CMR INSTITUTE OF TECHNOLOGY

USN | | | | | | | | | |

Internal Assessment Test 2 – May 2017

Sub:	Software Architectures					**Code:**	10IS81
Date:	08-05-17	Duration:	90 mins	Max Marks: 50	**Sem:** II	**Branch:**	CSE

Note: Answer any 5 questions. All questions carry equal marks. Total marks: 50

		Marks	OBE CO	OBE RBT
1.	Define architectural pattern. Explain dynamics of pipes and filters pattern.	[10]	CO4	L3
2.	Explain the structure and implementation steps of layers pattern.	[10]	CO4	L3
3.	Explain the structure, dynamics and consequences of whole part.	[10]	CO4	L4
4. a.	Explain the variants of proxy.	[06]	CO4	L5
b.	Explain the structure of master slave	[04]	CO4	L3
5.	Explain the structure and implementation steps and consequences of access control	[10]	CO4	L4
6. a.	Explain dynamics of master slave	[06]	CO4	L4
b.	List uses of architecture documentation and explain creating team structure.	[04]	CO5	L2
7.	Explain ADD in detail.	[10]	CO5	L4
8. a.	Explain documentation of views.	[06]	CO5	L3
b.	Explain documentation across views.	[04]		

CMR INSTITUTE OF TECHNOLOGY

SCHEME OF EVALUATION

Internal Assessment Test 1 – March 2017

Sub:	Software Architectures	**Code:**	10IS81
Date: 28/3/2017	Duration: 90 mins	Max Marks: 50	**Sem:** VIII	**Branch:** CSE

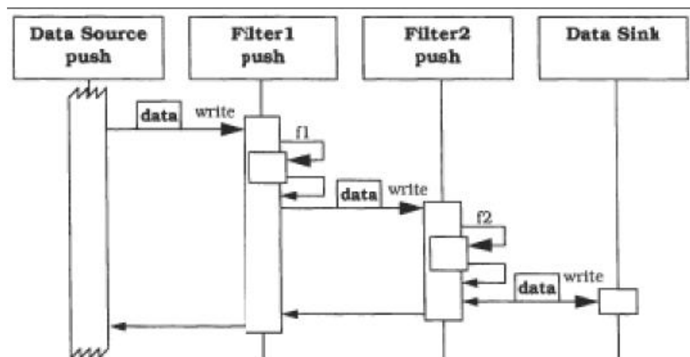
Note: Answer any 5 questions. Total marks: 50

Question No		Description	Distribution of Marks		Total Marks
1.		Definition of architectural pattern. 4 dynamics of pipes and filters pattern.	2M (2*4)M	10M	10M
2.		Structure of layers(Diagram and Explanation) Implementation Steps with explanation	5M 5M	10M	10M
3.		Structure of whole part Dynamics of whole part Consequences of whole part.	2M 4M 4M	10M	10M
4.	a.	Variants of proxy	(1*6)M	6M	10M
	b.	Structure of Master Slave(Diagram and Explanation)	(2*2)M	4M	
5.		Structure of access control Implementation steps of access control Consequences of access control	4M 2M 4M	10M	10M
6.	a.	Dynamics of master slave(Diagram and Explanation)	(3*2)M	6M	10M
	b.	Uses of architecture documentation Creating team structure	2M 2M	4M	
7.		ADD Definition Sample Input Explanation of Step 1 Explanation of Step 2 Explanation of Step 3	2M 1M 1M 5M 1M	10M	10M
8.	a.	Documentation of Views Diagram Explanation	 2M 4M	6M	10M
	b.	Documentation across Views Diagram Explanation	 2M 2M	4M	

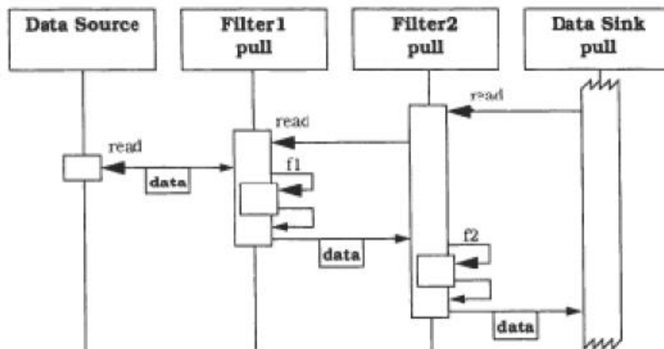
1. Define architectural pattern. Explain dynamics of pipes and filters pattern.
Architectural patterns express fundamental structural organization schemas for software systems. They provide a set of predefined subsystems, specify their responsibilities, and include rules and guidelines for organizing the relationships between them.

