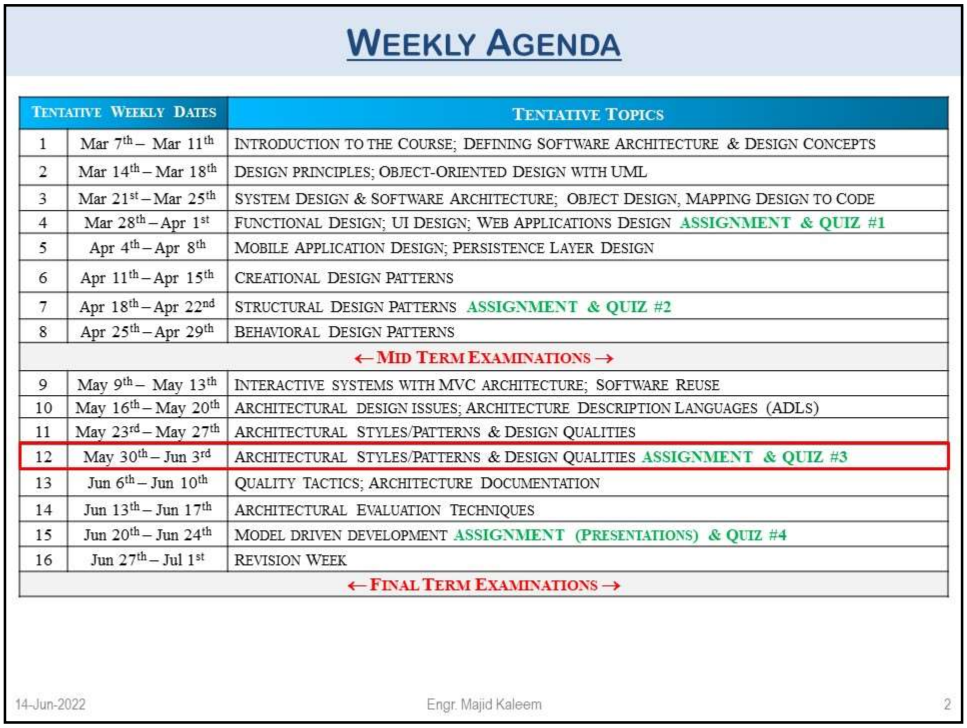
Software Design & Architecture
Spring 2022 - Week-12

