1. Introduction

A. Define

1. UML-

The Unified modeling Language (UML) is a language for specifying, visualizing, constructing, & documenting the artifacts of s/w systems as well as for bulsness modeling & other non-s/w systems.

UML building blocks

① Things ② Relationships ③ Diagrams.

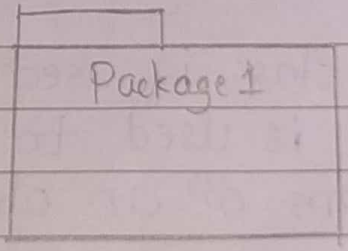
(Grouping things can be defined as a mechanism to group elements of UML model together)

2. Package-

① Package is the only one grouping thing available for gathering structural & behavioural things.

② It is the package which is used to group semantically related modeling elements into a single cohensive unit.

notation-



Things- ① Structural ② Behavioural
③ Grouping ④ Annotational

Page No.
Date

3. Annotational thing-

① Annotational things can be defined as a mechanism to capture remarks, descriptions & comments of UML model elements.

② In any diagram, explanation of different elements & their functionalities are very important.

B. Answer the following.

1. Explain structural Things.

→ ① A structural thing is used to describe the static part of model.

② They represent the physical & conceptual elements.

③ It is used to represent the things that are visible to human eyes.

④ Structural things are all about the physical part of the system.

structural things consist of:

① Class- A class is used to represent various objects. It is used to define the properties & operations of an object.

Class Notation -

Class Name
Attributes
Operations

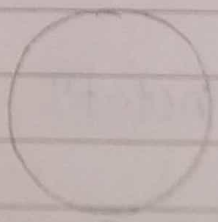
② Object - An object is an entity which is used to describe the behaviour & functions of a system. The class & object have the same notations. The only difference is that an object name is always underlined in UML.

Object Notation -

Object-name
+object-attributes

③ Interface - Interface is just like a template where you define different functions, not the implementat^{n}. When a class implements an interface, its functionality is also implemented. A circle notation represents it. It has a name which is generally written below the circle

Interface Notation -

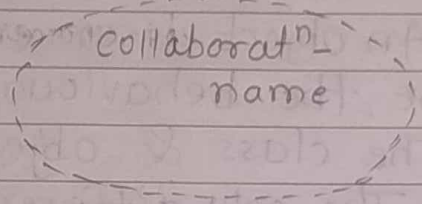


Interface-name

Page No.
Date

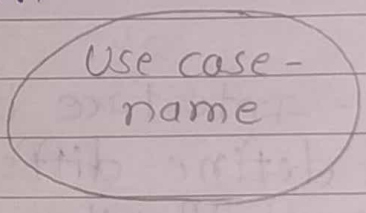
④ Collaboration - Collaboration defines an interaction between elements. Collaboration represents responsibilities. It is represented by a dotted ellipse with a name written inside it.

UML notation-



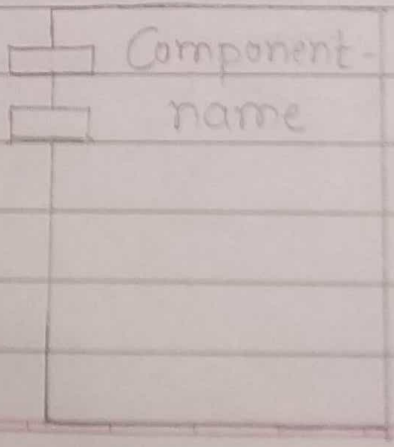
⑤ Use-case - Use case represents a set of actions performed by a system for a specific goal. They are used to represent high-level functionalities & how the user will handle the system.

Use-case notation-



⑥ Component - Component describes the physical part of a system. component is used to represent any part of a system. Additional elements can be added wherever required.

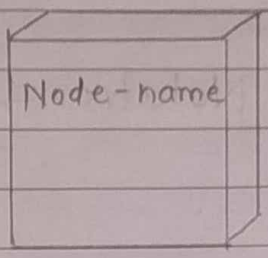
Component notat^{n}-



Page No.
Date

⑦ Node - A node can be defined as a physical element that exists at run time. It represents the physical part of a system such as the server, network, router etc.