PIPES AND FILTERS
The pipes and filter's architectural pattern provides a structure for systems that process a stream of data. Each processing step is encapsulated in a filter component. Data is passed through pipes between adjacent filters. Recombining filters allows you to build families of related systems.

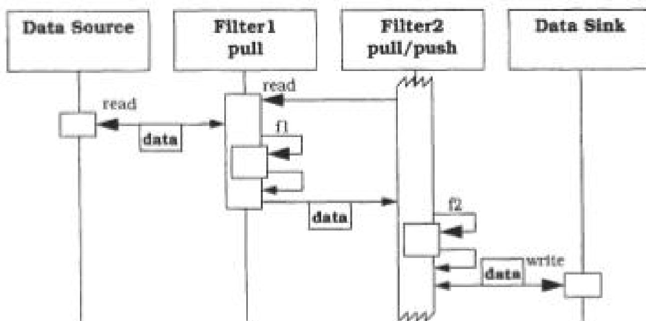
Dynamics:
The following scenarios show different options for control flow between adjacent filters.
Scenario 1 shows a push pipeline in which activity starts with the data source.



Scenario 2 shows a pull pipeline in which control flow starts with the data sink.



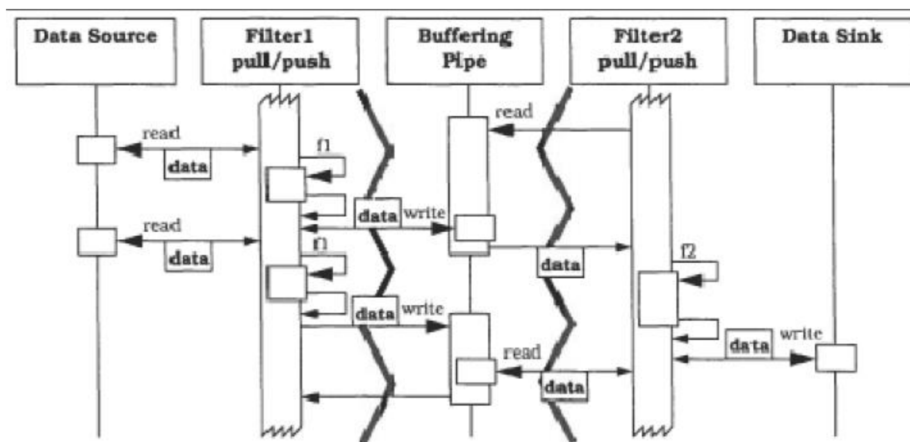
Scenario 3 shows a mixed push-pull pipeline with passive data source and sink. Here second filter plays the active role and starts the processing.



Scenario 4 shows a more complex but typical behavior of pipes and filter system. All filters actively pull, compute, and push data in a loop.

The following steps occur in this scenario:

1. Filter 2 tries to get new data by reading from the pipe. Because no data is available the data request suspends the activity of Filter 2-the buffer is empty.
2. Filter 1 pulls data from the data source and performs function f 1.
3. Filter 1 then pushes the result to the pipe.
4. Filter 2 can now continue, because new input data is available. Filter l can also continue, because it is not blocked by a full buffer within the pipe.
5. Filter 2 computes f 2 and writes its result to the data sink.
6. In parallel with Filter 2's activity, Filter 1 computes the next result and tries to push it down the pipe. This call is blocked because Filter 2 is not waiting for data-the buffer is full.
7. Filter 2 now reads new input data that is already available from the pipe. This releases Filter 1 so that it can now continue its processing



2. Explain the structure and implementation steps of layers pattern.

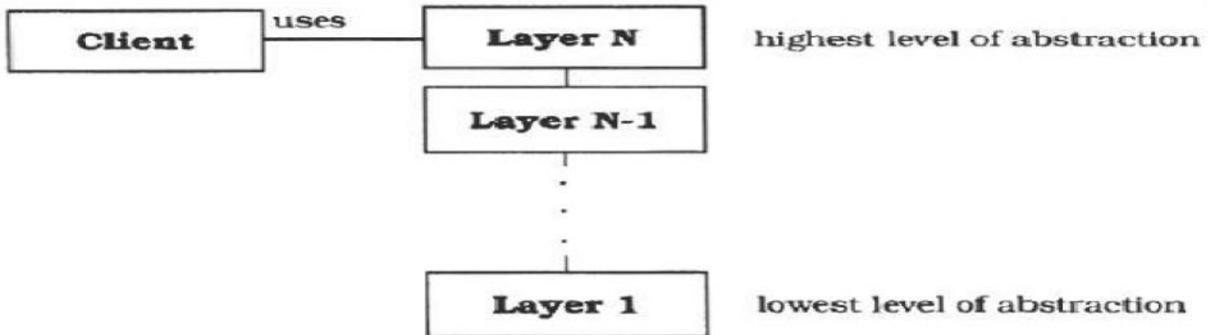
The layers architectural pattern helps to structure applications that can be decomposed into groups of subtasks in which each group of subtasks is at a particular level of abstraction.

