Software Design and Architecture

[Principles of Detailed Design] – Chapter 05, L01

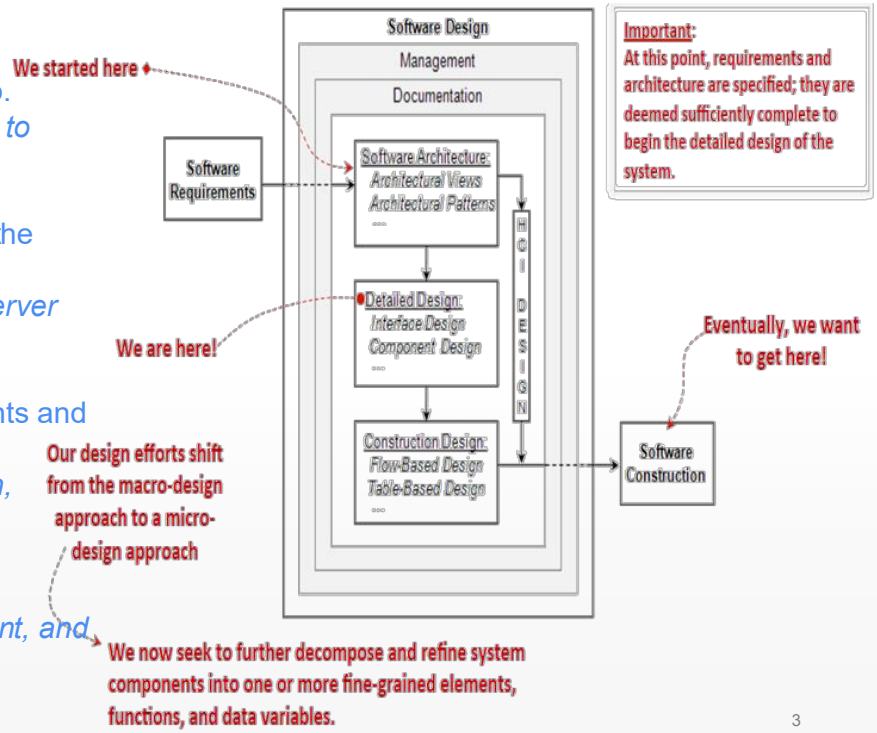
Lecture Outlines

- **Overview of Detailed Design**
  - ✓What is detailed design?
  - ✓Where does it fit?
- **Key Tasks in Detailed Design**
  - ✓Understanding architecture and requirements – (chapters 3 & 4)
  - ✓Creating detailed designs
  - ✓Evaluating detailed designs
  - ✓Documenting detailed designs
  - ✓Monitoring and controlling implementation

Lets think about, where we are …

Extra:

1. **Start (Requirements Phase):**
   Gather and document what the system must do.
   Example: In a banking app, users must be able to check balances or transfer money.
2. **Architecture (High-Level Design):**
   Define the overall structure and interactions of the system.
   Example: Decide the system will use a client-server model with a database, app, and security layer.
3. **Detailed Design (Refinement):**
   Break down architecture into smaller components and detailed functions.
   Example: For login, specify password validation, authentication, and session management.
4. **Goal (Construction Phase):**
   Start coding based on the detailed design.
   Example: Implement login, account management, and transfer features.



What is Detailed Design?

- **Detailed design** is:
  - ✓ The process of refining and expanding the preliminary design phase *software architecture* of a system or component to the extent that the design is sufficiently complete to be implemented.
- During ***Detailed Design*** designers go deep into each component to define its internal structure and behavioral capabilities, and the resulting design leads to natural and efficient construction of software.

Extra:

In this stage, designers focus on:

- ✓ **Internal structure**: Defining how each component works internally.
- ✓ **Behavior**: Specifying how each component behaves and reacts in different scenarios.

Extra:

1. Detailed Design (Refinement Phase)

•**What happens here:**
You don't write the actual code yet. Instead, you **specify in detail** *how* each part of the system will work.
•**Output:** Detailed specifications, algorithms, database schemas, interface definitions, class diagrams, etc.
•**Example:**

- Login requires: password validation rules (min length, special characters).
- Authentication: use hashing + salting.
- Session management: define session timeout, refresh tokens.

👉 Think of this as **blueprints before construction**.

2. Construction Phase (Implementation)

•**What happens here:**
Developers **translate the detailed design into actual code**.
•**Output:** Executable software (source code + builds).
•**Example:**

- Writing the actual `validatePassword()` function in Java/Python.
- Implementing the authentication API.
- Coding the session management module.

