Segales, Ace Niño B.

BSCpE 2A

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:

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

2. **Maintainability**: A well-designed system is easier to understand, debug, and enhance, reducing long-term costs.
3. **Efficiency**: By optimizing resources and processes, good design improves performance and user satisfaction.
4. **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:

1. **Requirement Analysis**: Understanding the problem and identifying the objectives and constraints.
2. **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.
3. **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

1. Install VS Code and Python on your system.
2. Research the history of software design methodologies (e.g., structured programming, Agile).
3. 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:

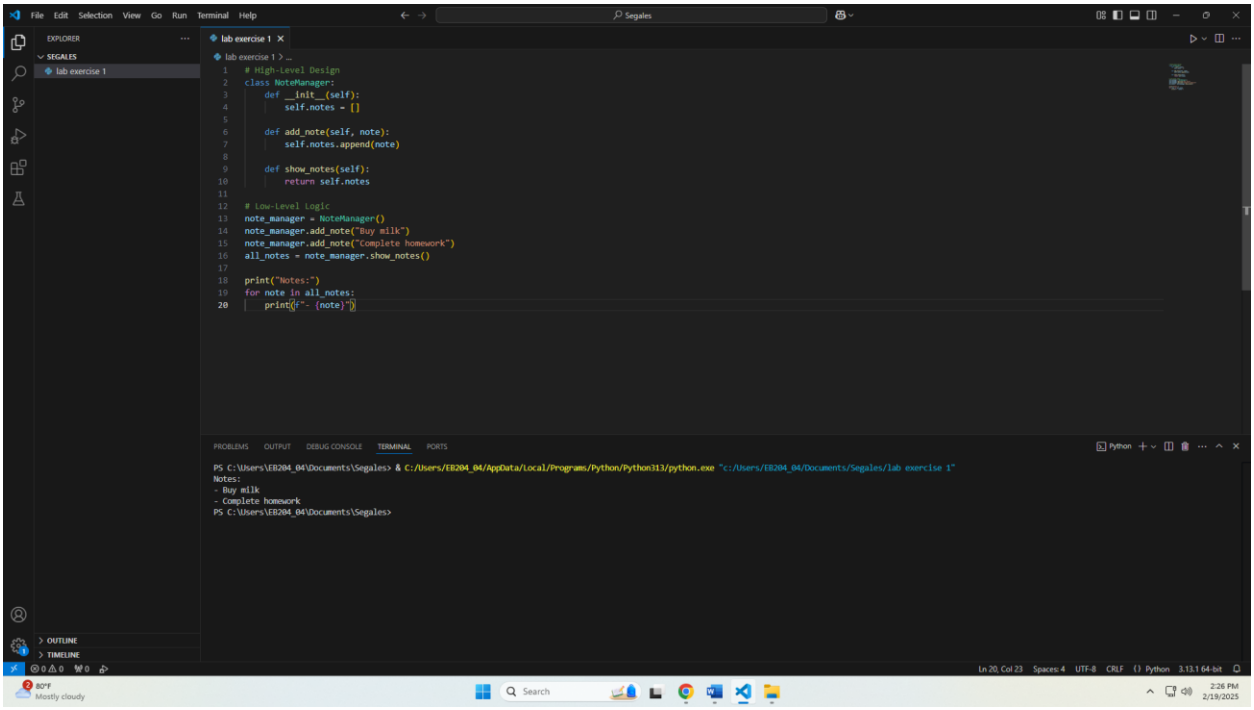
```
# Define the high-level design
class Calculator:
  def add(self, a, b):
    return a + b

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

Results

Provide screenshots of your table and Python program output.

Aspect	**High-Level Design (HLD**	**Low-Level Design (LLD)**
Focus	**NoteManager** class manages a list of notes with methods to add and display them.	An instance of **NoteManager** is created, notes are added, and then all notes are displayed.

Tools	Class definition and method structure	Instantiating the class and calling method
abstraction	High(Focuses on what the system does)	Low (focuses how the system does it
Example	class NoteManager: def __init__(self):	note_manager = NoteManager()



Follow-Up Questions

1. What are the differences between HLD and LLD?

High-Level Design (HLD) gives a comprehensive view of the system architecture and its components, emphasizing how they relate to and interact with each other. In contrast, Low-Level Design (LLD) provides in-depth specifications for each module, detailing implementation aspects, data structures, and algorithms, which serve as a roadmap for developers throughout the coding process.

2. Why is software design critical in modern development?

Software design plays a vital role in contemporary development as it offers clarity and structure, guarantees scalability and maintainability, promotes collaboration, mitigates

risks, optimizes performance, improves user experience, integrates compliance and security, enables adaptability to change, and ultimately contributes to cost efficiency.

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

Examples of poor software design that led to failure include the troubled launch of Healthcare.gov, which suffered from inadequate architecture, Knight Capital's trading glitch that resulted in a $440 million loss, and Toyota's unintended acceleration issues caused by faulty electronic throttle control software. Other instances are Microsoft Windows Vista's user interface and compatibility problems, Target's data breach due to weak security design, the loss of NASA's Mars Climate Orbiter because of inconsistent measurement systems, and the Boeing 737 Max crashes associated with flawed MCAS software design.

Findings

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.