Structure:

An individual layer can be described by the following CRC card:

Class Layer J	Collaborator • Layer J - 1
Responsibility • Provides services used by Layer J+1. • Delegates subtasks to Layer J - 1.	

The main structural characteristics of the layers patterns is that services of layer J are only use by layer J+1-there are no further direct dependencies between layers. This structure can be compared with a stack, or even an onion. Each individual layer shields all lower from direct access by higher layers.



Implementation:

The following steps describe a step-wise refinement approach to the definition of a layered architecture.

Define the abstraction criterion for grouping tasks into layers.

o This criterion is often the conceptual distance from the platform (sometimes, we encounter other abstraction paradigm as well).

o In the real world of software development we often use a mix of abstraction criterions. For ex, the distance from the hardware can shape the lower levels, and conceptual complexity governs the higher ones.

Determine the number of abstraction levels according to your abstraction criterion.

o Each abstraction level corresponds to one layer of the pattern.

o Having too many layers may impose unnecessary overhead, while too few layers can result in a poor structure.

Name the layers and assign the tasks to each of them.

o The task of the highest layer is the overall system task, as perceived by the client.

o The tasks of all other layers are to be helpers to higher layers.

Specify the services

o It is often better to locate more services in higher layers than in lower layers.

o The base layers should be kept 'slim' while higher layers can expand to cover a spectrum of applicability.

o This phenomenon is also called the '*inverted pyramid of reuse*'.

Refine the layering

o Iterate over steps 1 to 4.

o It is not possible to define an abstraction criterion precisely before thinking about the implied layers and their services.

o Alternatively it is wrong to define components and services first and later impose a layered structure.

Specify an interface for each layer.

o If layer J should be a 'black box' for layer J+1, design a flat interface that offers all layer J's services

Structure individual layers

o When an individual layer is complex, it should be broken into separate components. o This subdivision can be helped by using finer-grained patterns.

Specify the communication between adjacent layers.

o Push model (most often used): when layer J invokes a service of layer J+1, any required information is passed as part of the service call.

o Pull model: it is the reverse of the push model. It occurs when the lower layer fetches available information from the higher layer at its own discretion.

Decouple adjacent layers.

o For top-down communication, where requests travel top-down, we can use one-way coupling (i.e, upper layer is aware of the next lower layer, but the lower layer is unaware of the identity of its users) here return values are sufficient to transport the results in the reverse direction.
o For bottom-up communication, you can use callbacks and still preserve a top-down one way coupling. Here the upper layer registers callback functions with the lower layer.
o We can also decouple the upper layer from the lower layer to a certain degree using object oriented techniques.

Design an error handling strategy

o An error can either be handled in the layer where it occurred or be passed to the next higher layer. o As a rule of thumb, try to handle the errors at the lowest layer possible.

3. Explain the structure, dynamics and consequences of whole part.

Whole-part design pattern helps with the aggregation of components that together form a semantic unit. An aggregate component, the whole, encapsulates its constituent components, the parts, organizes their collaboration, and provides a common interface to its functionality. Direct access to the parts is not possible.

Structure

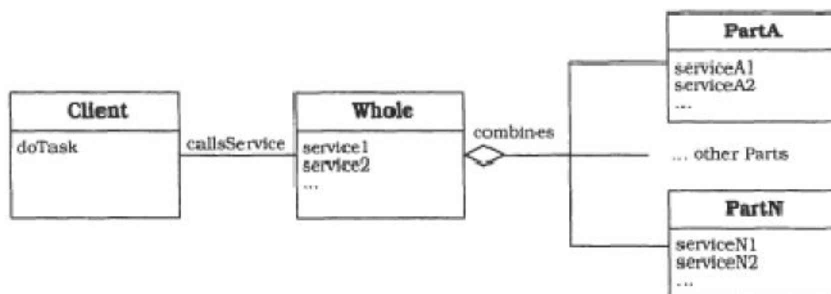
The Whole-Part pattern introduces two types of participant:

Whole

Whole object represents an aggregation of smaller objects, which we call parts. It forms a semantic grouping of its parts in that it co ordinates and organizes their collaboration. Some methods of whole may be just place holder for specific part services when such a method is invoked the whole only calls the relevant part services, and returns the result to the client. Each part object is embedded in exactly one whole. Two or more parts cannot share the same part. Each part is created and destroyed within the life span of the whole.

