Lab 1: Software Architecture

---

a. Read the following article

Who Needs an Architect? – Martin Fowler (PDF)

---

b. Watch the following video

What is Software Architecture? – YouTube

---

c. Explain clearly why software architecture is important

Software architecture is important because if defines the foundation on which a system is built, it establishes the structure, key components that determine how well the system meets functional and non-functional requirements. Being this, scalable, maintainable and adaptable to changes.

It provides a vision for developers to align with business requirements.

---

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

Software architecture focuses on **high level** decisions like how component interact each others, what technologies are used, it defines a long-term direction.

Software design, on the other hand, deals with **low level** implementation details like how clases, function and modules are built to implement a specific feature within an architecture.

---

e. Explain what makes software architecture so difficult

It makes difficult because it involves complex trade-offs and uncertain future conditions. Architects most make decision s early, often with incomplete information those decision have a long term consequences. Also, It is difficult to balance multiple quality attributes like security, performance, maintainability. Additionally, when a team and projects grows, keeping the architecture consistent and adaptable becomes challenging.

---

f. Explain clearly the main differences between software architecture in a traditional waterfall project and in an agile project

waterfall architecture is fixed and planned early, while **agile architecture is flexible and grows with the system**.

---

g. Suppose you need to define the architecture for a large, expensive system, and it is important that this system is future-proof because it will be used for at least 20 years.

To design a future-proof system, the architecture must be flexible, modular, and technology-agnostic to be adaptable for changes.

- Layer Architecture: separation concerns so individuals layers or module evolves independently
- Loosely coople and high cohesion: makes components independent but internally consistent.
- Scalability: plans to increase loads and distribure architecture.
- Maintainability: write clean code, well documented, testable code;
- Adaptability: use patterns like micro-services, event-drive or domain-drive architecture to handle changes.
- Regular review - refactor: allocate time for architecture evolution and modernization.

---

h. List all the tasks you can think of that a software architect needs to do in a software development project

- Define the **overall system structure** and component interactions.
- Select **technologies, frameworks, and tools** appropriate for the project.
- Ensure the architecture aligns with **business goals and constraints**.
- Define and maintain **non-functional requirements** (performance, scalability, security, etc.).
- Create **architectural documentation** and communicate the vision to the team.
- **Collaborate with developers**, testers, and product owners to ensure

consistency.

- **Review code and designs** to maintain architectural integrity.
- Identify and mitigate **technical risks** early in the project.
- Support **deployment and infrastructure decisions** (e.g CI/CD, cloud setup).
- Promote **best practices** and continuous improvement across the team.

---

i. For each of the following qualities, give at least one technique to increase this quality

QUALITY	TECHNIQUES TO IMPROVE
1. Performance	Use caching, database indexing, load balancing, and asynchronous processing.
2. Availability	Implement redundancy, failover mechanisms, clustering, and health monitoring.
3. Resilience (against failure)	Apply circuit breaker patterns, retries, and fault-tolerant design using microservices.
4. Reusability	Use modular design, component-based development, and reusable libraries or APIs.
5. Maintainability	Write clean, well-documented code; follow SOLID principles; use layered architecture and automated tests.