LLD vs HLD

In software engineering, building a complex system is like constructing a city.

You wouldn't start by laying bricks for a single house without a city plan. You first need to decide where the residential areas, commercial zones, power grids, and roads will go.

This city plan is your High-Level Design (HLD).

Once the city plan is approved, an architect takes a single plot of land in a residential zone and designs the detailed blueprint for a house specifying the number of rooms, the plumbing, the electrical wiring, and the materials to be used.

This detailed house blueprint is your Low-Level Design (LLD).

Both are essential, but they operate at different levels of abstraction and serve different purposes.

In this chapter, we will take a deeper look at their differences.

What is High-Level Design (HLD)?
High-Level Design (HLD) answers the question: “How should the system be structured and how will different modules interact?”

The primary focus of HLD is on the "what," not the "how." It answers questions like:

What are the major components or microservices? (e.g., User Service, Payment Service, Notification Service, Product Catalog).
How will these components communicate? (e.g., via REST APIs, gRPC, or a message queue like RabbitMQ or Kafka).
What technology stack will be used? (e.g., Java vs. Python, SQL vs. NoSQL database).
How will the system handle scalability, reliability, and availability? (e.g., using load balancers, database replication, CDNs).
What are the third-party integrations? (e.g., Stripe for payments, Twilio for SMS).
The output of HLD is a set of architectural diagrams, data flow diagrams, and technology choices that define the system's skeleton.

Example: HLD of a Ride-Hailing App

Services: Passenger Service, Driver Service, Matching Service, Billing Service.
Communication: Matching Service uses a message queue to broadcast ride requests. Passenger and Driver services communicate via WebSockets for real-time location updates.
Database: A NoSQL database for location data and a relational SQL database for user and billing information.

Infrastructure: Load balancers to distribute traffic, with services deployed as containers on Kubernetes for scalability.
What is Low-Level Design (LLD)?
Low-Level Design zooms in on a single component or module defined in the HLD. It's where the architectural abstractions are translated into concrete, implementable logic. LLD is the detailed blueprint that developers use to write the actual code.

The focus of LLD is on the "how." For a single module, it answers questions like:

What are the specific classes, and what are their responsibilities?
What are the attributes and methods of each class?
How do these classes relate to each other (inheritance, composition)?
What design patterns are most suitable (e.g., Factory, Singleton, Strategy)?
What are the specific method signatures, including parameters, return types, and exceptions?
Example: LLD of the Billing Service from the Ride-Hailing App

Classes: Ride, Invoice, PaymentStrategy, CreditCardPayment, WalletPayment.
Interfaces: An interface IPaymentStrategy with a method processPayment(amount). CreditCardPayment and WalletPayment would implement this interface.
Relationships: The Invoice class "has-a" Ride object (Composition).
Design Pattern: The Strategy Pattern is used to allow the user to switch between different payment methods seamlessly.
Key Differences: HLD vs. LLD at a Glance
HLD vs LLD

Aspect	High-Level Design (HLD)	Low-Level Design (LLD)
Focus	What components exist	How each component is built
Audience	Architects, stakeholders	Engineers, developers
Abstraction	System-level	Module/class-level
Artifacts	System architecture diagrams, tech stack choices	Class diagrams, interaction diagrams, method definitions
Example	"We'll use a microservices architecture with services for users, orders, and payments"	"The OrderService will use a PaymentProcessor interface with two implementations: StripeProcessor and RazorpayProcessor"