مدرس: مہندس ماجد کلیم

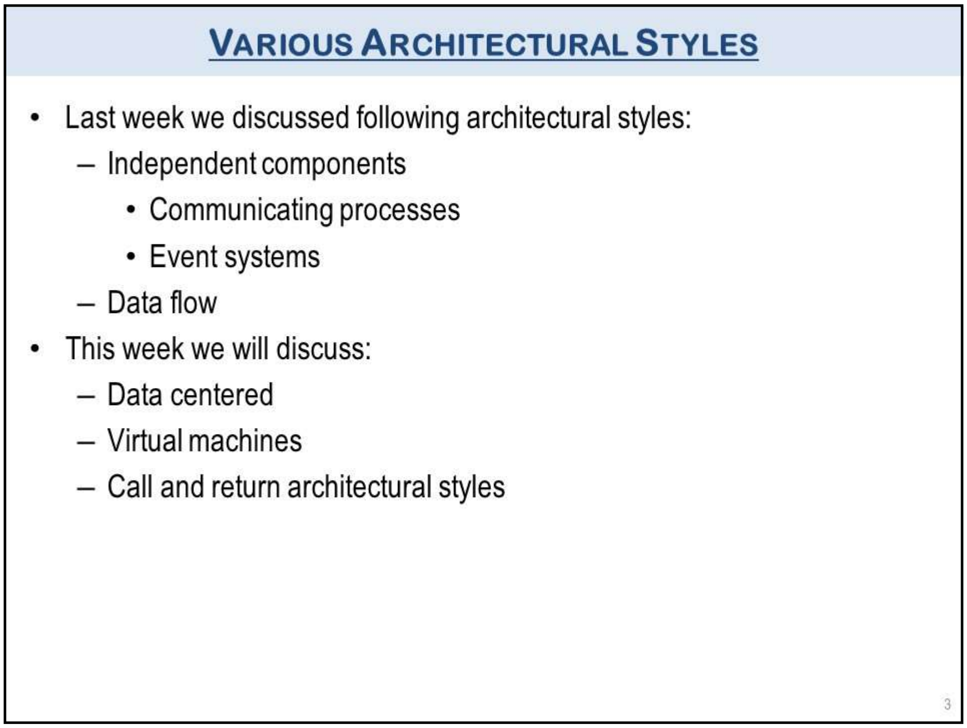
جامعہ بحریہ، واقعگاہ کراچی

Engr. Majid Kaleem



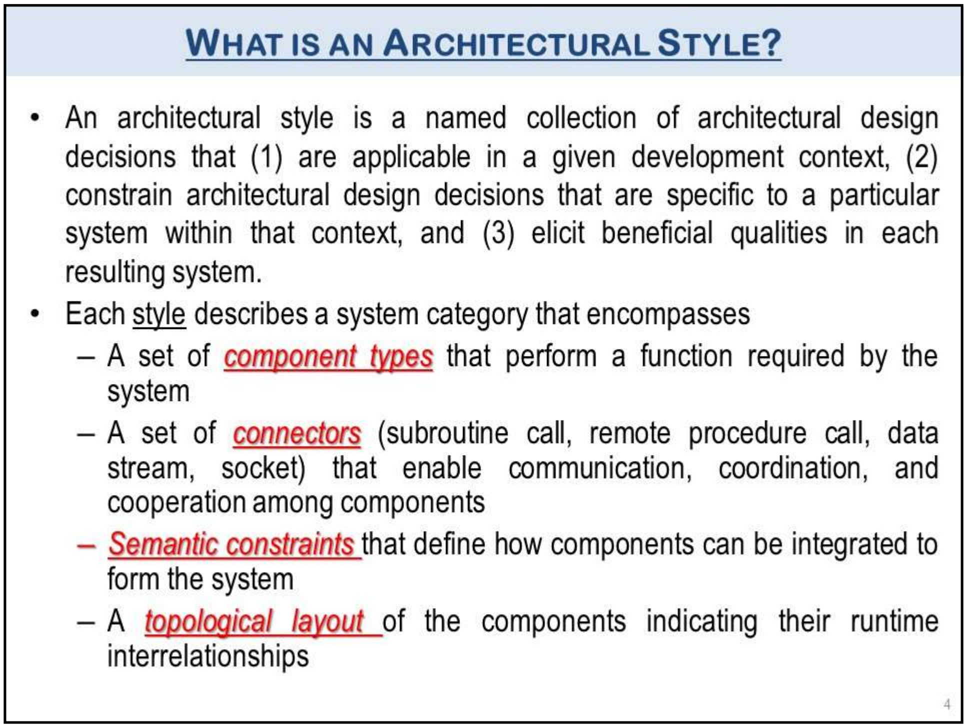
Weekly Agenda

Tentative Weekly Dates		Tentative Topics
1	Mar 7th – Mar 11th	Introduction to the Course; Defining Software Architecture & Design Concepts
2	Mar 14th – Mar 18th	Design principles; Object-Oriented Design with UML
3	Mar 21st – Mar 25th	System Design & Software Architecture; Object Design, Mapping Design to Code
4	Mar 28th – Apr 1st	Functional Design; UI Design; Web Applications Design **ASSIGNMENT & QUIZ #1**
5	Apr 4th – Apr 8th	Mobile Application Design; Persistence Layer Design
6	Apr 11th – Apr 15th	Creational Design Patterns
7	Apr 18th – Apr 22nd	Structural Design Patterns **ASSIGNMENT & QUIZ #2**
8	Apr 25th – Apr 29th	Behavioral Design Patterns
← Mid Term Examinations →		
9	May 9th – May 13th	Interactive systems with MVC architecture; Software Reuse
10	May 16th – May 20th	Architectural design issues; Architecture Description Languages (ADLS)
11	May 23rd – May 27th	Architectural Styles/Patterns & Design Qualities
12	May 30th – Jun 3rd	Architectural Styles/Patterns & Design Qualities **ASSIGNMENT & QUIZ #3**
13	Jun 6th – Jun 10th	Quality Tactics; Architecture Documentation
14	Jun 13th – Jun 17th	Architectural Evaluation Techniques
15	Jun 20th – Jun 24th	Model driven development **ASSIGNMENT (PRESENTATIONS) & QUIZ #4**
16	Jun 27th – Jul 1st	Revision Week
← Final Term Examinations →		



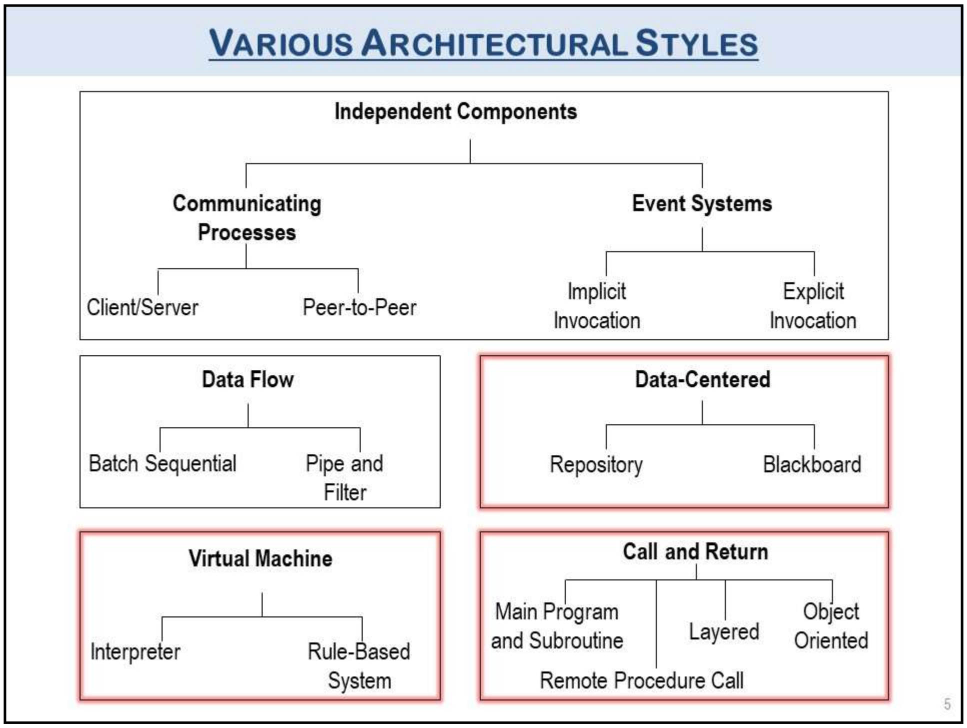
VARIOUS ARCHITECTURAL STYLES

- Last week we discussed following architectural styles:
  - Independent components
    - Communicating processes
    - Event systems
  - Data flow
- This week we will discuss:
  - Data centered
  - Virtual machines
  - Call and return architectural styles



WHAT IS AN ARCHITECTURAL STYLE?

- An architectural style is a named collection of architectural design decisions that (1) are applicable in a given development context, (2) constrain architectural design decisions that are specific to a particular system within that context, and (3) elicit beneficial qualities in each resulting system.
- Each style describes a system category that encompasses
  - A set of ***component types*** that perform a function required by the system
  - A set of ***connectors*** (subroutine call, remote procedure call, data stream, socket) that enable communication, coordination, and cooperation among components
  - ***Semantic constraints*** that define how components can be integrated to form the system
  - A ***topological layout*** of the components indicating their runtime interrelationships



VARIOUS ARCHITECTURAL STYLES

Independent Components

- **Communicating Processes**
  - Client/Server
  - Peer-to-Peer
- **Event Systems**
  - Implicit Invocation
  - Explicit Invocation