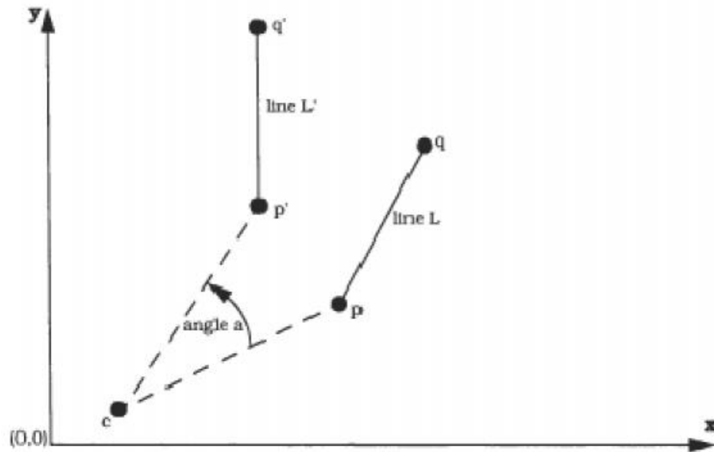
Class Whole	**Collaborators** • Part
Responsibility • Aggregates several smaller objects. • Provides services built on top of part objects. • Acts as a wrapper around its constituent parts.	

Class Part	**Collaborators** -
Responsibility • Represents a particular object and its services.	

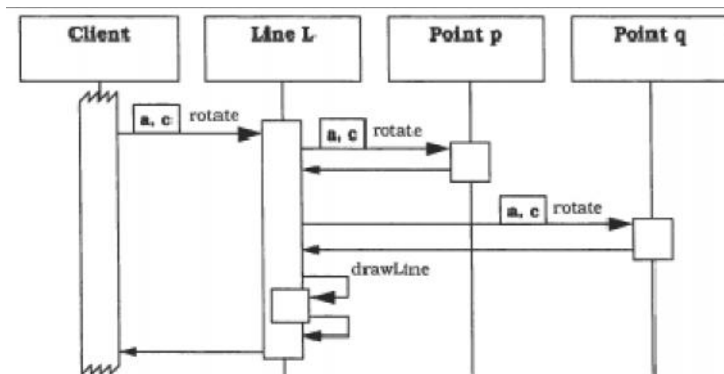


Dynamics:

The following scenario illustrates the behavior of a Whole-Part structure. We use the two-dimensional rotation of a line within a CAD system as an example. The line acts as a Whole object that contains two points p and **q** as Parts. A client asks the line object to rotate around the point c and passes the rotation angle as an argument. The rotation of a point **p** around a center c with an angle a



The scenario consists of four phases:
1. A client invokes the rotate method of the line L and passes the angle a and the rotation center c as arguments.
2. The line L calls the rotate method of the point p.
3. The line L calls the rotate method of the point q.
4. The line L redraws itself using the new positions of p **I** and q **I** as endpoints.



Consequences:
The whole-part pattern offers several *Benefits:*
Changeability of parts:
Part implementations may even be completely exchanged without any need to modify other parts or clients.
Separation of concerns:
Each concern is implemented by a separate part.
Reusability in two aspects:
Parts of a whole can be reused in other aggregate objects
Encapsulation of parts within a whole prevents clients from 'scattering' the use of part objects all over its source code.

The whole-part pattern suffers from the following *Liabilities:*

Lower efficiency through indirection
Since the Whole builds a wrapper around its Parts, it introduces an additional level of indirection between a client request and the Part that fulfils it.
Complexity of decomposition into parts.
An appropriate composition of a Whole from different Parts is often hard to find, especially when a bottom-up approach is applied.

4 a. Explain the variants of proxy.

Remote Proxy
Clients of remote components should be scheduled from network addresses and IPC protocols.

Protection proxy:
Components must be protected from unauthorized access.

Cache proxy:
Multiple local clients can share results from remote components.

Synchronization proxy:
Multiple simultaneous accesses to a component must be synchronized.

Counting proxy:
Accidental deletion of components must be prevented or usage statistics collected

Virtual proxy:
Processing or loading a component is costly while partial information about the component may be sufficient.

