UNIT-I

ENVISIONING ARCHITECTURE

What Software Architecture Is and What It Isn't:

Definition:

The software architecture of a program or computing system is the structure or structures of the system, which comprise

- *software elements,*
- *the externally visible properties of those elements, and*
- *relationships among them.*

"Externally visible" properties are those assumptions other elements can make of an element, such as its

- provided services
- performance characteristics,
- fault handling,
- shared resource usage,
- and so on.

implications of this definition:

- Architecture defines software elements
  - how the elements relate to each other
  - an architecture is foremost an abstraction of a system that suppresses details of elements that do not affect how they use, are used by, relate to, or interact with other elements. (only public part)
- Systems comprise more than one structure
  - no one structure can claim to be the architecture
  - Relationships and elements might be
    - runtime related ("send data", processes)
    - non-runtime related ("sub model of", "inherits from", class)
- Every computing system with software has a software architecture
  - It does not necessarily follow that the architecture is known to anyone.
  - An architecture can exist independently of its description or specification
- The behavior of each element is part of the architecture
  - allows elements to interact with each other
  - Add specification and properties to the elements and relationships
- The definition is indifferent as to whether the architecture for a system is a good one or a bad one
  - Evaluating the SW architecture in the context of use.

Other Points of View:

A few other definitions:

- Architecture is high-level design
  - the two are not interchangeable
  - Some tasks associated with design are not architectural (e.g. important data structures that will be encapsulated)
- Architecture is the overall structure of the system
  - implies (incorrectly) that systems have but one structure
- Architecture is the structure of the components of a program or system, their interrelationships, and the principles and guidelines governing their design and evolution over time.
  - process-centered definition --> An architecture can be discovered and analyzed independently of any knowledge of the process by which the architecture was designed or evolved

- Architecture is components and connectors.
  - Connectors imply a runtime mechanism
  - Definition concentrates on the runtime architectural structures

Architectural Patterns, Reference Models, and Reference Architectures:

- **Architectural Pattern:**

  A description of element and relation types together with a set of constraints on how they may be used.

  - A set of constraints on an architecture
  - Define a set or family of architectures
  - An architectural pattern is not an architecture
  - Exhibit known quality attributes
  - Often the architect's first major design choice
  - "Architectural style" has also been widely used
  - Example: "Client-Server"

- **Reference Model**:

- *A division of functionality together with data flow between the pieces.*

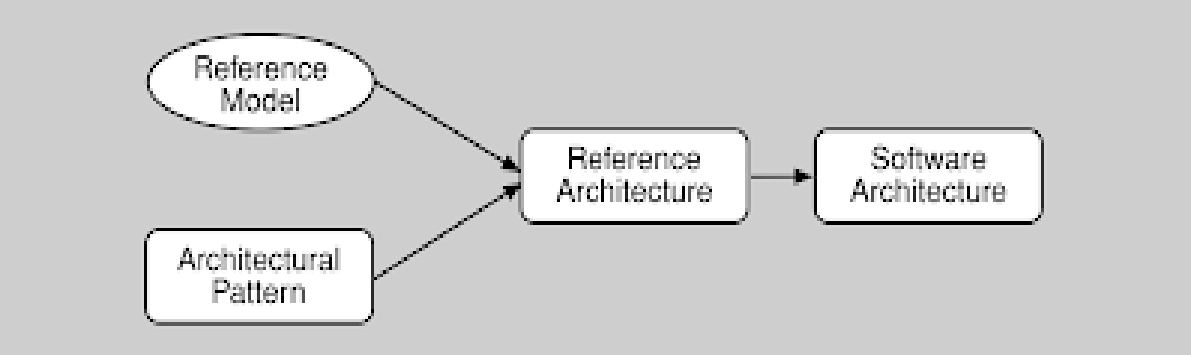
  - Standard decomposition of a known problem into parts that cooperatively solve the problem.
  - characteristic of mature domains
  - Example: Compiler; database management system

- **Reference Architecture**

  A reference model mapped onto software elements (that cooperatively implement the functionality defined in the reference model) and the data flows between them.

  - Whereas a reference model divides the functionality, a reference architecture is the mapping of that functionality onto a system decomposition.

Reference models, architectural patterns, and reference architectures are not architectures; they are useful concepts that capture elements of an architecture.
Each is the outcome of early design decisions.



Importance of Software Architecture:

- **Communication among stakeholders**

  - Every stakeholder has different concerns; SW Architecture can be used as a basis for mutual understanding, negotiation, consensus, and communication among stakeholders.