Data Flow

- Batch Sequential
- Pipe and Filter

Data-Centered

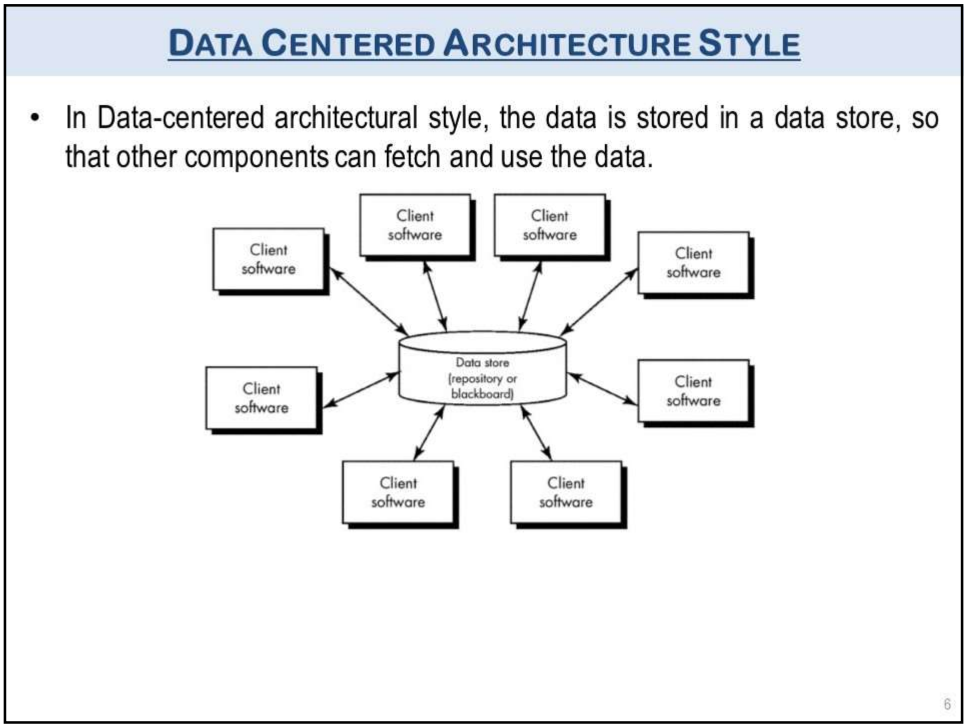
- Repository
- Blackboard

Virtual Machine

- Interpreter
- Rule-Based System

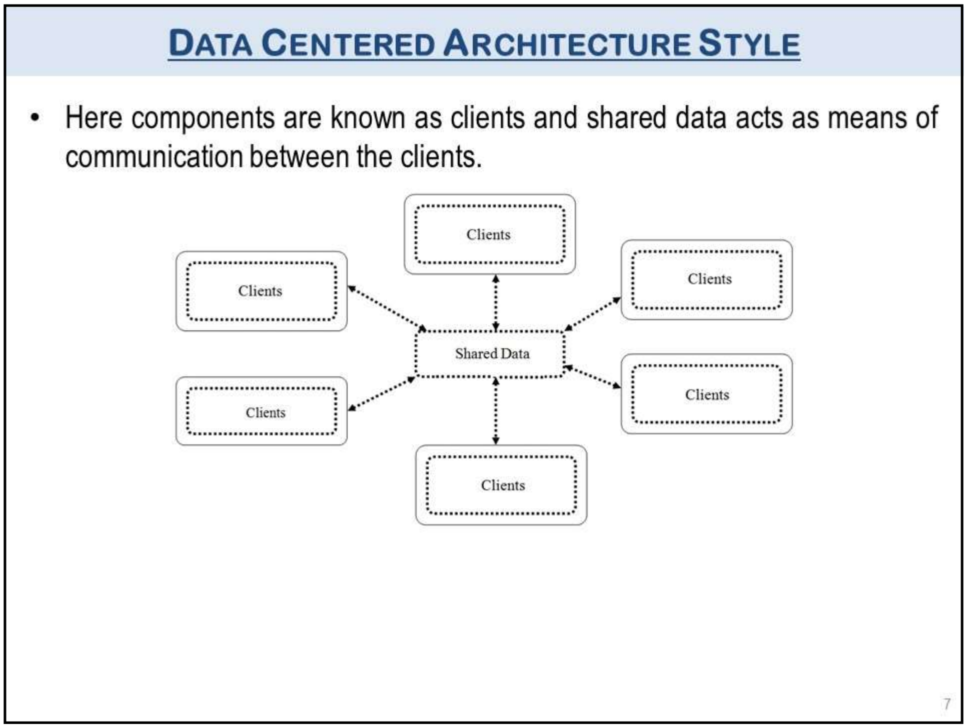
Call and Return

- Main Program and Subroutine
- Remote Procedure Call
- Layered
- Object Oriented



DATA CENTERED ARCHITECTURE STYLE

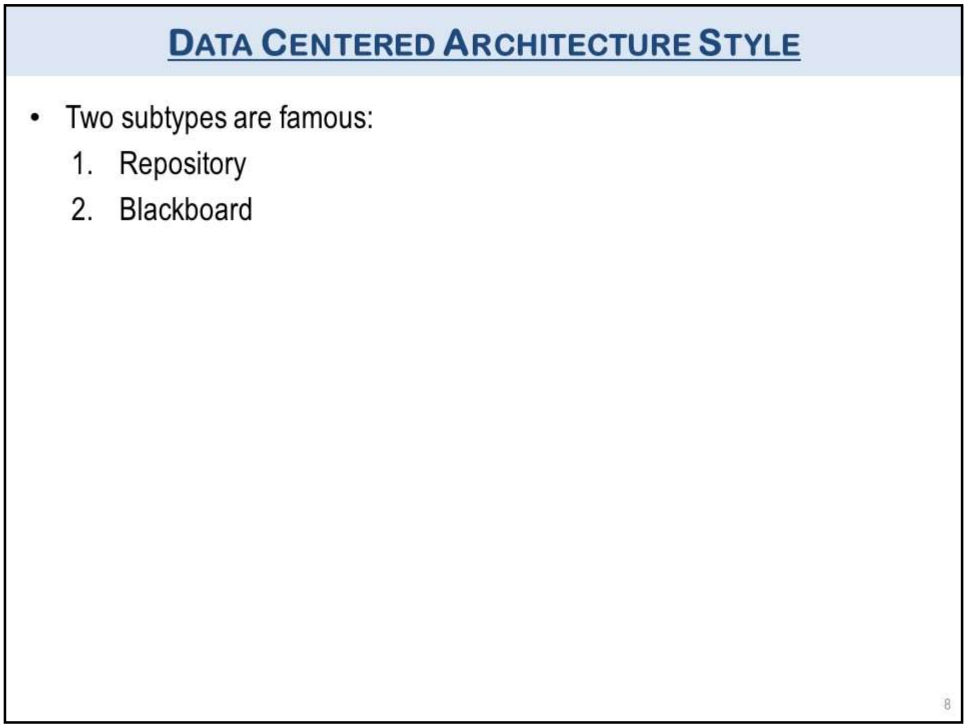
- In Data-centered architectural style, the data is stored in a data store, so that other components can fetch and use the data.



DATA CENTERED ARCHITECTURE STYLE

- Here components are known as clients and shared data acts as means of communication between the clients.