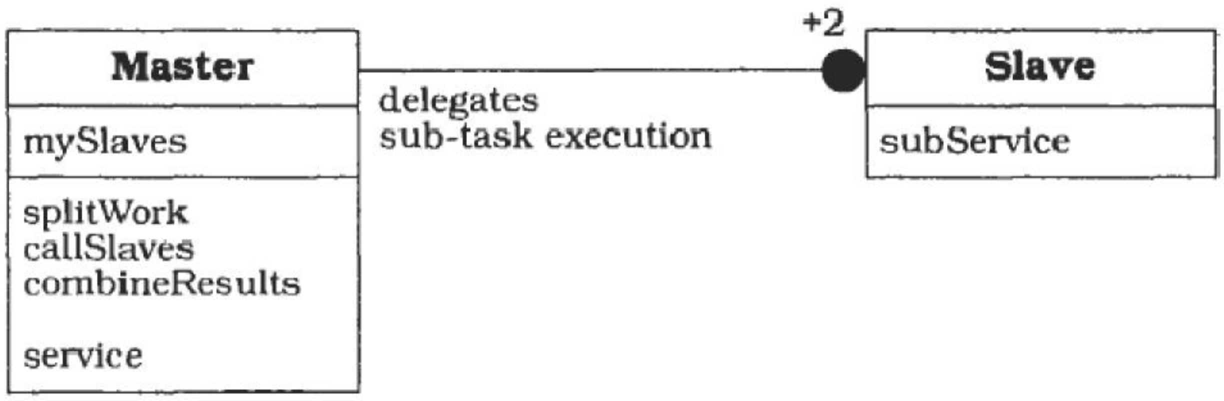
Firewall proxy:
Local clients should be protected from the outside world.

4. b. Explain the structure of master slave
*The **Master-Slave** design pattern supports fault tolerance, parallel computation and computational accuracy. A master component distributes work to identical slave components and computes a final result from the results these slaves return.*
Master component:
Provides the service that can be solved by applying the 'divide and conquer' principle.
It implements functions for partitioning work into several equal subtasks, starting and controlling their processing and computing a final result from all the results obtained.
It also maintains references to all slaves instances to which it delegates the processing of subtasks.
Slave component:
Provides a sub-service that can process the subtasks defined by the master
There are at least two instances of the slave component connected to the master.

Class: Master	Collaborators
Responsibility • Partitions work among several slave components • Starts the execution of slaves • Computes a result from the sub-results the slaves return.	• Slave

Class: Slave	Collaborators
Responsibility • Implements the sub-service used by the master.	-

The structure defined by the Master-Slave pattern is illustrated by the following OMT diagram.



5. Explain the structure and implementation steps and consequences of access control.

Proxy design pattern makes the clients of a component communicate with a representative rather than to the component itself. Introducing such a place holder can serve many purposes, including enhanced efficiency, easier access and protection from unauthorized access.

Structure:

Original

Implements a particular service

Client

Responsible for specific task

To do this, it involves the functionality of the original in an indirect way by accessing the proxy.

Proxy

Offers same interface as the original, and ensures correct access to the original. To achieve this, the proxy maintains a reference to the original it represents.

Usually there is one-to-one relationship b/w the proxy and the original.

Abstract original

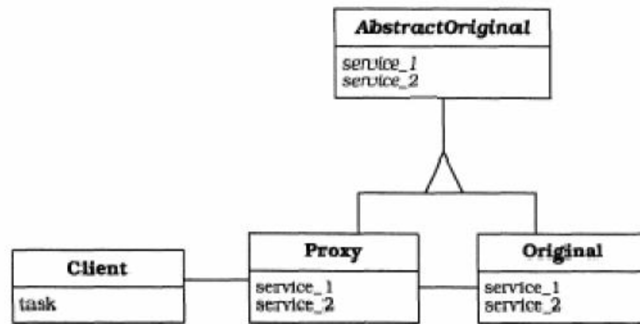
Provides the interface implemented by the proxy and the original. i.e, serves as abstract base class for the proxy and the original.

Class Client	***Collaborators***
Responsibilities • Uses the interface provided by the proxy to request a particular service. • Fulfills its own task.	• Proxy

Class AbstractOriginal	***Collaborators***
Responsibilities • Serves as an abstract base class for the proxy and the original.	-

Class Proxy	***Collaborator***
Responsibilities • Provides the interface of the original to clients. • Ensures a safe, efficient and correct access to the original.	• Original

Class Original	***Collaborators***
Responsibilities • Implements a particular service.	-



Implementation:

1. Identify all responsibilities for dealing with access control to a component Attach these responsibilities to a separate component the proxy.

2. If possible introduce an abstract base class that specifies the common parts of the interfaces of both the proxy and the original.

Derive the proxy and the original from this abstract base. **3. Implement the proxy's functions**

To this end check the roles specified in the first step

4. **Free the original and its client** from responsibilities that have migrated into the proxy.

5. **Associate the proxy and the original** by giving the proxy a handle to the original. This handle may be a pointer a reference an address an identifier, a socket, a port, and so on.

6. Remove all direct relationships between the original and its client Replace them by analogous relationships to the proxy.

