c. Write a one page essay where you explain clearly why software architecture is important

People often think of software architecture as something reserved for senior engineers or architects, those who've "graduated" from writing code and now just set rules for others. But as Martin Fowler once said, architecture is really just "what matters, whatever it is." In practice, it's the shared understanding among developers of how a system should work and which decisions are hard to undo later.

In fast-paced projects, it's common to hear things like, "Forget design for now, we just need to ship features." And sure, speed matters. But pushing architecture aside isn't just a technical debt problem, it's an economic one. Developers often argue for good design on principle, but in the business world, money usually beats morals. So if we want to make a case for architecture, we need to talk about cost, not just craftsmanship.

Let's break down quality into two types. External quality is what the customer sees features, performance, usability. It's the stuff that sells. Internal quality is under the hood clean code, solid architecture, maintainability. It's invisible to the end user. So if two apps look and behave the same, but one cost $100 more because it was built with better internal quality... why should anyone care? Why spend more on what you can't see?

Here's where Fowler's "Design Durability Hypothesis" comes in. Ask any experienced developer if they've worked on a project where adding new features got slower and messier over time, and you'll almost always get a yes. At some point, the codebase turns into a tangled mess touch one thing and three others break. Progress slows to a crawl. That's what happens when architecture gets ignored.

But it doesn't have to be that way. Some teams have the opposite experience where adding new features gets easier over time. Why? Because the code is clean, the architecture makes sense, and changes are isolated. Instead of fighting the system, developers work with it. The best part? You don't need to wait months to see the payoff. With good architecture, the benefits show up within weeks.

Take database schemas, for example. They've always been considered "architectural" because they're hard to change. But when Pramod Sadalage introduced tools for smooth schema migrations, that changed. Suddenly, databases became flexible. That's not a failure of architecture, it's a win. Unlike buildings, software isn't bound by physics. It's only limited by how well we organize and communicate.

At the end of the day, software architecture isn't about pretty diagrams or following trends. It's about keeping your system flexible so you can continue delivering value over time. Good architecture isn't a luxury or a long-term gamble, it's a practical, near-term investment. And if you ignore it, the cost isn't just technical debt, it's your project slowly grinding to a halt.

d/ Explain what the difference is between software architecture and software design

Software architecture and software design are closely related they're not completely separate things. Instead, they exist on a spectrum, and the difference depends on the situation.

What is Software Architecture?

- It's about the big, important decisions in a system.
- Focuses on major components, how they work together, and the overall structure.
- Involves choices that are hard to change later, so it's important to get them right early.
- It's also about team understanding the architecture reflects what experienced developers agree is important.

What is Software Design?

- Deals with smaller, detailed decisions in the code.
- Includes things like class structure, algorithms, and data structures.
- These choices usually affect just part of the system and are easier to change.
- Developers often make design decisions within their own modules or features.
- The Line Between Them
- There's no fixed line between architecture and design.
- It depends on the project and context.

e. Explain what makes software architecture so difficult.

Software architecture is difficult because it demands important decisions early on, often before the team fully understands the problem or requirements. As Ralph Johnson puts it, architecture is made up of "the decisions you wish you could get right early in a project." These early choices, such as selecting a framework, database, or component structure are typically hard to reverse later, making them high stakes.

Another challenge is that architecture isn't just technical; it's social. Developers, managers, and stakeholders often have different views on what's important, and reaching a shared understanding takes effort. On top of that, software is paradoxically flexible. Unlike physical structures, it can change but that flexibility depends entirely on how it's designed and organized. As Johnson notes, software is limited by imagination, design, and team structure, not technology. In the end, the hardest part of architecture is navigating uncertainty, aligning people, and making decisions that won't hold the system back as it grows.