1. Software Design in 2025: Process and Artifacts

As a software engineer who's built everything from web apps to enterprise systems, software design in 2025 feels like sketching a house before pouring concrete—it's all about planning smart so you don't waste time fixing big mistakes later. It's evolved with AI tools making it faster and more collaborative, but the heart is still turning user needs into a solid blueprint.

The Process

Design isn't a one-and-done; it's an iterative loop that fits into the bigger software lifecycle. In 2025, we lean on cloud tools and AI to speed it up—expect 2-4 weeks for a full design phase instead of months. Here's the simple flow:

- Gather Requirements: Talk to users (or simulate with AI chats) to nail down what the app needs to do. Write user stories like "As a student, I want to track grades so I can stay on top of classes."
- Brainstorm and Model: Sketch high-level ideas. Use tools like Miro or Figma AI to map user flows and system parts. Spot risks early, like "What if the server crashes during peak use?"
- Detail and Prototype: Dive into specifics—how screens look, how data flows. Build quick clickable prototypes to test with folks. AI helps generate initial code or layouts from your notes.
- Review and Refine: Get feedback from the team, run security scans, and iterate. Version everything in Git so you can rollback if needed.
- Handoff: Package it up for devs to build, with clear notes on why choices were made.

It's flexible—jump back if new info pops up—and focuses on scalability, security, and user-friendliness from day one. Pro tip: Always design for mobile-first in 2025; most users are on phones.

Key Artifacts

These are the "deliverables" that make the process tangible. For a project like building a grade-tracking app (relevant to something like USeP's BSIT coursework), we create stuff that's easy to share and update. Here are three essentials:

Artifact	Description	Why Relevant to the Project
UML Diagram	A visual map (like class or sequence diagrams) showing how classes/objects interact—think boxes and arrows for data flow.	Helps model the app's core logic, e.g., how "Student" class links to "Grade" database. Keeps the team synced without endless meetings.
Architecture Decision Record (ADR)	A simple Markdown doc logging choices like "We picked React for the frontend because it's fast and has great AI plugins." Includes pros/cons and trade-offs.	Tracks why we went microservices over monolith—super useful for project reviews or future maintenance in a team setting.
Mock System Overview	A high-level dashboard mockup (in tools like Lucidchart) showing the whole system: UI, backend, database, and APIs at a glance.	Gives a bird's-eye view of the grade app, e.g., "User logs in → API calls database → Dashboard updates." Great for stakeholder buy-in early.

These live in shared drives or repos, and AI tools auto-update them as you tweak. Bottom line: Good artifacts prevent "I thought you meant this" drama.

2. Trends: At Least 3 and How They Apply to USEP

Trends in 2025 are like upgrades to your toolkit—they make building software easier, greener, and smarter. As an engineer, I see them popping up in real projects, especially in academic settings like USeP (University of Southeastern Philippines), where courses like Software Engineering 2 emphasize practical, scalable designs for apps in education or local tech. I'll cover three big ones: microservices, AI-assistants, and sustainable architecture. For each, I'll explain what it is simply, then how it applies to USEP-style projects (e.g., team-built systems for tracking or collaboration).

- Microservices: Instead of one giant app (monolith), break it into small, independent "services" (like Lego blocks)—one for user auth, another for data storage. They talk via APIs and can be updated separately. How it Applies to USEP: In a course project like a campus event planner, microservices let your team scale just the registration part during busy sign-ups without crashing the whole thing. It teaches modularity, fitting USeP's focus on real-world engineering—deploy on cheap cloud like AWS free tier, and it handles growing user bases as the uni expands.
- AI-Assistants: Tools like Copilot or custom bots that suggest code, debug, or even design UIs based on your prompts (e.g., "Make a login screen with biometrics"). They're embedded in IDEs and cut manual work by half. How it Applies to USEP: For BSIT assignments, AI speeds up prototyping—generate UML from a user story in seconds, freeing time for creative tweaks. It aligns with USeP's computing curriculum pushing innovation; imagine AI auto-testing your app for bugs, making group projects less error-prone and more about problem-solving.
- Sustainable Architecture: Designing systems to use less energy, like optimizing code for efficient servers or using green clouds (e.g., low-power edge computing). It's about eco-friendly tech without sacrificing speed. How it Applies to USEP: In Philippine contexts like USeP's IoT-focused labs, it means building apps that run on solar-powered devices for rural data collection—reduces carbon footprint while teaching ethics. For a project, swap heavy databases for lightweight ones, cutting costs and aligning with global trends USeP covers in advanced courses.