7



DATA CENTERED ARCHITECTURE STYLE

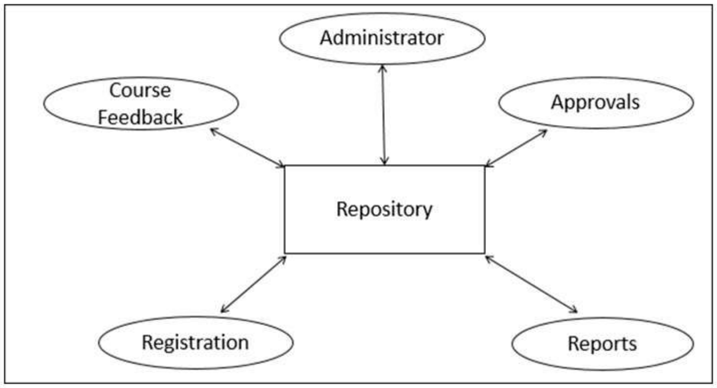
- Two subtypes are famous:
  1. Repository
  2. Blackboard

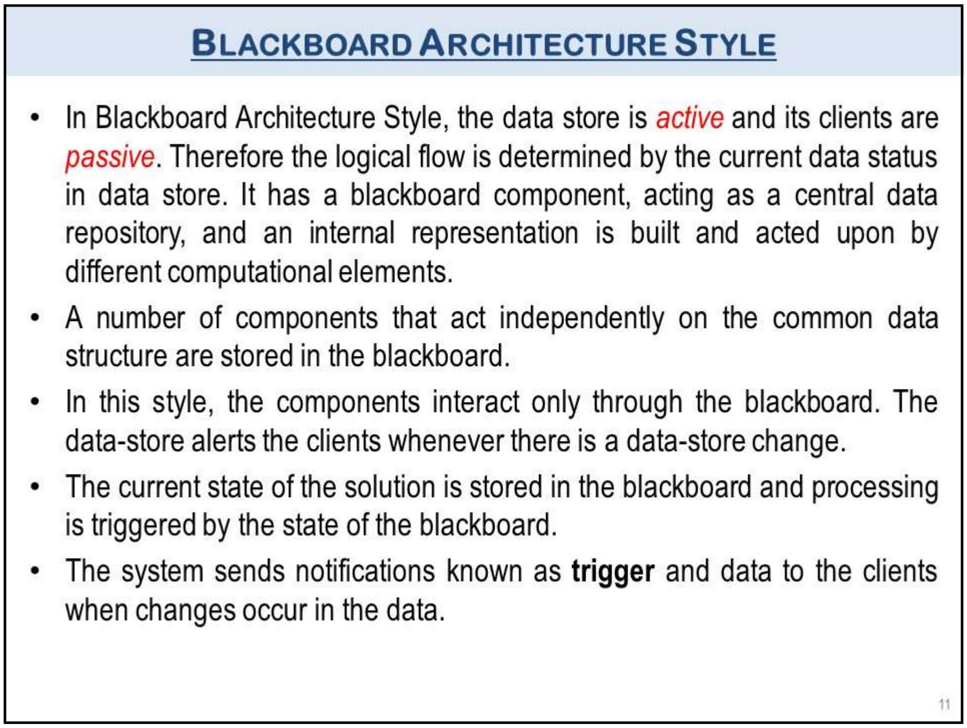
8

Repository Style

- In Repository Architecture Style, the data store is *passive* and the clients (software components or agents) of the data store are *active*, which control the logic flow. The participating components check the data-store for changes.
- The client sends a request to the system to perform actions (e.g. insert data).
- The computational processes are independent and triggered by incoming requests.
- If the types of transactions in an input stream of transactions trigger selection of processes to execute, then it is traditional database or repository architecture, or passive repository.
- This approach is widely used in DBMS, library information system, the interface repository in CORBA, compilers and CASE (computer aided software engineering) environments.

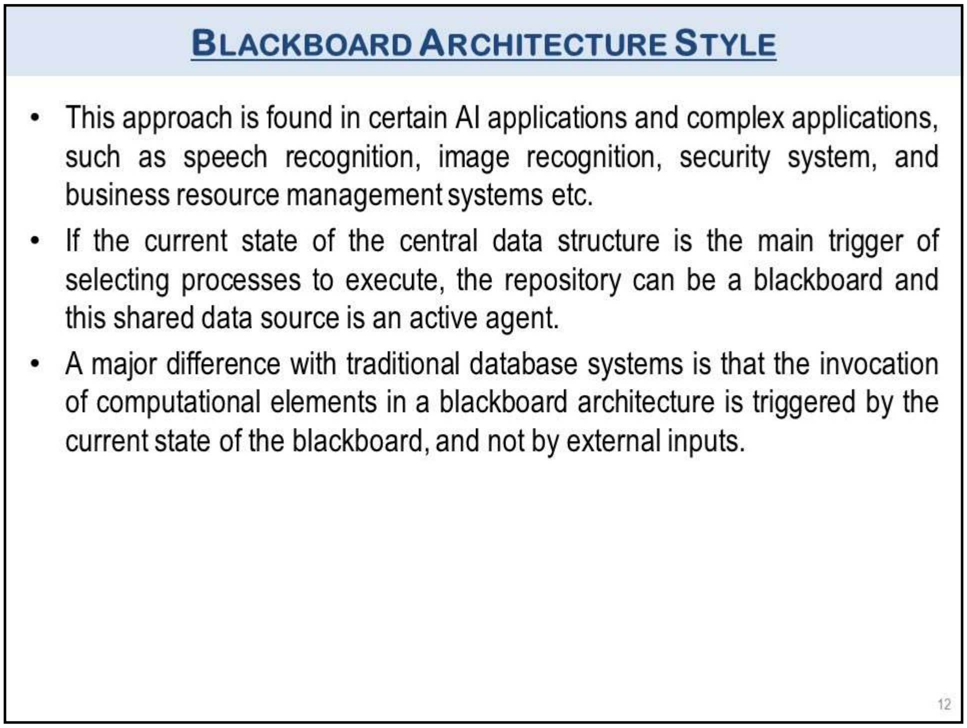
Repository Style





Blackboard Architecture Style

- In Blackboard Architecture Style, the data store is *active* and its clients are *passive*. Therefore the logical flow is determined by the current data status in data store. It has a blackboard component, acting as a central data repository, and an internal representation is built and acted upon by different computational elements.
- A number of components that act independently on the common data structure are stored in the blackboard.
- In this style, the components interact only through the blackboard. The data-store alerts the clients whenever there is a data-store change.
- The current state of the solution is stored in the blackboard and processing is triggered by the state of the blackboard.
- The system sends notifications known as **trigger** and data to the clients when changes occur in the data.



