

Gabriel Q. Escobido

BSCoE – 2A

## **SOFTWARE DESIGN**

### **Laboratory Exercise No. 1 CH1**

#### **Title: Introduction to Software Design, History, and Overview**

##### **Brief Introduction**

Software design is a critical phase in the software development lifecycle that translates requirements into a blueprint for constructing a system. This exercise introduces the history of software design and explains why it plays a significant role in modern software development.

##### **Objectives**

Understand the historical evolution of software design.

Learn the key phases of software design.

Explore why software design is critical in producing reliable systems.

##### **Detailed Discussion**

##### **History and Evolution**

The evolution of software design has its roots in the earliest days of computing. During the 1950s and 1960s, programming was often done in machine code or assembly language, with little thought given to structured design. Over time, as software complexity grew, structured programming was introduced in the 1970s, emphasizing modularity and readability. This led to the emergence of methodologies like the Waterfall Model, which provided a sequential approach to software development. In the 1990s, Agile methodologies revolutionized software design by promoting iterative development and continuous feedback, enabling teams to adapt to changing requirements rapidly. Today, modern design practices integrate DevOps principles, focusing on collaboration, automation, and scalability.

##### **Importance of Software Design**

Software design is critical for several reasons:

**Scalability:** Proper design ensures that the software can handle growth in users, data, or functionality without significant rework.

**Maintainability:** A well-designed system is easier to understand, debug, and enhance, reducing long-term costs.

**Efficiency:** By optimizing resources and processes, good design improves performance and user satisfaction.

**Reliability:** Thoughtful design minimizes bugs and errors, ensuring the software meets user expectations.

## Phases of Software Design

The design phase is typically divided into three key stages:

**Requirement Analysis:** Understanding the problem and identifying the objectives and constraints.

**High-Level Design (HLD):** Outlining the system's architecture, including major modules, their interactions, and external interfaces. HLD often involves tools like UML diagrams and architectural blueprints.

**Low-Level Design (LLD):** Providing a detailed specification of individual modules and their internal logic, often represented using pseudocode or flowcharts.

Phase	Description
Requirement Analysis	Understanding the problem and objectives.
HLD	Outlining the system's architecture, including modules and interactions.
LLD	Specifying detailed logic for modules and components.

## Materials

Computer system with VS Code IDE installed.

Python (latest version).

## Time Frame

1 hour

## Procedure

Install VS Code and Python on your system.

Research the history of software design methodologies (e.g., structured programming, Agile).

Use the table format provided below to list the differences between HLD and LLD.

Aspect	High-Level Design (HLD)	Low-Level Design (LLD)
Focus	Overall system architecture	Detailed logic of each component
Tools	Architecture diagrams, UML	Pseudocode, flowcharts

