📌 Software Development Models

Model	Description
Waterfall	A **linear, sequential** approach where each phase (Requirement → Design → Implementation → Testing → Deployment) must be completed before moving to the next. Best for projects with **stable requirements** and minimal changes.
V-Model (Verification & Validation Model)	An **extension of Waterfall** where each development phase has a corresponding **testing phase**. Often used in **safety-critical industries** like healthcare, aerospace, and finance.
Unified Process (UP)	A **structured, iterative framework** where development progresses in **phases** (Inception, Elaboration, Construction, Transition). Balances flexibility with a **well-defined structure**. Suitable for large enterprise systems.
Spiral Model	Focuses on **risk assessment & iteration**. Each cycle consists of **planning, risk analysis, development, and evaluation**. Best for **high-risk projects** where requirements evolve.
Prototyping	A **proof-of-concept** model where a **working prototype** is built before full-scale development. Helps gather feedback early but can lead to **scope creep** if overused.
Agile	A **flexible, iterative approach** that emphasizes **customer collaboration, working software, and responsiveness to change**. Ideal for dynamic projects where requirements evolve frequently.
RAD (Rapid Application Development)	A **fast-paced development model** that emphasizes **prototyping, iterative development, and quick user feedback**. Best for projects needing **rapid delivery with evolving requirements**.

📌 Agile Methodologies

Method	Description

Scrum	A structured Agile framework where work is divided into **fixed-length Sprints (1–4 weeks)**. Teams hold **Daily Stand-ups**, maintain a **Product Backlog**, and conduct **Sprint Planning, Reviews, and Retrospectives**. Best for **team-based product development**.
Kanban	A **visual workflow management method** where tasks move through **a Kanban board (Backlog → In Progress → Testing → Done)**. No fixed iterations, work is **pulled** as capacity allows. Used in **DevOps, maintenance, and support teams**.
Scrumban	A **hybrid of Scrum and Kanban**, combining Scrum's **structured planning** with Kanban's **flexibility**. Ideal for **long-term projects** with changing priorities.
Extreme Programming (XP)	Focuses on **high-quality coding**, **continuous feedback**, **pair programming**, **test-driven development (TDD)**, and **small, frequent releases**. Best for **small, highly skilled teams needing top-tier code quality**.
Lean Development	Derived from **Toyota's Lean Manufacturing**, it emphasizes **waste elimination, fast delivery, and continuous learning**. Helps **optimize software processes** and improve efficiency.

📌 Software Estimation Techniques

Estimation Method	**Description**
COCOMO (Constructive Cost Model)	A **mathematical model** that estimates development effort based on **Lines of Code (LOC)** and project complexity. Best for **large, structured projects** in traditional environments.
Function Point Analysis (FPA)	Measures software **size and complexity** based on **user functions** instead of LOC. Useful for **business applications and cross-platform projects**.
Planning Poker	An Agile estimation technique where **team members independently assign effort estimates** using **Fibonacci numbers (1, 2, 3, 5, 8...)**, then discuss and reach a consensus. Helps **reduce bias in team estimation**.
T-Shirt Sizing	A quick estimation method where tasks are categorized as **Small (S), Medium (M), Large (L), Extra Large (XL)**

	based on complexity. Useful for **early-stage project estimation**.
Wideband Delphi	A technique where **experts independently estimate effort**, results are aggregated anonymously, then revised in multiple rounds until consensus is reached. Best for **complex projects requiring expert judgment**.

📌 Testing Approaches

Testing Type	Description
Unit Testing	Tests **individual components or functions** in isolation. Helps detect bugs early and is commonly used in **Test-Driven Development (TDD)**.
Integration Testing	Ensures that multiple **modules work together correctly**. Helps detect **interface errors between components**.
System Testing	Tests the **entire system** to verify if it meets the requirements (functional and non-functional). Focuses on **end-to-end validation**.
Acceptance Testing	Performed by **clients/users** to determine if the system **meets business needs** before deployment.
Blackbox Testing	Tests the software **without looking at internal code structure**. Focuses only on **input-output behavior**.
Whitebox Testing	Examines the **internal logic and code paths** to ensure correctness. Used by **developers for debugging and optimization**.
Model-Based Testing (MBT)	Uses **system models (e.g., flowcharts, state machines) to generate test cases automatically**. Best for **complex systems with predefined logic**.

📌 Test Automation & Mocking

Technique	Description
Test Doubles (Mocking, Stubbing, Fakes, Dummies)	Fake objects that **replace real dependencies** during testing. Used to isolate **System Under Test (SUT)**.
Robot Framework	A **keyword-driven test automation tool** for **Acceptance Test-Driven Development (ATDD)**.

Supports multiple languages (Python, Java, Selenium).

📌 Lean Development: Waste Elimination Strategies

Lean Principle	Description
Eliminate Waste	Remove **non-value activities** like unnecessary features, rework, delays, and excessive documentation.
Amplify Learning	Encourage **continuous feedback** through **prototypes, testing, and retrospectives**.
Decide as Late as Possible	Delay major decisions until **all relevant data is available**.
Deliver as Fast as Possible	Focus on **speed to market** with **short iterations & quick feedback loops**.
Empower the Team	Give **decision-making power** to developers rather than relying on strict management.
Build Integrity In	Ensure **long-term maintainability** by focusing on **quality, refactoring, and continuous integration**.
See the Whole	Consider **the entire system**, not just individual parts, to avoid inefficiencies.

📌 When to Use Scrum vs. Kanban vs. XP vs. Lean

Method	Description
Scrum	Best for **structured teams** working in **fixed-length iterations (Sprints)**. Provides **predictability & clear roles**.
Kanban	Ideal for **continuous work environments** like **DevOps, support teams, and maintenance**. No fixed iterations.
XP (Extreme Programming)	Best for **high-quality software development with test-driven development (TDD), pair programming, and continuous feedback**.
Lean	Focuses on **maximizing efficiency, reducing waste, and delivering fast**. Best for **optimizing workflows**.

📌 Final Takeaways

1️⃣ **Use Waterfall/V-Model/UP** → When requirements are **fixed, structured, & documented**.
2️⃣ **Use Agile (Scrum, Kanban, XP)** → When requirements **change frequently & fast iterations are needed**.
3️⃣ **Use COCOMO/FPA for estimation** → When project scope is **predictable**. **Use Planning Poker/T-Shirt Sizing** for Agile.
4️⃣ **Use Lean** → When you need to **maximize efficiency & remove bottlenecks**.
5️⃣ **Use XP/TDD** → When **high-quality code & test coverage are crucial**.
6️⃣ **Use Kanban** → When **tasks flow continuously (support, bug fixing, operations, DevOps)**.
7️⃣ **Use Scrum** → When work needs **structured planning & sprint-based delivery**.

This breakdown provides **clear definitions for each method** so you can **compare them easily and pick the right one** for your project! 🚀