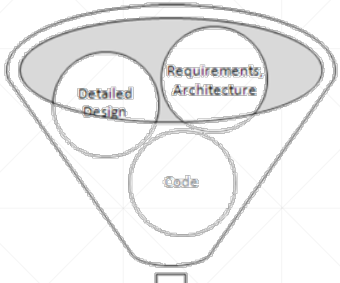
👉 Think of this as **building the house according to the blueprint**.

What is Detailed Design?

Detailed design is closely related to architecture and construction; therefore successful designers (during detailed design) are required to have or acquire full understanding of the system's requirements and architecture.

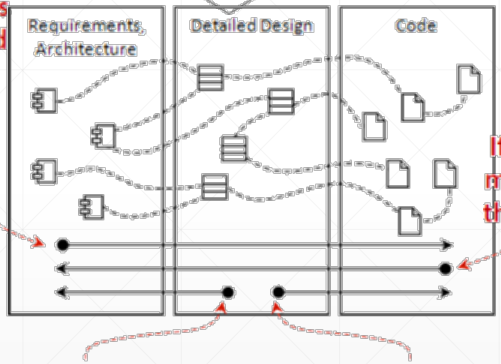
- ✓Just as architecture provides the bridge between requirements and design, detailed design provides the bridge between design and code.
- ✓They must also be proficient in particular design strategies (e.g., object-oriented), programming languages, and methods and processes for software quality control.

Designer's Mental Model During Detailed design



Important:
During detailed design, the use of industry-grade development tools are essential for modeling, code generation, compiling generated code, reverse engineering, software configuration management, etc.

If given requirements and architecture, detailed designers must move the project forward all the way to code



If given code, detailed designers must be able to reverse engineer the code to produce detailed and architectural designs.

When starting at detailed design, designers must be able to produce both code and architectural designs

Extra:

- **Detailed design** sits between **requirements/architecture** and **code**.
  - If you are given **requirements + architecture**, you must design details until you reach code.
  - If you are given **code**, you should be able to go backwards (reverse engineer) to recreate the detailed design and architecture.
- So detailed designers need to think in **both directions** → from design to code, and from code back to design.
- Tools (like UML modeling, code generators, compilers, and version control) are essential in this stage.

- **What is Reverse Engineering?**

Reverse engineering = **analyzing existing code** to figure out:

- what the system does,
- how it is structured,
- and how it maps back to design and requirements.

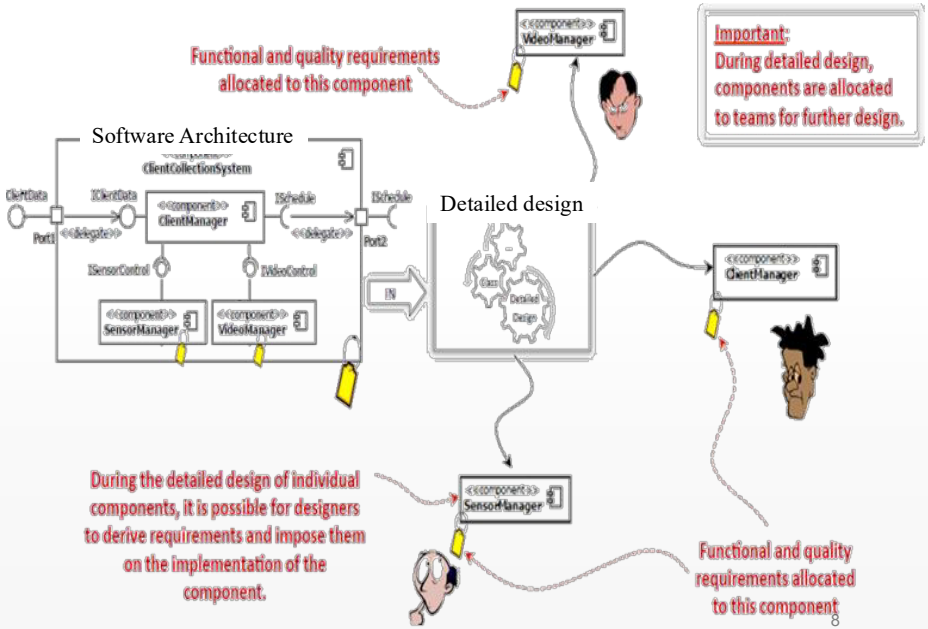
Key Tasks in Detailed Design

- The major tasks identified for carrying out the detailed design activity include:
  1. Understanding the architecture and requirements
  2. Creating detailed designs
  3. Evaluating detailed designs
  4. Documenting software design
  5. Monitoring and controlling implementation

1. Understanding Architecture and Requirements

➢Unlike the software architecture, were the complete set of requirements are evaluated and well understood, designers during detailed design activity focus on requirements allocated to their specific components.

Extra:
During **detailed design**, big components from architecture (e.g., **SensorManager, VideoManager, ClientManager**) are **assigned to teams**. Each team refines its component, adds detailed requirements (like validation or performance), and prepares it for **implementation (coding)**.