- **Early design decisions**

  - Software architecture manifests the earliest design decisions, these decision are
    - the most difficult to get correct and
    - the hardest to change later
  - The Architecture Defines Constraints on Implementation
    - Implementation must conform to
      - prescribed design decisions
      - resource allocation decisions
    - Architectures are both prescriptive and descriptive

- The Architecture Dictates Organizational Structure.
  - work breakdown structure
    - work assignments to teams
    - plans, schedules, budgets
    - communication channels among teams
  - dependency and coupling important for work assignment
  - As soon as the work packages are assigned to teams or subcontractors, it is "freezed" and can no longer be changed easily
- The Architecture Inhibits or Enables a System's Quality Attributes.
- Architecture allows to predict system quality attributes without waiting until the system is developed or deployed
- The Architecture Makes It Easier to Reason about and Manage Change
  - Three kind of changes: local, nonlocal, and architectural
- The Architecture Helps in Evolutionary Prototyping
  - Architecture can be analyzed and prototyped as skeletal system. This helps in two ways
    - The system is executable early in the product's life cycle. Elements can be replaced step by step
    - potential performance problems can be identified early
- The Architecture Enables More Accurate Cost and Schedule Estimates
  - Estimations based on system pieces are more accurate
  - The architecture work results in review and validation of the requirements

- **Transferable abstraction of a system**

  - SW architecture constitutes a (relatively small) model that is transferable across similar systems
  - Software architecture can serve as the basis for reuse of
    - requirements
    - development - support artifacts (templates, tools, etc.)
    - code / components
    - experience
  - Software Product Lines Share a Common Architecture
    - Set of software-intensive systems sharing a common, managed set of features
    - powerful approach to multi-system development that shows order-of-magnitude payoffs in time to market, cost, productivity, and product quality
  - Systems Can Be Built Using Large, Externally Developed Elements
    - composing or assembling elements that are likely to have been developed separately, even independently, from each other
    - commercial components, subsystems, and compatible communications interfaces all depend on the principle of interchangeability
    - Architectural mismatch: if subsystems have been built with conflicting architectural assumptions

- Less Is More: It Pays to Restrict the Vocabulary of Design Alternatives
  - restricting to a relatively small number of choices when it comes to program cooperation and interaction
  - Advantages: enhanced re-use, more regular and simpler designs that are more easily understood and communicated, more capable analysis, shorter selection time, and greater interoperability
- An Architecture Permits Template-Based Development
  - Templates can be used to capture in one place the inter-element interaction mechanisms.
- An Architecture Can Be the Basis for Training

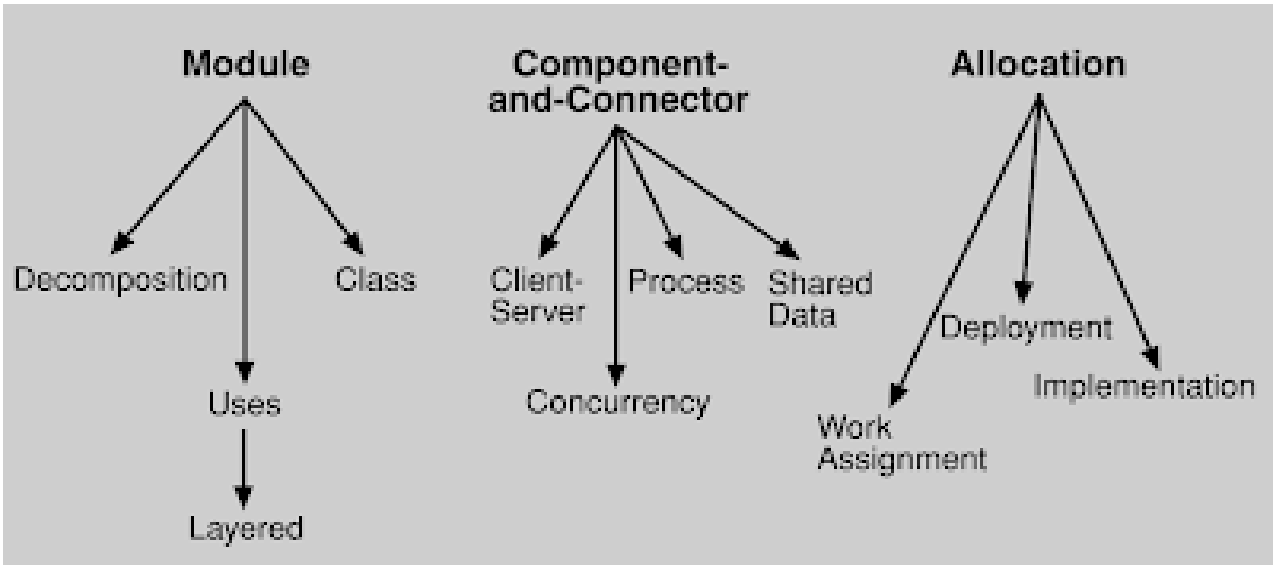
Architectural Structures and Views:

- Definition

  - **View**
    A representation of a set of elements and the relations among them.
  - **Structure**
    The set of elements itself, as they exist in software or hardware

- Restrict our attention at any one moment to one (or a small number) of the software system's structures.
- To communicate meaningfully about an architecture, we must make clear which structure or structures we are discussing at the moment

SOFTWARE STRUCTURES



- ## Module structures:

  Elements are modules, which are units of implementation.
  * What is the primary functional responsibility assigned to each module?
  * What other software elements is a module allowed to use?
  * What other software does it actually use?

  - **Decomposition**
    * shows how larger modules are decomposed into smaller ones recursively
  - **Uses**
    * The units are: modules, procedures or resources on the interfaces of modules
    * The units are related by the uses relation
  - **Layered**
    * "uses relations" structured into layers
  - **Class, or generalization**
    * shows the “inherits-from” or “is-an-instance-of” relations among the modules

- ## Component-and-connector structures:

  Elements are runtime components (units of computation) and connectors (communication vehicles among components)

  The relation is attachment, showing how the components and connectors are hooked together

  * What are the major executing components and how do they interact?
  * What are the major shared data stores?
  * Which parts of the system are replicated?
  * How does data progress through the system?
  * What parts of the system can run in parallel?
  * How can the system’s structure change as it executes?

  - **Process, or communicating processes**
    * units are processes or threads that are connected with each other by communication, synchronization, and/or exclusion operations
  - **Concurrency**
    * The units are components and the connectors are “logical threads”
    * A logical thread is a sequence of computation that can be allocated to a separate physical thread
  - **Shared data, or repository**
    * This structure comprises components and connectors that create, store, and access persistent data
  - **Client-server**
    * The components are the clients and servers, and the connectors are protocols and messages

- **Allocation structures:**

  The relationship between the software elements and the elements in one or more external environments

  * What processor does each software element execute on?
  * In what files is each element stored during development, testing, and system building?
  * What is the assignment of software elements to development teams?

    - **Deployment**
      * Shows how software (usually a process from a component-and-connector view) is assigned to hardware-processing and communication elements
      * Relations are “allocated-to” and “migrates-to” if the allocation is dynamic

    - **Implementation**
      * how software elements (usually modules) are mapped to the file structure(s)
    - **Work assignment**
      * assigns responsibility for implementing and integrating the modules to development teams

Table 2.1. Architectural Structures of a System

Software Structure	Relations	Useful for
Decomposition	Is a submodule of; shares secret with	Resource allocation and project structuring and planning; information hiding, encapsulation; configuration control
Uses	Requires the correct presence of	Engineering subsets; engineering extensions
Layered	Requires the correct presence of; uses the services of; provides abstraction to	Incremental development; implementing systems on top of "virtual machines" portability
Class	Is an instance of; shares access methods of	In object-oriented design systems, producing rapid almost-alike implementations from a common template
Client-Server	Communicates with; depends on	Distributed operation; separation of concerns; performance analysis; load balancing
Process	Runs concurrently with; may run concurrently with; excludes; precedes; etc.	Scheduling analysis; performance analysis
Concurrency	Runs on the same logical thread	Identifying locations where resource contention exists, where threads may fork, join, be

		created or be killed
Shared Data	Produces data; consumes data	Performance; data integrity; modifiability
Deployment	Allocated to; migrates to	Performance, availability, security analysis
Implementation	Stored in	Configuration control, integration, test activities
Work Assignment	Assigned to	Project management, best use of expertise, management of commonality

RELATING STRUCTURES TO EACH OTHER:

- Elements of one structure are related to elements of other structures
- Often the dominant structure is module decomposition, because it spawns the project structure
- Structures represent a powerful separation-of-concerns approach for creating the architecture
- Structures are basis for architecture documentation

WHICH STRUCTURES TO CHOOSE? (Or) VIEW

- Philippe Kruchten "Four Plus One":