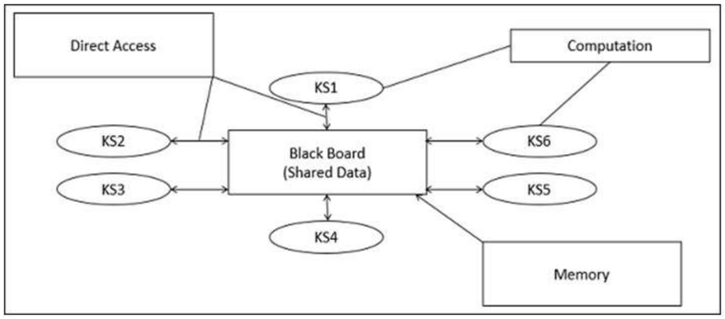
Blackboard Architecture Style

- This approach is found in certain AI applications and complex applications, such as speech recognition, image recognition, security system, and business resource management systems etc.
- If the current state of the central data structure is the main trigger of selecting processes to execute, the repository can be a blackboard and this shared data source is an active agent.
- A major difference with traditional database systems is that the invocation of computational elements in a blackboard architecture is triggered by the current state of the blackboard, and not by external inputs.

BLACKBOARD ARCHITECTURE STYLE

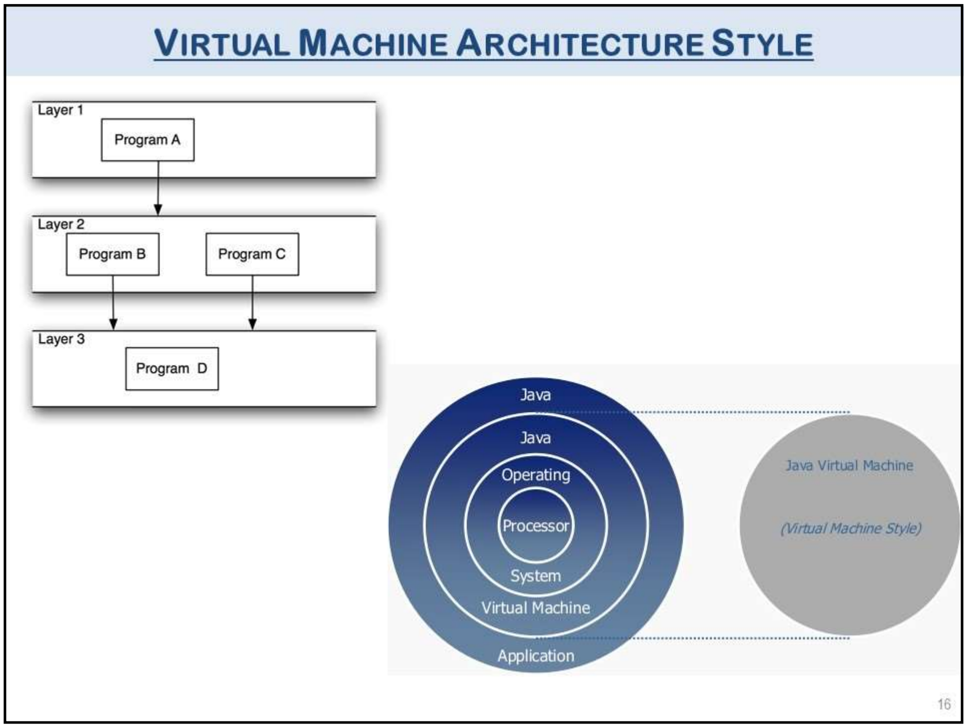
- The blackboard model is usually presented with three major parts −
- **Knowledge Sources (KS)**
- Knowledge Sources, also known as **Listeners** or **Subscribers** are distinct and independent units. They solve parts of a problem and aggregate partial results. Interaction among knowledge sources takes place uniquely through the blackboard.
- **Blackboard Data Structure**
- The problem-solving state data is organized into an application-dependent hierarchy. Knowledge sources make changes to the blackboard that lead incrementally to a solution to the problem.
- **Control**
- Control manages tasks and checks the work state.

BLACKBOARD ARCHITECTURE STYLE



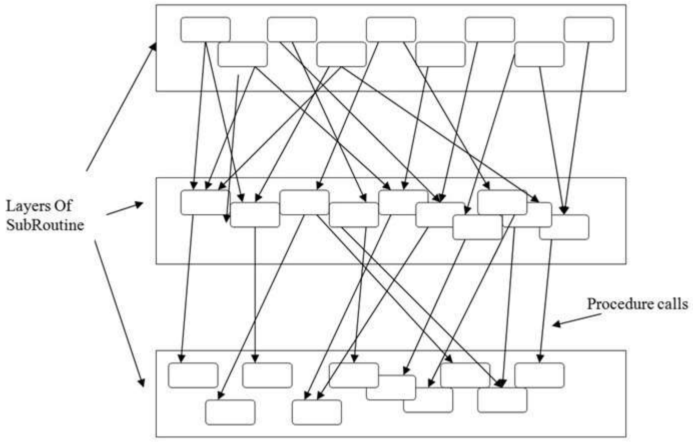
VIRTUAL MACHINE ARCHITECTURE STYLE

- **Summary:** Consists of an ordered sequence of layers, each *layer* or *virtual machine*, offers a set of services that may be accessed by programs (subcomponents) residing within the layer above it.
- **Components:** Layers offering a set of services to other layers, typically comprising several programs (subcomponents).
- **Connector:** Typically procedure calls.
- **Data elements:** Parameters passed between layers.
- **Topology:** Linear for strict virtual machines; a directed cyclic graph in looser interpretation.



