their interfaces designed
- Data structure design
- Algorithm design

The Software Design Process



Levels of Software Design

- Architectural design (high-level design)
  - **Architectural design** represents the most abstract level of the system which involves identification of main modules and their interconnections
  - architecture - the overall structure, main modules and their connections
  - addresses the main non-functional requirements (e.g., reliability, performance)
  - hard to change
  - a bad architecture can not be saved by good construction methods!

- Detailed design (low-level design)
  - **Detailed Design** defines logical structure of each module and their interfaces to communicate with other modules.
  - detailed enough to be implemented in the programming language

Design vs. Architecture

Architecture

- Software architecture is the **high-level** structure of a software system, the way it is organized into modules, components, their relationships (structure), and how they work together and so on.
- Software architecture focuses on the **structure and interactions** of the software's individual components but does not get into much detail about how different components work when each is taken alone.

Design

- Software design is the process of planning and creating a **blueprint** for a software application. The goal of software design is to produce a software system that is both effective and efficient.
- Design, on the other hand, concentrates on the implementation **details of individual programs**, often delving into considerable depth. It centers on the creation of algorithms and how to store data.

The Principles of Software Design

There is a set of general laws of design that characterize the nature of design

1. The Principle of Totality
2. The Principle of Time
3. The Principle of Value
4. The Principle of Resources
5. The Principle of Synthesis
6. The Principle of Iteration
7. The Principle of Change
8. The Principle of Relationships
9. The Principle of Competence
10. The Principle of Service

General Principles of software design

- The design process should not suffer from "tunnel vision"
- The design should be traceable to the analysis model
- The structure of the software design should mimic the structure of the problem domain.
- The design should be structured to accommodate change.
- Design is not coding, coding is not design.
- The design should be reviewed to minimize conceptual (semantic) errors.

The Principles of Software Design

1. The Principle of Totality:
   - All design requirements are **always interrelated** and must always be treated as such throughout a design task
     - Requirement conflicts
     - Requirement prioritization
     - Design decisions

2. The Principle of Time:
   - The features and characteristics of **all products change as time passes.**
     - User friendliness
     - Resource intensive

The Principles of Software Design

3. The Principle of Value:
   - The characteristics of all products have **different relative values** depending upon the different circumstances and times in which they may be used.
     - **Changes with time**
     - Changes with circumstances
     - Adaptability for a wide range of user types should be considered.

4. The Principle of Resources:
   - The design, manufacture and life of all products and systems depend upon the **materials, tools and skills** upon which we can call.
     - development tools,
     - run time support systems,
     - human resource
     - application domain-specific tools and equipment,

The Principles of Software Design

5. The Principle of Synthesis:
   - All features of a product must **combine to satisfy all** the characteristics we expect it to possess with an acceptable relative importance for as long as we wish, bearing in mind the resources available to make and use it.
     - trade-offs between **desirable features and functions**

6. The Principle of Iteration:
   - Design requires **processes of evaluation** that begin with the first intentions to explore the need for a product or system.
     - designs have to be changed to correct errors and to improve quality.

The Principles of Software Design

7. The Principle of Change:

- Design is a process of change, an activity undertaken not only to **meet changing circumstance**, but also to bring about changes to those circumstances by the nature of the product it creates.
  - The design of a software system must take into consideration how it changes the way that we will work and live as the consequence of using the system.

8. The Principle of Relationships:

- Design work cannot be undertaken effectively without established **working relationships with all stakeholders**
  - *Customers, Users, System administrator, Project managers, Developers, Requirements analysts, Designers, Programmers, Testers*

The Principles of Software Design

9. The Principle of Competence:

- Design competence is the ability to create a synthesis of features that achieves all desired characteristics in terms of their required life and relative value, using available effective information
  - **competence of the designer.**

10. The Principle of Service:

- *Design* ***must satisfy everybody****, and not just those* for whom its products are directly intended.
  - it must be easy to maintain,
  - easy to reuse,
  - easy to transport to other operation environments and to
  - be inter-operable to other software systems, etc.

Summary

- Introduction to CSE 301-SDA
- Introduction to design
- Importance of design
- Difference between design and architecture
- The Principles of Software Design

Q & A

Any Question?