- Logical :
  The elements are "key abstractions," which are manifested in the object-oriented world as objects or object classes. This is a module view

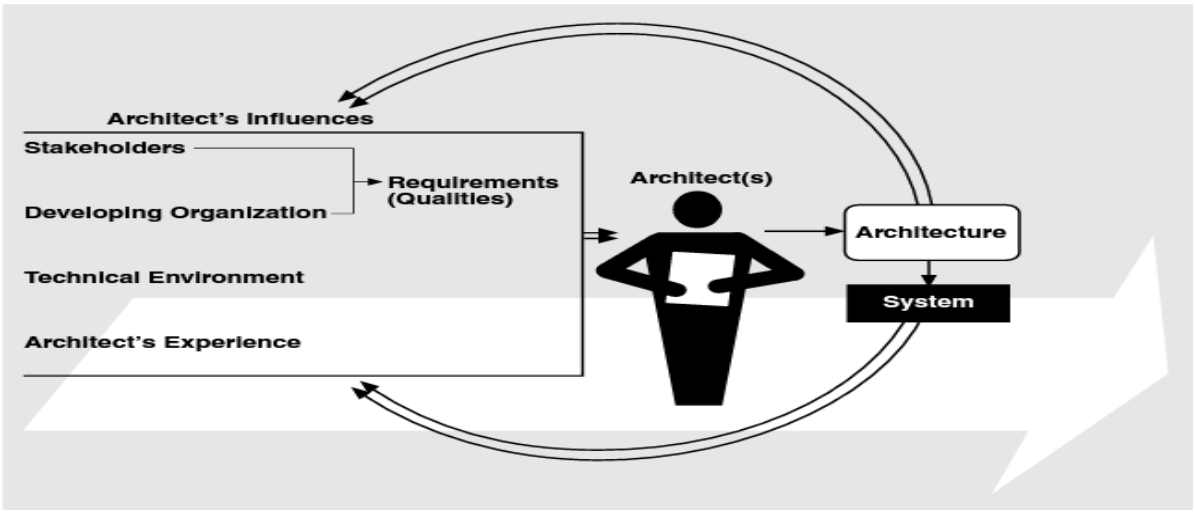
- Process:
  This view addresses concurrency and distribution of functionality. It is a component-and-connector view

- Development:
  This view shows the organization of software modules, libraries, subsystems, and units of development. It is an allocation view, mapping software to the development environment

- Physical:
  This view maps other elements onto processing and communication nodes and is also an allocation view (which others call the deployment view

Architectures Influences:

- Architecture is the result of a set of business and technical decisions.
- The requirements make explicit some—but only some—of the desired properties



ARCHITECTURES ARE INFLUENCED BY

- **SYSTEM STAKEHOLDERS**

  - Stakeholders have different concerns (which may be contradictor)
  - Having an acceptable system involves properties such as
    - performance
    - reliability
    - availability
    - platform compatibility
    - memory utilization
    - network usage

        - security
        - modifiability
        - usability
        - interoperability with other systems
    - Often requirements document do not capture these properties well enough
    - Architects must identify and actively engage stakeholders early in the life cycle to
        - understand the real constraints of the system
        - manage the stakeholders' expectations
        - negotiate the system's priorities
        - make tradeoffs
    - Architects need more than just technical skill: diplomacy, negotiation, and communication skills are essential

- **THE DEVELOPING ORGANIZATION**

    - An architecture is influenced by the structure or nature of the development organization
        - immediate business investment
          (eg. existing architecture)
        - long-term business investment
          (eg. long term infrastructure)
        - organizational structure
          (e.g. subcontracting, skills of employees)

- **THE BACKGROUND AND EXPERIENCE OF THE ARCHITECTS**

    - What worked in the past and what not
    - Education, training, exposure to architectural patterns
    - wish to experiment with something new

