PART A

UNIT 1

BASICS OF SOFTWARE ARCHITECTURE :

All basic points typed in capital letters are explained in detail in the 8 units . 8 units are grouped according to VTU syllabus.

DEFINATION :

THE SOFTWARE ARCHITECTURE IS THE STRUCTURTE OF STRUCTURES OF A SYSTEM WHICH COMPRISES OF EXTERNALLY VISIBLE PROPERTIES OF ELEMENTS AND THEIR RELATIONSHIPS

SOFTWARE PROCESSES AND ABC (ARCHITECTURAL BUSSINESS CYCLE)

1. CREATING BUSSINESS CASE
2. UNDERSTANDING THE REQUIREMENTS
3.CREATING ARCHITECTURE
4.DOCUMENTATION AND COMMUNICATION
5.ANALYSING ARCHITECTURE
6.GENERAL IMPLEMENTATION CONFERMING TO ARCHITECTURE

ARCHITECT SHOULD HAVE(PROCESS RECOMMENDATIONS)

1. FUNCTIONAL REQUIREMENTS
2. ONE STATIC AND ONE DYNAMIC VIEW
3.QUANTITATIVE ATTRIBUTES(THROUGH PUT)
4.INCREMENTAL IMPLEMENTATION
5.TIME BUDGETS

STRUCTURAL RULES

1. WELL DEFINED MODULES(FUNCTIONAL RESPONSIBILITIES)
2. WELL DEFINED INTERFACES
3. QUALITY ATTRIBUTES
4. DO NOT DEPEND UPON VERSIONS
5. EVERY TASK FOR A PERTICULAR PROCESSOR SHOULD BE EASILY MODIFIABLE.

IMPLICATIONS OF THE DEFINATION IN DETAIL :

1. ARCHITECTURE DEFINES SOFTWARE ELEMENTS
2. SYSTEM COMPRISES OPF MORE THAN ONE STRUCTURE
3. THE BEHAVIOR OF EACH ELEMENT IS PART OF ARCHITECTURE.
4. ARCHITECTURE IS A HIGH LEVEL DESIGN

VASA SHIP DESIGN

1. **1620 , SWEDEN**
2. **70 METER LONG , CARRIES 300 SOLDIERS , 64 HEAVY GUNS MOUNTED ON GUN DECKS**
3. **DESIGNER DIED , NO WRITTEN DOCUMENTS**
4. **POOR REQUIREMENTS AND EXTENSION .**
5. **1628 SHIP DROWNED.**

Moral of the ship design : poor requirements and extensibility , all quality attributes are explained in unit 3

Architectural Patterns, Reference Models, and Reference Architectures

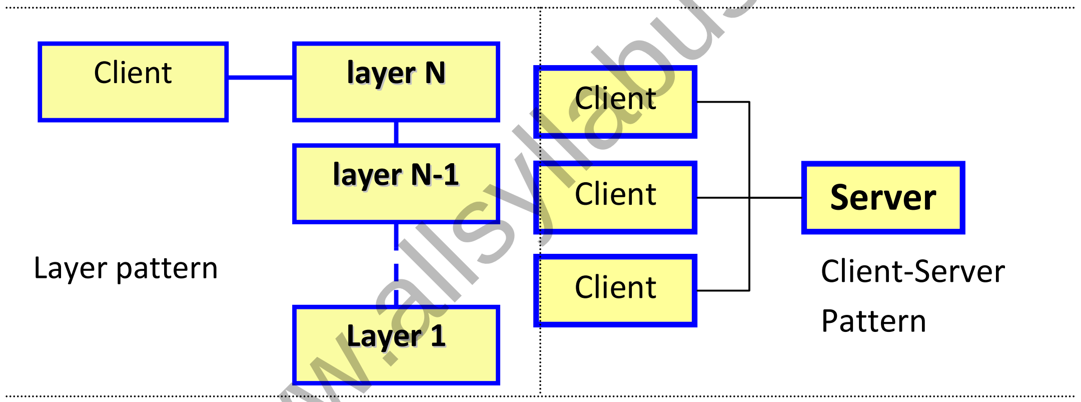
Fowler in his Analysis of Patterns (1996), defines Patterns as “an idea that has been useful in one practical context and will probably be useful in others” Architectural Patterns are the highest level patterns, which specify fundamental structure of an application