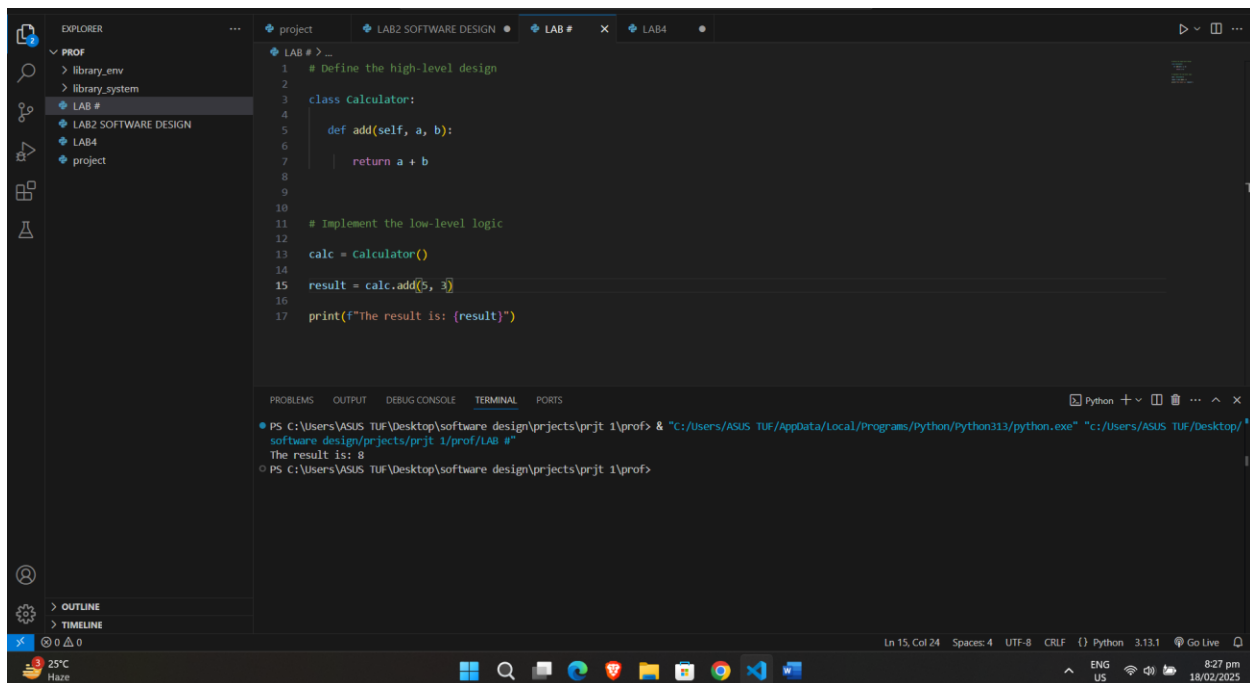
Create a small Python program that demonstrates the design process.

Example Code:

```
# Define the high-level design
class Calculator:
    def add(self, a, b):
        return a + b

# Implement the low-level logic
calc = Calculator()
result = calc.add(5, 3)
print(f"The result is: {result}")
```

**Results**

A screenshot of the Visual Studio Code (VS Code) editor interface. The Explorer sidebar on the left shows a project structure with folders like 'library\_env', 'library\_system', and 'LAB #'. The main editor window displays a Python file named 'LAB #.py'. The code defines a 'Calculator' class with an 'add' method and implements low-level logic to calculate the sum of 5 and 3, printing the result. The output window at the bottom shows the command prompt execution: 'PS C:\Users\ASUS TUF\Desktop\software design\projects\prjt 1\prof> & "C:\Users\ASUS TUF\AppData\Local\Programs\Python\Python313\python.exe" "C:\Users\ASUS TUF\Desktop\software design\projects\prjt 1\prof\LAB #.py"', resulting in 'The result is: 8'. The status bar at the bottom indicates the current line and column (Ln 15, Col 24) and the Python version (3.13.1).

Provide screenshots of your table and Python program output.

## Follow-Up Questions

1. What are the differences between HLD and LLD?

HLD (High-Level Design) focuses on the overall system architecture, providing a broad view of components and their interactions, while LLD (Low-Level Design) dives into the detailed design, specifying the implementation of individual components, data structures, and algorithms.

2. Why is software design critical in modern development?

Software design is critical in modern development because it ensures scalability, maintainability, and efficiency, preventing costly rework and enabling teams to build robust, flexible systems.

3. Can you identify examples where poor design has led to software failure?

Examples of poor design leading to software failure include the *Knight Capital Group* incident, where a software bug in their trading algorithm caused a \$440 million loss, and *Healthcare.gov*, where poor system architecture and inadequate testing led to massive launch failures.

Summarize your understanding of software design, HLD, and LLD.

## Summary

Software design transforms requirements into actionable plans. By understanding its phases and importance, students can create efficient systems.

### Conclusion

Good design is the foundation of reliable, maintainable software. This exercise highlights the critical role of both HLD and LLD in achieving that.