Consequences:

The Proxy pattern provides the following *Benefits:*

Enhanced efficiency and lower cost

The Virtual Proxy variant helps to implement a 'load-on-demand' strategy. This allows you to avoid unnecessary loads from disk and usually speeds up your application

Decoupling clients from the location of server components

By putting all location information and addressing functionality into a Remote Proxy variant, clients are not affected by migration of servers or changes in the networking infrastructure. This allows client code to become more stable and reusable.

Separation of housekeeping code from functionality.

A proxy relieves the client of burdens that do not inherently belong to the task the client is to perform.

The Proxy pattern has the following *Liabilities:*

Less efficiency due to indirection

All proxies introduce an additional layer of indirection.

Over kill via sophisticated strategies

Be careful with intricate strategies for caching or loading on demand they do not always pay.

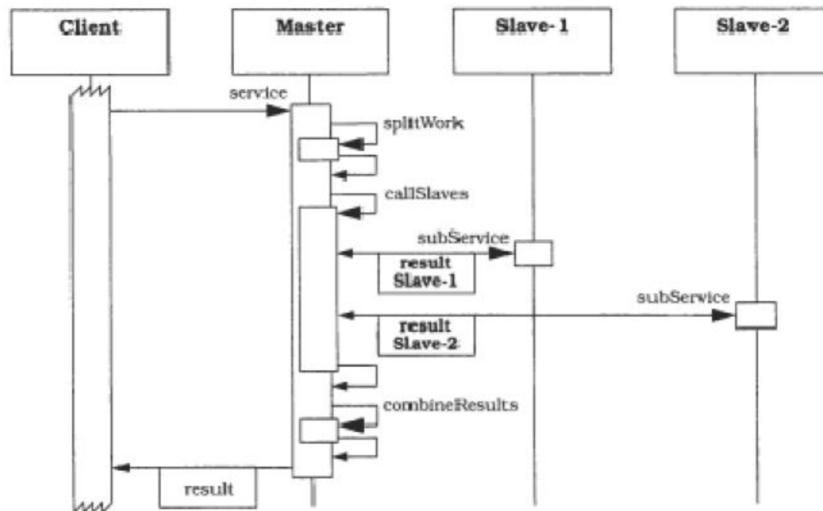
---

6. a. Explain dynamics of master slave

The scenario comprises six phases:

- A client requests a service from the master.
- The master partitions the task into several equal sub-tasks.
- The master delegates the execution of these sub-tasks to several slave instances, starts their execution and waits for the results they return.
- The slaves perform the processing of the sub-tasks and return the results of their computation back to the master.
- The master computes a final result for the whole task from the partial results received from the slaves.

- The master returns this result to the client.



6. b. List uses of architecture documentation and explain creating team structure
Uses of Architecture Documentation:
1. Architecture documentation is both prescriptive and descriptive.
2. All of this tells us that different stakeholders for the documentation have different needs—different kinds of information, different levels of information, and different treatments of information.
3. Documentation facilitates that communication. Some examples of architectural stakeholders and the information they might expect to find in the documentation.
4. Documentation that was easy to write but is not easy to read will not be used, and "easy to read" is in the eye of the beholder—or in this case, the stakeholder.
Forming team structure
Once the architecture is accepted we assign teams to work on different portions of the design and development.

- Once architecture for the system under construction has been agreed on, teams are allocated to work on the major modules and a work breakdown structure is created that reflects those teams.
- Each team then creates its own internal work practices.
- For large systems, the teams may belong to different subcontractors.
- Teams adopt “work practices’ including
  - Team communication via website/bulletin boards o Naming conventions for files
  - Configuration/revision control system
  - o Quality assurance and testing procedure

The teams within an organization work on modules, and thus within team high level of communication is necessary.

