Daryl D. Silva

BSCpE-2B2

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.

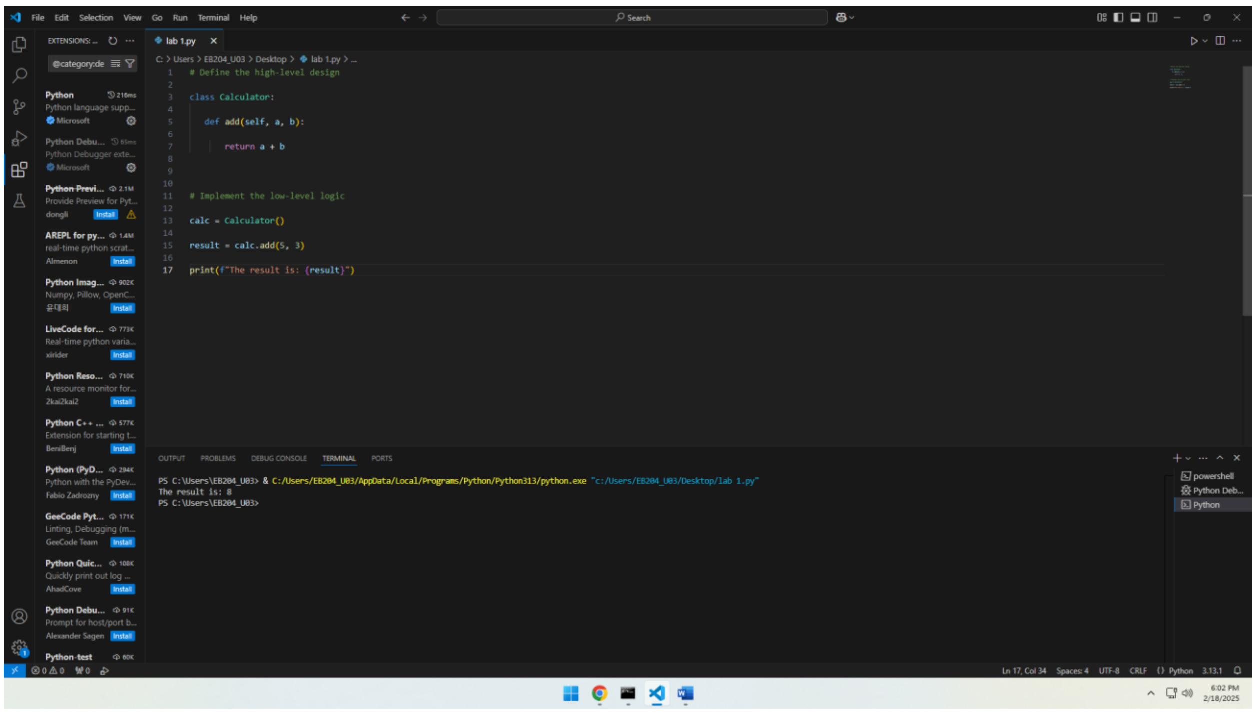
Create a small Python program that demonstrates the design process.

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.



```
# 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}")
```

```
PS C:\Users\EB204_U03> & C:/Users/EB204_U03/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/EB204_U03/Desktop/lab 1.py"
The result is: 8
PS C:\Users\EB204_U03>
```

Follow-Up Questions

1. What are the differences between HLD and LLD?

Answer: **HLD (High-Level Design)**:

- Focuses on system architecture and components.
- Provides an overview of the system's structure.
- Includes modules, data flow, and external interfaces.
- Designed for stakeholders, not developers.

LLD (Low-Level Design):

- Details the implementation of individual components.
- Specifies algorithms, data structures, and interface logic.
- Created for developers to guide coding.
- Focuses on internal workings and system behavior.

2. Why is software design critical in modern development?

   Answer:

   - Ensures maintainability, scalability, and performance.
   - Reduces development time and costs by preventing rework.
   - Provides clear guidelines for developers, improving collaboration.
   - Helps identify and address potential issues early, improving software quality.

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

Findings

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

Answer: Software Design is the process of planning and structuring software to ensure it meets user requirements, is maintainable, and performs well. It involves creating blueprints for both high-level architecture and detailed implementations.

HLD (High-Level Design) focuses on the overall architecture, identifying major components, their interactions, and system flow. It's more abstract, providing a broad overview for stakeholders.

LLD (Low-Level Design) dives into the specifics of each component, outlining detailed implementation aspects such as algorithms, data structures, and code logic. It's meant for developers to guide actual coding and implementation.

Summary

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

Answer: **Software design** bridges the gap between requirements and implementation, transforming abstract needs into structured plans that guide development. By grasping its phases—**HLD** for overarching architecture and **LLD** for detailed implementation—students can design systems that are efficient, scalable, and maintainable. This understanding helps them craft robust solutions, ensuring that software meets both functional and non-functional requirements effectively.

Conclusion

Software Design is a fundamental process that transforms user requirements into structured, actionable plans for building software systems. From its early roots in the 1960s, where the focus was on basic functionality, to the complex, scalable systems we build today, the evolution of software design has been driven by the need for efficiency, flexibility, and maintainability. Understanding the phases of design—**High-Level Design** (HLD) and **Low-Level Design** (LLD)—empowers developers to create systems that are not only functional but also adaptable to future changes. By mastering the principles of software design, developers can ensure their systems are optimized for both current needs and future growth, fostering innovation and long-term success.