**What is software architecture?**

The software architecture of a system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both.

소프트웨어 아키텍처는 프로그램이나 컴퓨터 시스템을 만드는 소프트웨어 구성요소와 이 구성요소들이 외부에 드러내는 속성과 이 구성요소들 사이의 관계로 이루어진 시스템의 구조이다.

**What is architectural drivers?**

Architectural drivers are requirements that will shape the software architecture.

Architectural drivers include Functional Requirements, Quality Attributes and Constraints.

**Why is software architecture needed?**

Enhancing Communication among Stakeholder

Improving Cost and Schedule Estimates

Carrying Early Design Decisions

Defining Constraints on an Implementation

Influencing the Organizational Structure

Inhibiting or Enabling a System’s Quality Attributes

Predicting System Qualities

Reasoning About and Managing Change

Enabling Evolutionary Prototyping

Providing a Basis for Training

여러 이해관계자 간의 의사소통 수단이다.

비용과 일정에 대한 추론을 할 수 있는 중요한 산출물이다.

초기 설계 의사 결정 방향에 대한 선언이다.

개발의 제약 사항을 정한다.

개발 구조를 결정한다.

시스템 품질 속성을 장려하거나 억제한다.

시스템 품질을 예측할 수 있다.

시스템 변화 이유를 좀더 쉽게 파악하거나 관리할 수 있다.

점진적 프로토타입 개발에 도움이 된다.

재사용할 수 있으며 타 시스템에도 적용 가능한 시스템에 대한 추상화된 표현이다.

새로운 팀 구성원을 교육하는 기초가 된다.

Architecture에 기반한 개발은 component의 생성 그 자체 보다는 component의 관계에 집중한다.

**Availability Tactics**

Availability refers to a property of software that it is there and ready to carry out its task when you need it to be.



**Ping/echo** refers to an asynchronous request/response message pair exchanged between nodes, used to determine reachability and the round-trip delay through the associated network path. But the echo also determines that the pinged component is alive and responding correctly. The ping is often sent by a system monitor. Ping/echo requires a time threshold to be set; this threshold tells the pinging component how long to wait for the echo before considering the pinged component to have failed (“timed out”). Standard implementations of ping/echo are available for nodes interconnected via IP.

**Active redundancy (hot spare).** This refers to a configuration where all of the nodes (active or redundant spare) in a protection group receive and process identical inputs in parallel, allowing the redundant spare(s) to maintain synchronous state with the active node(s). Because the redundant spare possesses an identical state to the active processor, it can take over from a failed component in a matter of milliseconds. The simple case of one active node and one redundant spare node is commonly referred to as 1+1 (“one plus one”) redundancy. Active redundancy can also be used for facilities protection, where active and standby network links are used to ensure highly available network connectivity.

**Architecture Description**

Architecture described by Architecture Description.

Architecture Description provides Rationale.

Architecture Description organized by View.

아키텍처는 아키텍처 기술서로 문서화되어야 활용될 수 있다. 아키텍처 기술서는 아키텍처를 결정한 근거를 제시해서 쓸 데 없는 논쟁이나 자원 낭비를 막아야 한다. 아키텍처 기술서는 뷰들로 이루어지며, 뷰는 이해관계자들의 시스템에 대한 관심을 관점에 맞춰 작성한 것이다.

**Broker Pattern**

The broker pattern separates clients from servers by inserting an intermediary.

The client remains completely ignorant of the identity, location, and characteristics of the server.

Brokers add a layer of indirection, and hence latency, between clients and servers, and that layer may be a communication bottleneck. 🡪 Mitigated by reduce overhead and/or increase resource.

The broker can be a single point of failure. 🡪 Mitigated by active/passive redundancy tactic.

A broker adds up-front complexity.

A broker may be a target for security attacks. 🡪 Mitigated by detect intrusion, verify message integrity and/or detect message delay.

A broker may be difficult to test.

**Component-and-Connector Patterns**

Broker Pattern

The broker pattern defines a runtime component, called a broker, that mediates the communication between a number of clients and servers.

Model-View-Controller Pattern

The MVC pattern breaks system functionality into three components: a model, a view, and a controller that mediates between the model and the view.

Pipe-and-Filter Pattern

Data is transformed from a system’s external inputs to its external outputs through a series of transformations performed by its filters connected by pipes.

Client-Server Pattern

Clients initiate interactions with servers, invoking services as needed from those servers and waiting for the results of those requests.

Peer-to-Peer Pattern

Computation is achieved by cooperating peers that request service from and provide services to one another across a network.

Service-Oriented Architecture Pattern

Computation is achieved by a set of cooperating components that provide and/or consume services over a network.

Publish-Subscribe Pattern

Components publish and subscribe to events. When an event is announced by a component, the connector infrastructure dispatches the event to all registered subscribers.

Shared-Data Pattern

Communication between data accessors is mediated by a shared data store. Control may be initiated by the data accessors or the data store. Data is made persistent by the data store.