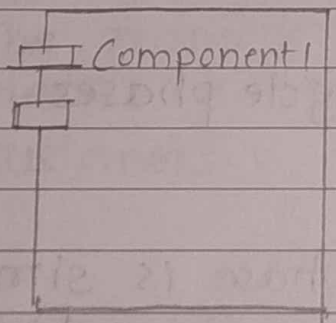
Node notation-

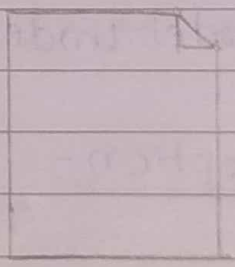


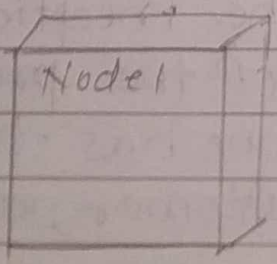
⑧ Deployment diagram - It represents the physical h/w on which system is installed. A deployment diagram represents the physical view of a system. It denotes the communication & interaction betⁿ various parts of the system.

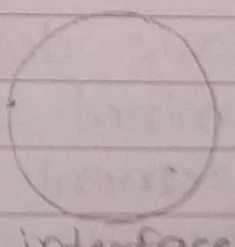
deployment diagram consist following notations -

a. A node
b. A component
c. An artifact
d. An interface.









Page No.
Date

⑨ Active Class Notatⁿ - Active class looks similar to a class with a solid border. Active class is generally used to describe the concurrent behaviour of a system. Active class is used to represent the concurrency in a system.

Active Class Notation -

Class Name
Attributes
Operations

2. What is Unified process & Explain the phases of Unified Process (UP).

→ The Unified Process (UP) combines commonly accepted best practices, such as an iterative lifecycle & risk-driven development into a cohensive & well-documented descriptⁿ. Unified process is based on the enlargement & refinement of a system through multiple iterations, with cyclic feedback & adaptation.

UP model undergoes 4 life cycle phases -

1. Inception -

The inception phase is similar to the requirements collection & analysis stage of the S/W model. During the inception phase, the basic idea & structure of the project is determined. In this phase, one has to collect requirements from the customer, analyze the project's feasibility.

2. Elaboration -

In this phase, you'd be expanding upon the activities undertaken in the inception phase. The major goals of this phase include creating fully functional requirements (use-cases) & creating a detailed architecture for fulfillment of the requirements.

3. Construction -

In this phase, you will be writing actual code & implementing the application features for each iteration. This period is also where integrations with other services or existing s/w should occur.

4. Transition -

In this phase, you would be rolling out the next iterations to the customer & fixing bugs for previous releases. You would also deploy builds of the s/w to the customer. The transition phase is when the finished product is finally released & delivered to customers.

3. Explain types Relationship in details.

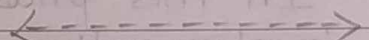
→

① Association Relationship -

Association is used to represent the relationship between two elements of a system. Association is basically a set of

links that connects the elements of a UML model. It also describes how many objects are associated i.e. how many elements are taking part in that interaction. Association is represented by a dotted line with (without) arrows on both sides. the multiplicity is also mentioned at the end (1, *, etc.) to show how many objects are associated.

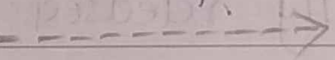
Association relationship is denoted as follows,



② Dependency Relationship -

Dependency is a relationship between two things in which change in one element also affects the other. It describes the dependent elements & the direct^{n} of dependency. Dependency is represented by a dotted arrow & the arrow head represents the independent element & the other end represents the dependent element.

Dependency relationship is denoted as follows,

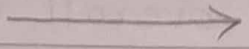


③ Generalization Relationship -

Generalization can be defined as a relationship which connects a specialized element with a generalized element. It basically describes the inheritance relationship in the object oriented world. It is a parent & child relationship. Generalizat^{n} is represented by an arrow with a hollow arrow head.

Page No.
Date

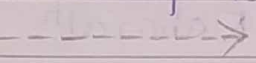
Generalizatn relationship is denoted as follows -



④ Realization Relationship -

Realizatn can be defined as a relationship in which two elements are connected. Realization relationship is widely used while denoting interfaces. It is denoted as a dotted line with a hollow arrowhead at one end.

Realization relationship is denoted as follows -



⑤ Extensibility -

All the languages (programming or modeling) have some mechanism to extend its capabilities such as syntax, semantics, etc. UML also has the following mechanisms to provide extensibility features. extensibility notatn are used to ~~represent some extra behaviour of the system.~~ enhance the power of language.