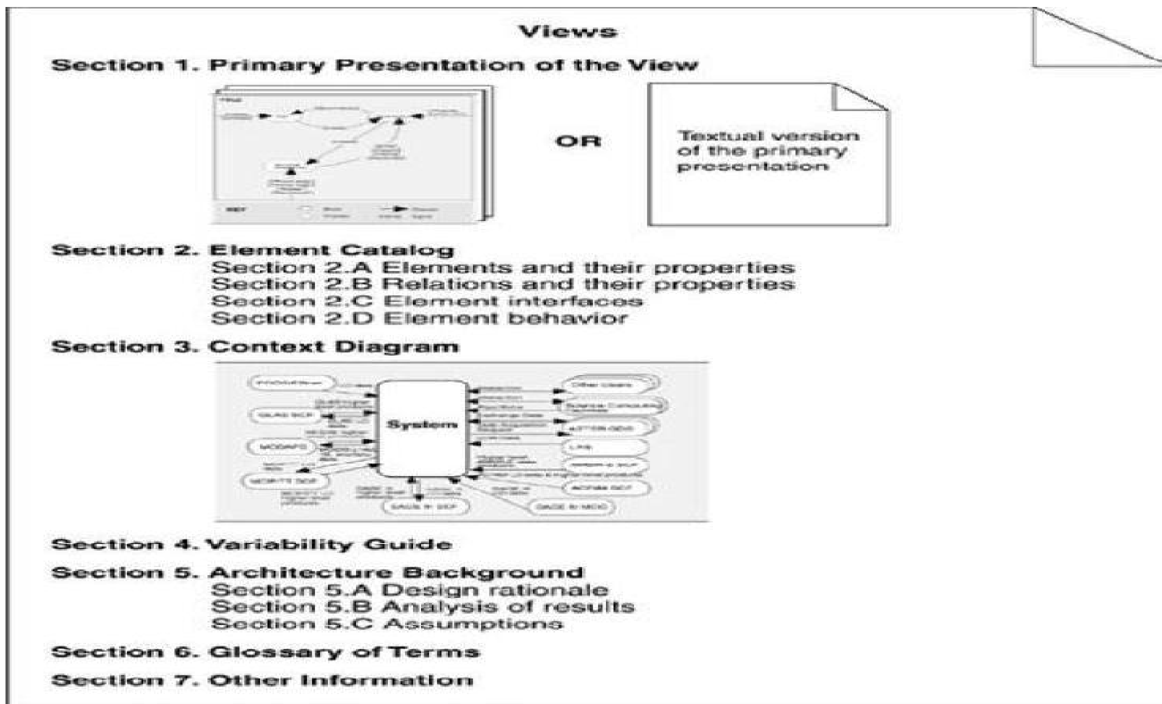
---

7. Explain ADD in detail.
A method for designing an architecture to satisfy both quality requirements and functional requirements is called attribute-driven design (ADD). ADD takes as input a set of quality attribute scenarios and employs knowledge about the relation between quality attribute achievement and architecture in order to design the architecture.
The output of ADD is the first several levels of a module decomposition view of an architecture and other views as appropriate.
Sample Example
Design a product line architecture for a garage door opener with a larger home information system the opener is responsible for raising and lowering the door via a switch, remote control, or the home

information system. It is also possible to diagnose problems with the opener from within the home information system.
o **Input** to ADD: a set of requirements
o Functional requirements as use cases
o Constraints
o Quality requirements expressed as system specific quality scenarios
o **Scenarios** for garage door system
o Device and controls for opening and closing the door are different for the various products in the product line
o The processor used in different products will differ
o If an obstacle is (person or object) is detected by the garage door during descent, it must stop within 0.1 second
o The garage door opener system needs to be accessible from the home information system for diagnosis and administration.
o It should be possible to directly produce an architecture that reflects this protocol

ADD STEPS
Steps involved in attribute driven design (ADD)
1. Choose the module to decompose
o Start with entire system
o Inputs for this module need to be available
o Constraints, functional and quality requirements
2. Refine the module
a) Choose architectural drivers relevant to this decomposition b) Choose architectural pattern that satisfies these drivers
c) Instantiate modules and allocate functionality from use cases representing using multiple views d) Define interfaces of child modules
e) Verify and refine use cases and quality scenarios
3. Repeat for every module that needs further decomposition

***8. a.* Explain documentation of views.**
Primary presentation- elements and their relationships, contains main information about these system , usually graphical or tabular.
Element catalog- details of those elements and relations in the picture,
Context diagram- how the system relates to its environment
Variability guide- how to exercise any variation points a variability guide should include documentation about each point of variation in the architecture, including
o The options among which a choice is to be made
o The binding time of the option. Some choices are made at design time, some at build time, and others at runtime.
Architecture background –why the design reflected in the view came to be? an architecture background includes
o rationale, explaining why the decisions reflected in the view were made and why alternatives were rejected
o analysis results, which justify the design or explain what would have to change in the face of a modification
o assumptions reflected in the design
Glossary of terms used in the views, with a brief description of each.
Other information includes management information such as authorship, configuration control data, and change histories. Or the architect might record references to specific sections of a requirements document to establish traceability.