VIRTUAL MACHINE ARCHITECTURE STYLE

VIRTUAL MACHINE ARCHITECTURE STYLE

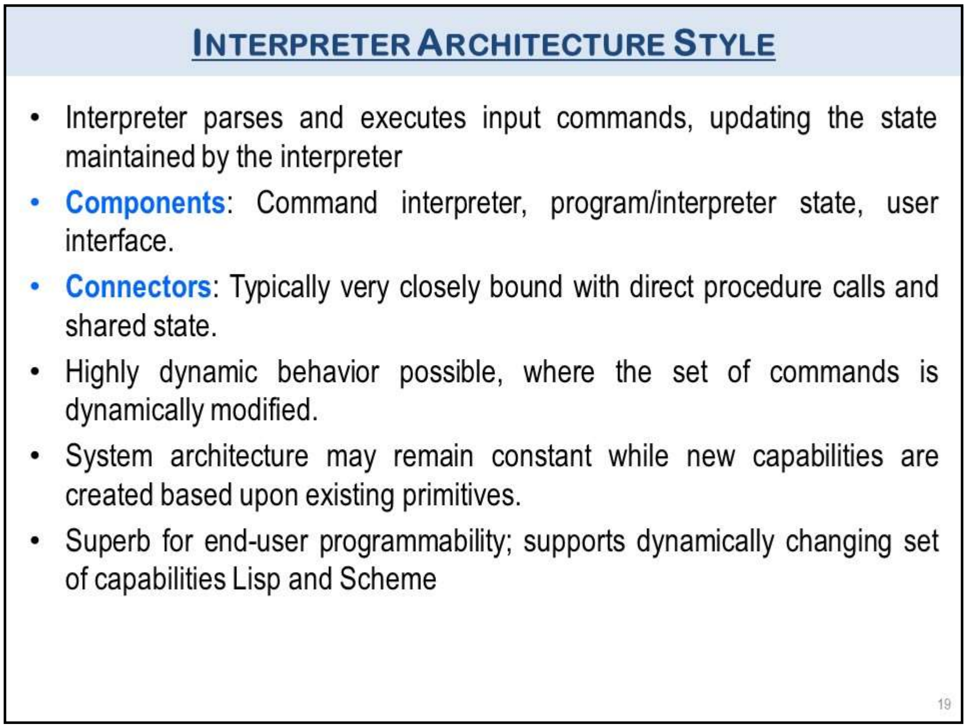


17

VIRTUAL MACHINE ARCHITECTURE STYLE

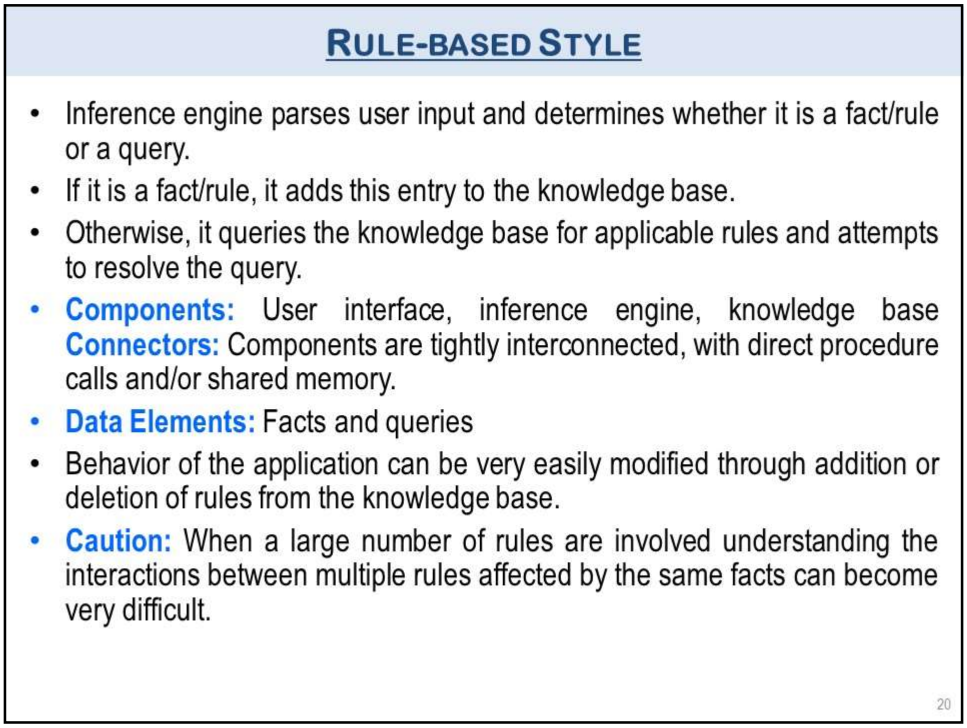
- Two subtypes are famous:
  1. Interpreter
  2. Rule-based

18



INTERPRETER ARCHITECTURE STYLE

- Interpreter parses and executes input commands, updating the state maintained by the interpreter
- **Components**: Command interpreter, program/interpreter state, user interface.
- **Connectors**: Typically very closely bound with direct procedure calls and shared state.
- Highly dynamic behavior possible, where the set of commands is dynamically modified.
- System architecture may remain constant while new capabilities are created based upon existing primitives.
- Superb for end-user programmability; supports dynamically changing set of capabilities Lisp and Scheme



RULE-BASED STYLE

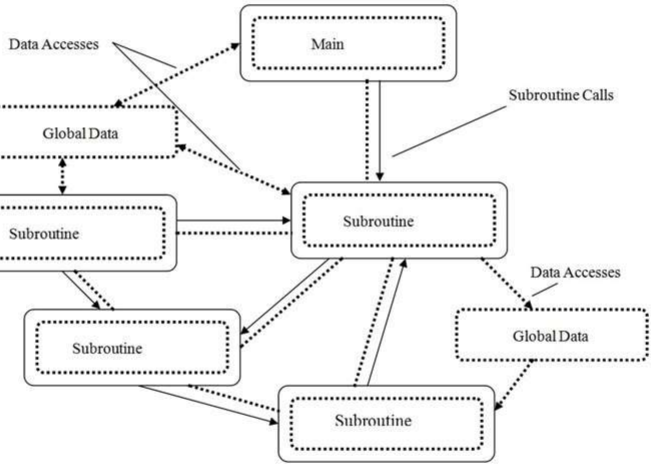
- Inference engine parses user input and determines whether it is a fact/rule or a query.
- If it is a fact/rule, it adds this entry to the knowledge base.
- Otherwise, it queries the knowledge base for applicable rules and attempts to resolve the query.
- **Components:** User interface, inference engine, knowledge base **Connectors:** Components are tightly interconnected, with direct procedure calls and/or shared memory.
- **Data Elements:** Facts and queries
- Behavior of the application can be very easily modified through addition or deletion of rules from the knowledge base.
- **Caution:** When a large number of rules are involved understanding the interactions between multiple rules affected by the same facts can become very difficult.

CALL AND RETURN ARCHITECTURAL STYLE

- Call and Return architectural style is most widely used style since many years.
- In call and return architectural style large complex systems are divided into several smaller components known as subroutines.
- Each component can only be executed only when it gets the control.
- Each component has its own fixed entry (point at which execution starts) and exits (point at which execution terminates) locations.
- Passing control from one component to other is known as subroutine calls.
- Every component gets the call (this call can also be known as function call) for control and has to return the control to other components before termination. Call and Return architectural style has three sub categories.

