What is Low Level Design or LLD

LLD, or Low-Level Design, is a phase in the software development process where detailed system components and their interactions are specified. It involves converting the high-level design into a more detailed blueprint, addressing specific algorithms, data structures, and interfaces. LLD serves as a guide for developers during coding, ensuring the accurate and efficient implementation of the system’s functionality. LLD describes class diagrams with the help of methods and relations between classes and program specs.

Low-level designing is also known as object-level designing or micro-level or detailed designing.

In the world of software development, low-level design (LLD) refers to the process of breaking down a high-level overview of a system into specific, detailed components and their interactions. It's the blueprint that bridges the gap between abstract ideas and practical implementation. Here's a breakdown of what LLD involves:

Focus:

Detailing components: While high-level design (HLD) focuses on the big picture, LLD dives deep into individual modules, functions, and algorithms. It defines their responsibilities, interactions, and internal workings.

Data structures and algorithms: LLD specifies the data structures to be used for storing and manipulating information within each component. It also outlines the algorithms used to implement specific functionalities.

Interfaces and APIs: LLD defines how different components will communicate with each other, including API specifications and data exchange formats.

Benefits:

Clearer implementation: A well-defined LLD acts as a roadmap for developers, providing a clear understanding of what needs to be built and how. This reduces confusion and rework during coding.

Improved efficiency: LLD allows for optimization at the component level, ensuring efficient data structures and algorithms are chosen for each task.

Better testing: By clearly defining individual components and their interactions, LLD facilitates a more focused and effective testing process.

Tools and techniques:

UML diagrams: These visual representations, like class diagrams and sequence diagrams, help depict the relationships between components and their data flow.

Pseudocode: This informal language describes algorithms in a human-readable format, enabling clear communication of logic without writing actual code.

Documentation: Detailed specifications for each component, including data structures, interfaces, and expected behavior, are crucial for code consistency and future reference.

Overall, LLD is a crucial step in the software development process. It bridges the gap between abstract ideas and concrete implementation, ensuring smooth development, efficient system performance, and easier maintainability in the long run.

What is High Level Design?

High-level design or HLD refers to the overall system, a design that consists description of the system architecture and design and is a generic system design that includes:

- System architecture
- Database design
- Brief description of systems, services, platforms, and relationships among modules.

High-level design or HLD is also known as macro level designing.

High-level design (HLD) is the big-picture roadmap for a system, product, service, or process. It focuses on the "what" and "why" rather than the intricate details of "how". Think of it as the architectural sketch before the detailed blueprints. Here's what HLD encompasses:

Key Features:

System architecture: This outlines the main components of the system and their overall interaction. It's like a high-level diagram showing the building blocks without getting bogged down in bricks and mortar.

Data flows: HLD describes how data moves through the system, indicating its origin, transformation, and destination. Imagine the plumbing layout without worrying about the specific pipe dimensions.

Module relationships: It defines how different components communicate and rely on each other. This helps ensure all parts work together seamlessly.

Non-technical overview: While some technical vocabulary might be used, HLD should be largely understandable to stakeholders even without deep technical knowledge. Imagine explaining your project to an investor, focusing on the core functionalities and benefits.

Benefits of HLD:

Early communication: It facilitates discussions and feedback from stakeholders at an early stage, shaping the direction of the project before significant resources are invested.

Reduced rework: By clarifying the overall vision, HLD minimizes the need for later changes and saves time and effort during development.

Clear roadmap: It provides a guiding light for developers, ensuring they understand the goals and constraints of the system.

Consistent direction: HLD serves as a reference point for everyone involved, fostering collaboration and preventing project drift.

Examples of HLD deliverables:

Architectural diagrams: Visual representations of the system's components and their connections.

Flowcharts: Depicting the data flow through the system.

Feature breakdowns: Outlining the functionalities and their relationships.

High-level user stories: Describing the user experience from a broad perspective.

To sum it up, HLD is the foundation on which successful projects are built. It ensures everyone is aligned with the vision and goals, paving the way for smooth development and a satisfying final product.

Is C++ a Low Level Language OR High Level Language?

C++ itself is neither a high-level design (HLD) nor a low-level design (LLD) language. It's a general-purpose, programming language that can be used for both HLD and LLD purposes.

HLD: C++ can be used to create high-level abstractions like frameworks, libraries, and APIs. These components focus on the "what" and "why" of a system, without delving into the specific implementation details. This makes it suitable for outlining the overall architecture and functionality of a system at a high level.

LLD: C++ can also be used to write low-level code that directly interacts with hardware, memory, and other system resources. This requires detailed knowledge of the underlying hardware and operating system, making it suitable for implementing the intricate workings of a system at a granular level.

So, whether C++ is used for HLD or LLD depends entirely on the context and the programmer's intent. It's a versatile language that can cater to both abstraction and detailed implementation, making it a powerful tool for developers of all levels.