Architectural Patterns are composed of a set of component types (e.g., Process, procedure) that perform some function at runtime. The components are arranged in a specific topological layout showing their runtime relationships. It also includes rules and guidelines for organizing the relationships between them. Patterns also specify a set of connectors (e.g., Data streams, sockets) that mediate communication among components. An architectural pattern is generic and domain-neutral.

The client-server pattern is an example of architectural pattern, which has two element types (Client and Server). Their relationship is a protocol that the server uses to communicate with each of its clients. The semantic constraint is that the clients don't communicate directly with each other. Note that functionality is excluded here.

The characteristics of all architectural patterns are that they exhibit known quality attributes. Some patterns solve performance problems, others apply to high-security systems, or high-availability goals. An architectural pattern is basically reuse of experience. What makes it a pattern is the fact that it is recurring. If a design solution in its essential distilled form is found in multiple systems and is a successful design, then this solution becomes a pattern

Examples of Architectural pattern include Layer, Pipe and Filter and Client-Server Pattern.



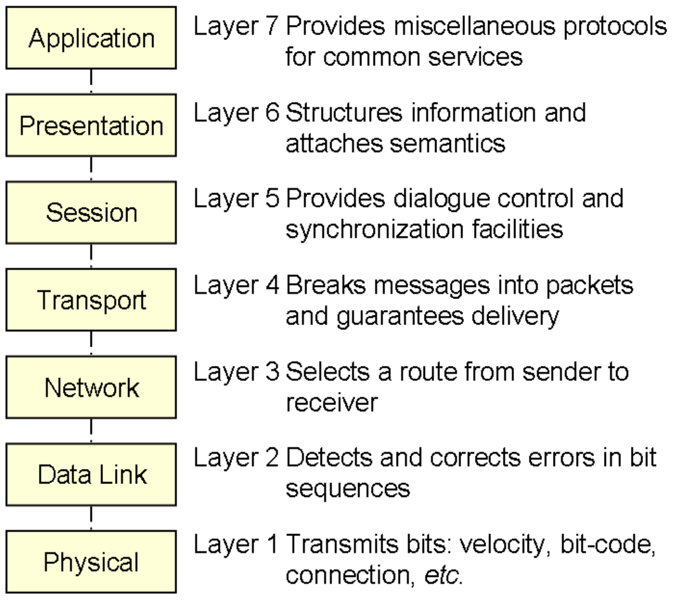
Reference Models

A reference model is a pre-engineered and integrated organizational views. They arise from experience, and are thus a characteristic of mature domains. For example, an operating system has certain major subsystems like file system, a memory manager, a process scheduler, a network interface, and an inter-process communication subsystem.

A reference model specifies the division of functionality into elements, together with data flow between them. Note that this is not the same as the "pattern," which is a way to enable that particular flow. So what you have is a standard decomposition of a known problem into parts that cooperatively solve the problem.

For example, the Compiler reference model includes a description of parts and data flow between them.

A popular example of a reference model is the OSI Reference Model shown below


Application
Layer 7 Provides miscellaneous protocols for common services
Presentation
Layer 6 Structures information and attaches semantics
Session
Layer 5 Provides dialogue control and synchronization facilities
Transport
Layer 4 Breaks messages into packets and guarantees delivery
Network
Layer 3 Selects a route from sender to receiver
Data Link
Layer 2 Detects and corrects errors in bit sequences
Physical
Layer 1 Transmits bits: velocity, bit-code, connection, *etc.*

A reference model is used to clarify things within an environment or a problem space by providing a description of the problem and highlighting the major concerns of the stakeholders

A reference model creates standards for both the elements in the model and their relationships to one another. It also helps break down a large problem space into smaller problems that can be understood, tackled, and refined.