CALL AND RETURN ARCHITECTURAL STYLE

Structure of call and return architecture.



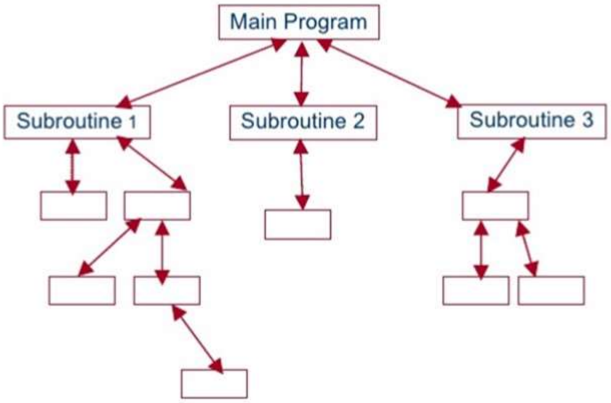
CALL AND RETURN ARCHITECTURAL STYLE

- Some common types are:
  1. Main Program and Subroutine
  2. Remote Procedure Call
  3. Layered System
  4. Object Oriented

MAIN PROGRAM AND SUBROUTINE

- Hierarchical structure of call and return style are referred as Main program and subroutine.
- In this Component which has access to whole program is known as main program.
- Control is given from main program to all other subroutine.
- Group of subroutines that shares same data are known as Module.
- Shared data that can be accessed by all subroutines are known as global variables.
- The main property of this style is to reduce complexity and encourage easy modifications

MAIN PROGRAM AND SUBROUTINE

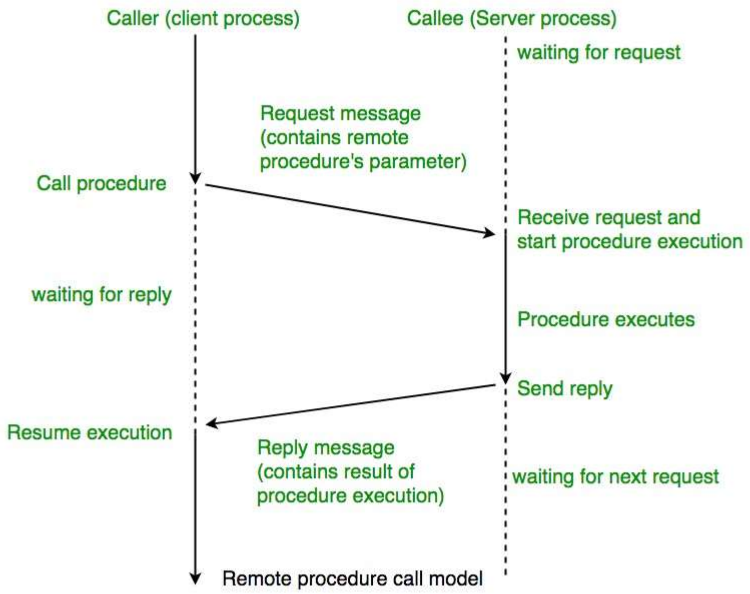


REMOTE PROCEDURE CALL

- Remote Procedure Call (RPC) is a powerful technique for constructing distributed, client-server based applications.
- It is based on extending the conventional local procedure calling so that the called procedure need not exist in the same address space as the calling procedure.
- Today the most widely used RPC styles are JSON-RPC and XML-RPC.
- Even SOAP can be considered to follow an RPC architectural style.
- The central concept in RPC is the procedure.
- The procedures do not need to run on the local machine, but they can run on a remote machine within the distributed system.
- When using an RPC framework, calling a remote procedure should be as simple as calling a local procedure.

REMOTE PROCEDURE CALL

The image part with relationship ID rId2 was not found in the file.



Remote procedure call model

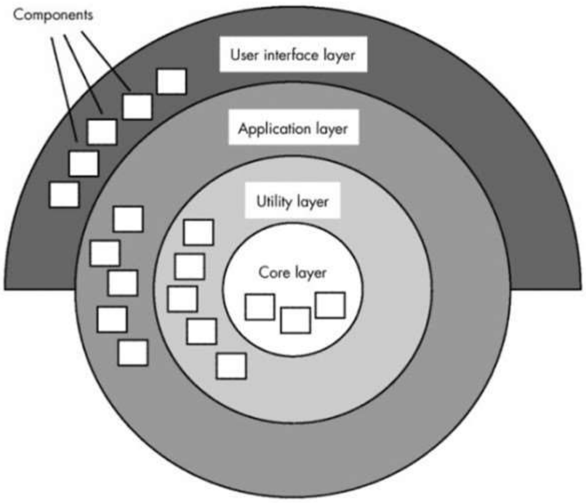
27

LAYERED SYSTEM

- As name indicates the subroutines are organized into number of groups called layers.
- Each layer contains many subroutines which supplies services to the layers above and below it.
- Execution of subroutines is dependent on the subroutine call from its upper layer.
- In some systems only the adjacent layers are connected and the other layers are hidden.
- Extendibility is the prime factor of this style.
- Modifications are simple and easy (Changes in functionality of one layer will only affect the adjacent layers).

28

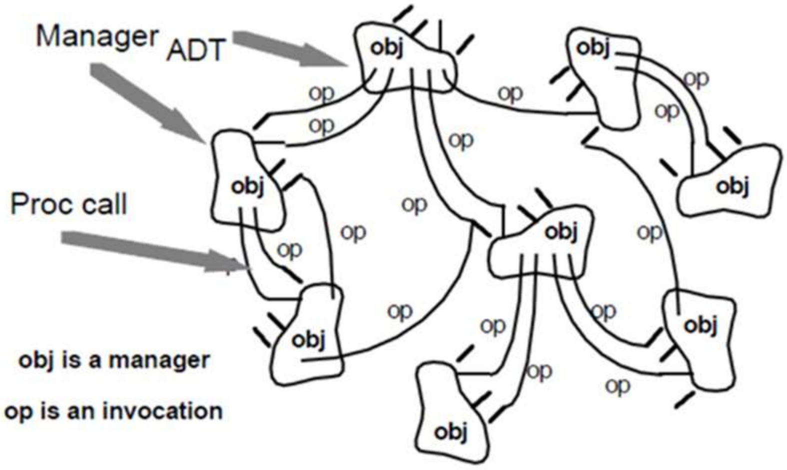
LAYERED SYSTEM



OBJECT ORIENTED

- **Summary:** State strongly encapsulated with functions that operate on that state as objects. Objects must be instantiated before objects' methods can be called.
- **Components:** Objects (aka. Instance of a class).
- **Connector:** Method invocation (procedure calls to manipulate state).
- **Data elements:** Arguments to methods.
- **Topology:** Can vary arbitrarily; components may share data and interface functions through inheritance hierarchies.
- **Additional constraints imposed:** Commonly: shared memory (to support use of pointers).
- **Quality yielded:** Integrity of data operations: data manipulated only by appropriate functions. Abstraction: implementation details hidden.