2. Creating Detailed Designs

- After the architecture and requirements for assigned components are well understood, the detailed design of software components can begin.
  (Detailed design consist of both **structural and behavioral designs)**.

- When creating detailed designs, focus is placed on the following:
  - ✓ **Interface Design** - Internal & External
  - ✓ **Graphical User Interface** (GUI) Design
  - ✓ **Internal Component** Design
    - ✓ Structural
    - ✓ Behavioral
  - ✓ **Data Design** ~ *Database; data dictionary*

3. Evaluating Detailed Designs

- The most popular technique for evaluating detailed designs involves ***Technical Reviews***. When conducting technical reviews, keep in mind the following:
  - ✓ Send a review notice with enough time for others to have appropriate time to thoroughly review the design.
  - ✓ Include a technical expert in the review team, as well as stakeholders of your design.
  - ✓ Include a member of the software quality assurance or testing team in the review.
  - ✓ During the review, focus on the important aspects of your designs; those that show how your design helps meet functional and non-functional requirements.
  - ✓ Document the review process. (Make sure that any action items generated during the review are captured and assigned for processing)

Extra:
Technical Reviews (to evaluate detailed design):

1. **Send review notice early (give time to prepare).**
2. **Include experts, stakeholders, and QA/test team.**
3. **Focus on how design meets functional & non-functional requirements.**
4. **Document review and assign action items.**

Example: When reviewing the design of an e-commerce checkout system, focus on how the design ensures secure payments (non-functional) and smooth order processing (functional).

4. Documenting Detailed Designs

- Documentation of a project's software design is mostly captured in the **software design document (SDD)**, also known as software design description. The SDD is used widely throughout the development of the software. (Used by programmers, testers, maintainers, systems integrators, etc.)

- Other forms of documentation include:
  - ✓ **Interface Control Document**
    - A document describes system's interfaces and how the components of the system communicate.
  - ✓ **Version Control Document**
    - Contains information about what is included in a software release, including different files, scripts and executable. Different versions of the design depend on specific software release

Extra:

Software Design Documentation

- **Software Design Document (SDD):** Main document describing the design. Used by programmers, testers, maintainers, and integrators.
- **Interface Control Document:** Defines system interfaces and how components communicate.
- **Version Control Document:** Lists what is included in each software release (files, scripts, executables).

4. Documenting Detailed Designs

- The sections of the SDD and sample table of contents:

Section	Description
Date of issue and status	Date of issue is the day on which the SDD has been formally released. Every time the SDD is updated and formally released, there should be a new date of issue.
Scope	Scope provides a high level overview of the intended purpose of the software. It sets a limit as to what the SDD will describe and defines the objectives of the software.
Issuing organization	Issuing organization is the company which produced the SDD.
Authorship	Authorship pertains to who wrote the SDD and certain copyright information.
References	References provide a list of all applicable documents that are referred to within the SDD. If there is a certain technology that is used within the design, it is important to refer to the corresponding documentation on that technology, so it may be referenced. When reading the referenced documents, stakeholders may uncover inconsistencies in how the technology should be used and how it is used in the software design.
Context	Description of the context of the SDD.
Body	Body is the main section of the SDD where the design is documented. This is where stakeholders look to understand the software and how it is to be constructed.
Summary	
Glossary	A glossary provides definitions for all software related terms and acronyms used in the SDD.
Change history	Change history is a brief description of the items added to, deleted from, or changed within the SDD.

5. Managing Implementation

- Detailed design **synchronicity** is concerned with the degree of how well detailed designs adhere to the software architecture and how well software code adheres to the detailed design.
  - ✓ Low degree of synchronicity points to a flaw in the process and can lead to software project failure.
- Particular attention needs to be paid when projects enter the maintenance phase or when new engineers are brought into the project.
- Processes must be in place to ensure that overall synchronicity is high

Extra:

Detailed Design Synchronicity

- Ensures detailed design follows architecture and code follows design.
- Low synchronicity → flaws, risk of project failure.
- Critical during maintenance or when new engineers join.
- Processes must keep synchronicity high.

- Example: If the architecture requires **encrypted login**, but the detailed design forgets encryption and the code just stores plain passwords → synchronicity is broken.

Summary

- **Overview of Detailed Design**
  - ✓ What is detailed design?
  - ✓ Where does it fit?
- **Key Tasks in Detailed Design**
  - ✓ Understanding architecture and requirements – (chapters 3 & 4)
  - ✓ Creating detailed designs
  - ✓ Evaluating detailed designs
  - ✓ Documenting detailed designs
  - ✓ Monitoring and controlling implementation
- **Next … Design Patterns**