A reference model helps improve communication by creating a model of the entities with clear roles, and their relationships

Reference Architectures

Reference architecture is a reference model mapped onto software elements and the data flows between them. It is a proven template solution for architecture for a particular domain. It also provides a common vocabulary with which to discuss implementations, often with the aim to stress commonality.

In a Reference Architecture, the software elements cooperatively implement the functionality defined in the reference model. That is, it maps the functionality of a reference model onto system decomposition. This mapping may not be one to one. That is an element may implement a part of a function or several functions.

An example Reference architecture might map the “compiler” reference model onto “pipe and filter” architecture pattern. Note that this is still not a software architecture, because it is too generic, nevertheless it’s a start.

Characteristics of a Good Reference Architectures

A reference architecture is a description of a solution that solves a specific recurring problem. Organizations adopt certain reference architecture as an accepted approach to solving a specific problem within a domain. This will be used repeatedly, in numerous projects to solve similar problems.

A reference architecture is a *proven* approach to solving the specific problem, that is, successfully used in previous projects and applications. Many of the reference architectures in use within organization that have been recognized from successful implementations, documented for easy reuse.

A reference architecture needs to be well described if it's to be useful. It needs to assist with decision making by architects, project managers and developers. The reference architecture needs to describe what problem is solves, when it should be used and how it should used. It also should include standards that need to be implemented as part of the solution.

Reference Architecture Benefits

A reference architecture provides a common domain-specific language for the various stakeholders. Within an organization that adopts a reference architecture it provides consistency of implementation of technology to solve problems. They also supports enterprise architecture and IT governance by encouraging adherence to common standards and patterns. Another important benefit is it encourages adoption of common asset reuse approaches. This is accommodated by documenting existing well-proven designs. Reference architectures are especially useful as they help build complex systems.

Importance of Software Architectures:

There are fundamentally three reasons for software architecture's importance:

1. Communication among stakeholders. Software architecture represents a common abstraction of a system that can be used as a basis for mutual understanding, negotiation, consensus, and communication.
2. Early design decisions. Software architecture manifests the earliest design decisions about a system, and these early bindings carry weight far out of proportion to their individual gravity with respect to the system's remaining development, its deployment, and its maintenance life. It is also the earliest point at which design decisions governing the system to be built can be analyzed.
3. Transferable abstraction of a system. Software architecture constitutes a relatively small, intellectually graspable model for how a system is structured and how its elements work together, and this model is transferable across systems

1. Communication among stakeholders
    a. Each stakeholder of a software system is concerned with different system characteristics that are affected by the architecture.

b. Architecture provides a common language in which different concerns can be expressed, negotiated, and resolved at a level that is intellectually manageable even for large, complex systems

2. Architecture manifests the earliest set of design decisions

The early decisions are the most difficult to get correct and the hardest to change later in the development process, and they have the most far-reaching effects.

a. The Architecture Defines Constraints on Implementation

An implementation exhibits architecture if it conforms to the structural design decisions described by the architecture.
Resource allocation decisions also constrain implementations.
The constraints permit a separation of concerns that allows management decisions to make the best use of personnel and computational capacity.

b. The Architecture Dictates Organizational Structure

The structure dictated by architecture on the system becomes engraved in the structure of the development project. Because the system architecture includes the highest-level decomposition of the system, it is typically used as the basis for the work breakdown structure, which in turn dictates units of planning, scheduling, and budget; inter team communication channels; configuration control and file system organization; integration and test plans and procedures; and even minutiae such as how the project intranet is organized and how many team picnics there are

A side effect of establishing the work breakdown structure is to freeze some aspects of the software architecture. A group that is responsible for one of the subsystems will resist having its responsibilities distributed across other groups.

c. The Architecture Inhibits or Enables a System's Quality Attributes

Whether a system will be able to exhibit its desired quality attributes is substantially determined by its architecture.