Views

Section 1. Primary Presentation of the View

OR

Textual version of the primary presentation

Section 2. Element Catalog
Section 2.A Elements and their properties
Section 2.B Relations and their properties
Section 2.C Element interfaces
Section 2.D Element behavior

Section 3. Context Diagram

System

Section 4. Variability Guide

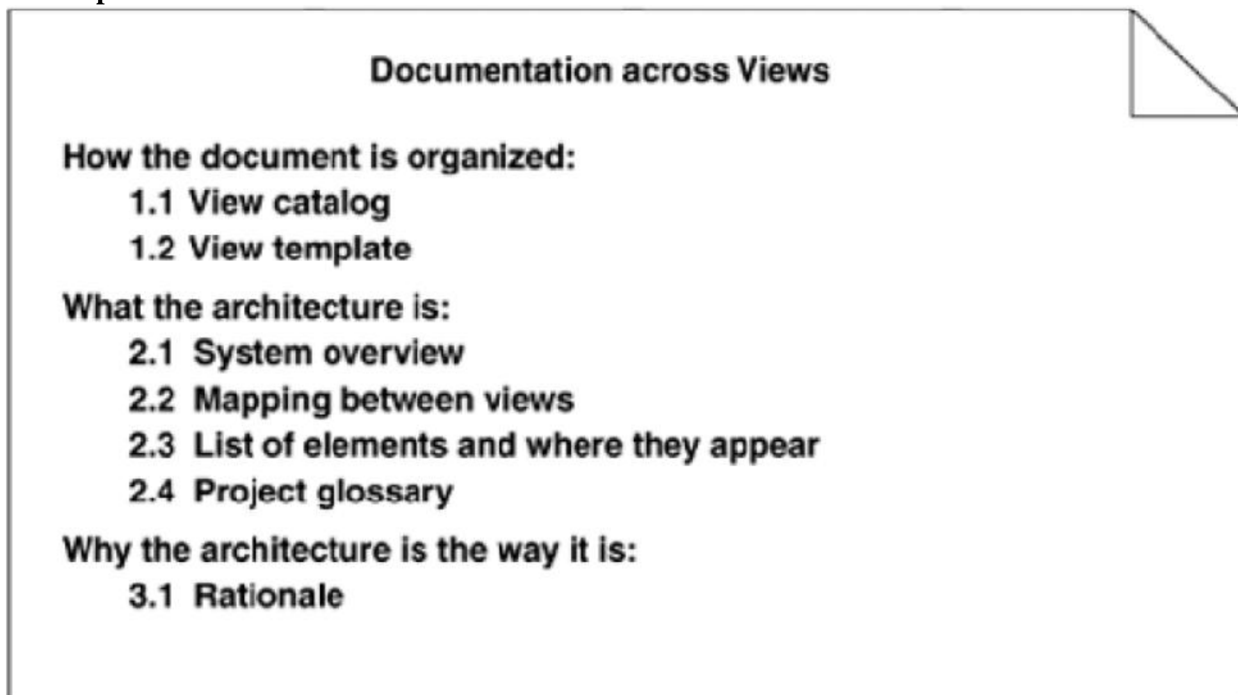
Section 5. Architecture Background
Section 5.A Design rationale
Section 5.B Analysis of results
Section 5.C Assumptions

Section 6. Glossary of Terms

Section 7. Other Information

Source: Adapted from [Clements 03].

8. b. Explain documentation across views.


Documentation across Views

How the document is organized:
1.1 View catalog
1.2 View template

What the architecture is:
2.1 System overview
2.2 Mapping between views
2.3 List of elements and where they appear
2.4 Project glossary

Why the architecture is the way it is:
3.1 Rationale

Source: Adapted from [Clements 03].

HOW THE DOCUMENTATION IS ORGANIZED TO SERVE A STAKEHOLDER

Every suite of architectural documentation needs an introductory piece to explain its organization to a novice stakeholder and to help that stakeholder access the information he or she is most interested in.

View Catalog

A view catalog is the reader's introduction to the views that the architect has chosen to include in the suite of documentation.

View Template
A view template is the standard organization for a view. It helps a reader navigate quickly to a section of interest, and it helps a writer organize the information and establish criteria for knowing how much work is left to do.

WHAT THE ARCHITECTURE IS
This section provides information about the system whose architecture is being documented, the relation of the views to each other, and an index of architectural elements.

System Overview
This is a short prose description of what the system's function is, who its users are, and any important background or constraints.

Mapping between Views
Since all of the views of an architecture describe the same system, it stands to reason that any two views will have much in common.

Element List
The element list is simply an index of all of the elements that appear in any of the views, along with a pointer to where each one is defined. This will help stakeholders look up items of interest quickly.

Project Glossary
The glossary lists and defines terms unique to the system that have special meaning. A list of acronyms, and the meaning of each, will also be appreciated by stakeholders. If an appropriate glossary already exists, a pointer to it will suffice here.

WHY THE ARCHITECTURE IS THE WAY IT IS: RATIONALE
Cross-view rationale explains how the overall architecture is in fact a solution to its requirements. One might use the rationale to explain:

- The implications of system-wide design choices on meeting the requirements or satisfying constraints.
- The effect on the architecture when adding a foreseen new requirement or changing an existing one.
- The constraints on the developer in implementing a solution.
- Decision alternatives that were rejected.

In general, the rationale explains why a decision was made and what the implications are in changing it.