- **THE TECHNICAL ENVIRONMENT**

    - practices and techniques of the professional community

- **RAMIFICATIONS OF INFLUENCES ON AN ARCHITECTURE**

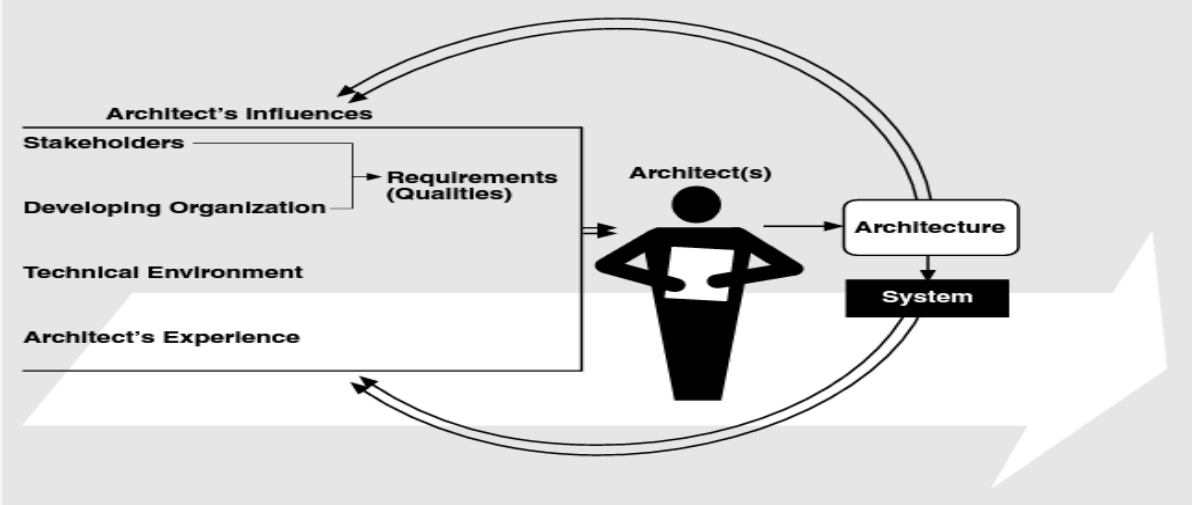
- **THE ARCHITECTURES AFFECT THE FACTORS THAT INFLUENCE THEM**

    - The structure of THE DEVELOPING ORGANIZATION
    - Units of implemented software drives development project's and team structure

- The goals of THE DEVELOPING ORGANIZATION
- Successful system -> establish a foothold in a particular market area.
- Stakeholder requirements for the next system when it is based on the current system
- THE ARCHITECT'S EXPERIENCE
- A few system will influence the software engineering culture

Software Processes and the Architecture Business Cycle:

Software Process is a term given to the organization, Ritualization, and management of software development activities.



Architecture Activities include

- **Creating the Business Case for the system**
  - Product cost?
  - Target market?
  - Time to Market?
  - Interface with other systems
  - System limitation
- **Understanding the requirements**

- Is the system a variation of an existing system?
- Prototype may help to understand the requirements
- It is not until the architecture is created that some tradeoffs among requirements become apparent and force a decision on requirements priorities.

- **Create or Selecting the Architecture**
  - The conceptual integrity is the key to sound system design
  - Conceptual integrity can only be had by a small number of minds

- **Communicating the architecture**
  - Architecture must be communicated clearly and unambiguously to all of the stakeholders (incl. developers, testers and management)
  - Documentation

- **Analyzing or Evaluating the Architecture**
  - Choosing among candidate designs

- **Implementing Based on the architecture**
  - Environment/Infrastructure that assist developer in creating and maintaining the architecture

- **Ensuring Conformance to an Architecture**

  - Constant vigilance is required to ensure that the actual architecture and its representation remain faithful to each other during this phase

What Makes a "Good" Architecture?

- An architecture fits more or less for some stated purpose
- An architecture can only be evaluated in the context of specific goals

Process rules of thumb

- The architecture should be the product of a single architect or a small team with an identified leader
- The architect (team) should have the functional requirements and quality attributes (prioritized)
- The architecture should be well documented
- The architecture should be reviewed with the stakeholders
- The architecture should be evaluated for quality attributes
- The architecture should lend to incremental implementation (via the creation of a "skeletal" system)

- The architecture should result in a specific set of resource contention areas. The resolution of which is clearly specified, circulated and maintained.

Structural rules of thumb

- Well defined modules whose functional responsibilities are allocated on the principles of
    - information hiding (including abstraction of computing infrastructure)
    - separation of concern
- Each module should have a well-defined interface
    - Hides changeable aspects
    - allows the development teams to work independently
- Quality attributes should be achieved using well-known architecture tactics specific to each attribute
- If a architecture depends upon a commercial product, it should be structured such that changing to a different product is inexpensive.
- Creating / Consumption of data should be separated in different modules
- Parallel-Processing modules: Well defined processes or tasks that do not necessarily mirror the module decomposition
- Task/Process assignment to processor should be changeable
- The architecture should feature a small
- number of simple interaction patterns