- If high performance is the criteria, the time-based behavior of elements must be managed as well as the frequency and volume of inter-element communication.
- If modifiability is important, responsibilities are assigned to elements such that changes to the system do not have far-reaching consequences.
- If security is the issue, inter-element communication must be protected and managed and information access by the components.
- If scalability is needed in the system, the use of resources is carefully localized to facilitate the introduction of higher-capacity replacements.
- If the project needs to deliver incremental subsets of the system, inter-component usage must be carefully managed.
- If the elements of the system are required to be re-usable in other systems, restrict inter-element coupling so that when an element is extracted, it does

not come out with too many attachments to its current environment to be useful.

d. Predicting System Qualities by Studying the Architecture

It is possible to make quality predictions about a system based solely on an evaluation of its architecture.
Architecture evaluation techniques such as the Architecture Tradeoff Analysis Method support top-down insight into the attributes of software product quality that is made possible by software architectures.

e. The Architecture Makes It Easier to Reason about and Manage Change

Software systems change over their lifetimes. They do so often and often with difficulty.
Every architecture partitions possible changes into three categories: local, nonlocal, and architectural.
A local change can be accomplished by modifying a single element.
A nonlocal change requires multiple element modifications but leaves the underlying architectural approach intact.
An architectural change affects the fundamental ways in which the elements interact with each other.
So an effective architecture is one in which the most likely changes are also the easiest to make.
Reasoning about the architecture can provide the insight necessary to make decisions about proposed changes.

f. The Architecture Helps in Evolutionary Prototyping

Once an architecture has been defined, it can be analyzed and prototyped as a skeletal system. This helps in two ways:
The system is executable early in the product's life cycle. Its fidelity increases as prototype parts are replaced by complete versions of the software.
A special case of having the system executable early is that potential performance problems can be identified early in the product's life cycle.
Each of these benefits reduces the risk in the project.

g. The Architecture Enables More Accurate Cost and Schedule Estimates

Cost and schedule estimates are an important management tool to enable the manager to acquire the necessary resources and to understand whether a project is in trouble. Cost estimations based on an understanding of the system pieces are, inherently, more accurate than those based on overall system knowledge.
Each team will be able to make more accurate estimates for its piece than a project manager will and will feel more ownership in making the estimates come true.

3. Architecture as a transferable, re-usable model

The earlier in the life cycle re-use is applied, the greater the benefit that can be achieved. While code re-use is beneficial, re-use at the architectural level provides tremendous leverage for systems with similar requirements.

a. Software Product Lines Share a Common Architecture

A software product line or family is a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way.

Chief among these core assets is the architecture that was designed to handle the needs of the entire family.

Similar to other capital investments, the architecture for a product line becomes a developing organization's core asset.

b. Systems Can Be Built Using Large, Externally Developed Elements

One key aspect of architecture is its organization of element structure, interfaces, and operating concepts.

The most significant principle of this organization is interchangeability.

When the components that are candidates for importation and re-use are distinct subsystems that have been built with conflicting architectural assumptions, unanticipated complications can increase the effort required to integrate their functions.

c. It Pays to Restrict the Vocabulary of Design Alternatives

As useful architectural patterns and design patterns are collected, it becomes clear that computer programs can be combined in more or less infinite ways.

Voluntarily restricting ourselves to a relatively small number of choices when it comes to program cooperation and interaction has tremendous benefits.

d. An Architecture Permits Template-Based Development

An architecture embodies design decisions about how elements interact that, while reflected in each element's implementation, can be localized and written just once.

Templates can be used to capture in one place the inter-element interaction mechanisms.

e. An Architecture Can Be the Basis for Training

The architecture, including a description of how elements interact to carry out the required behavior, can serve as the introduction to the system for new project members.