OBJECT ORIENTED



Manager ADT

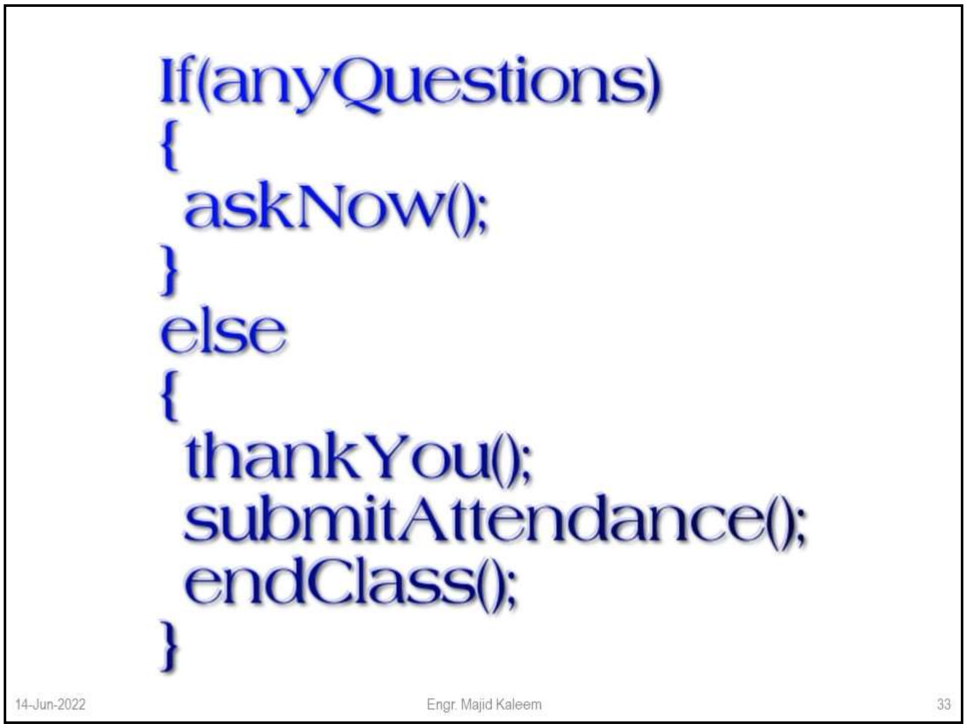
Proc call

obj is a manager

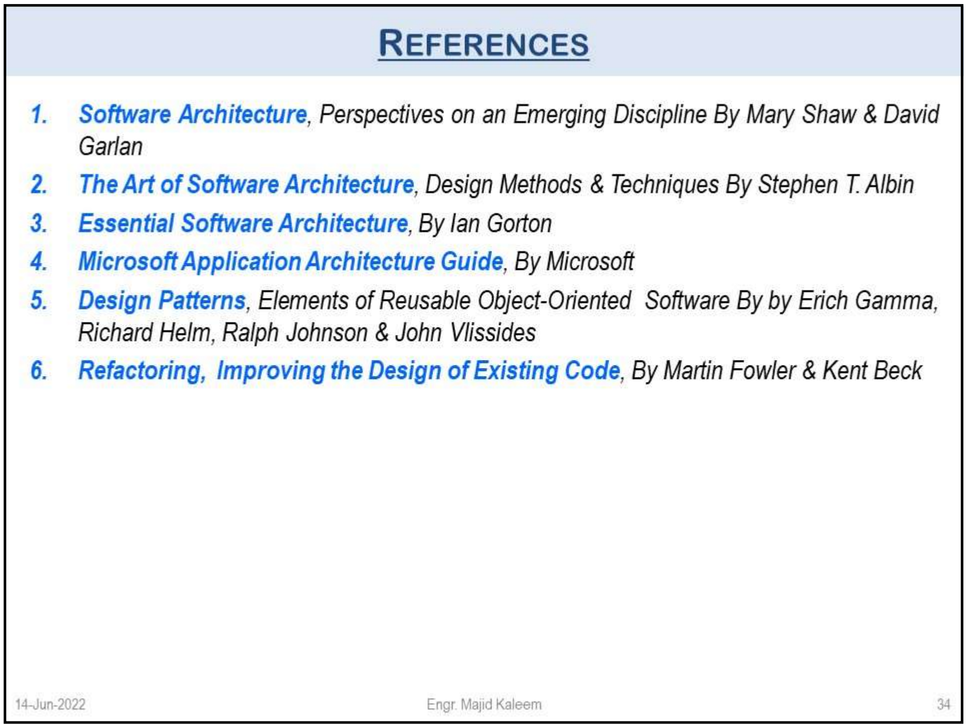
op is an invocation

ARCHITECTURAL QUALITY ATTRIBUTES

- There could be a number of quality attributes, some important are given below:
- **Conceptual Integrity** is the underlying vision or theme unifying the components and their interactions. The architecture should do similar things in similar ways.
- **Correctness and Completeness** is concerned with checking the architecture for errors and omissions.
- **Buildability** is concerned with the organization’s capabilities to actually construct the architecture in question.



```
If(anyQuestions)
{
 askNow();
}
else
{
 thankYou();
 submitAttendance();
 endClass();
}
```



References

1. ***Software Architecture***, *Perspectives on an Emerging Discipline By Mary Shaw & David Garlan*
2. ***The Art of Software Architecture***, *Design Methods & Techniques By Stephen T. Albin*
3. ***Essential Software Architecture***, *By Ian Gorton*
4. ***Microsoft Application Architecture Guide***, *By Microsoft*
5. ***Design Patterns***, *Elements of Reusable Object-Oriented Software By by Erich Gamma, Richard Helm, Ralph Johnson & John Vlissides*
6. ***Refactoring, Improving the Design of Existing Code***, *By Martin Fowler & Kent Beck*