7. How are diciplines & phases of UP related.

→

Disciplines -

1. Buisness modeling - domains object modelling & dynamic modeling of the business processes.
2. Requirements - requirements analysis of system under consideration. Includes activities like writing use cases & identifying nonfunctional requirements.

3. Analysis & design - covers aspects of design, including the overall architecture.
4. Implementat^{n} - programming & building the system (except the deployment).
5. Test - Involves testing activities such as test planning, development of test scenarios, alpha & beta testing, regression testing, acceptance testing.
6. Deployment - The deployment activities of developed system.

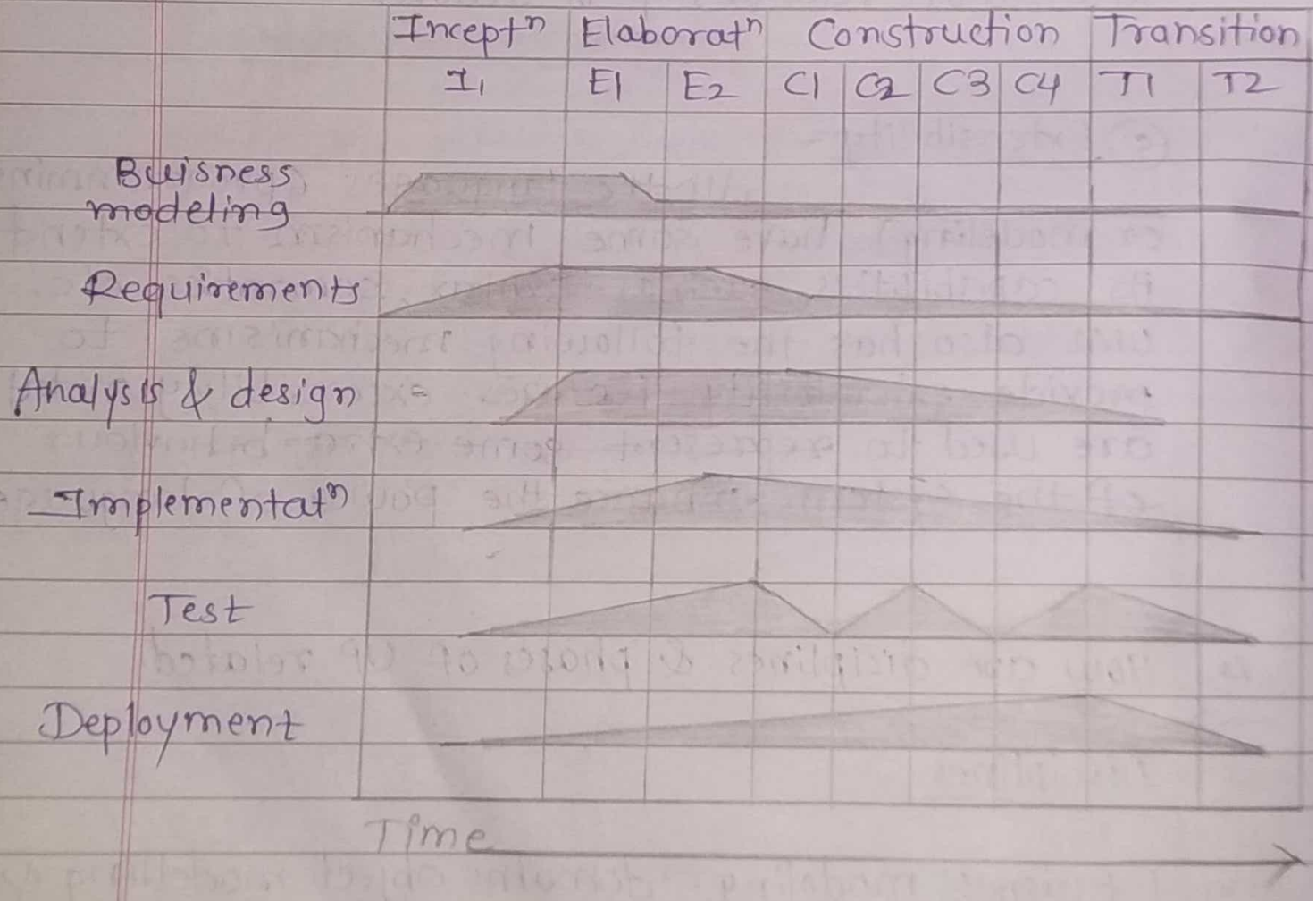


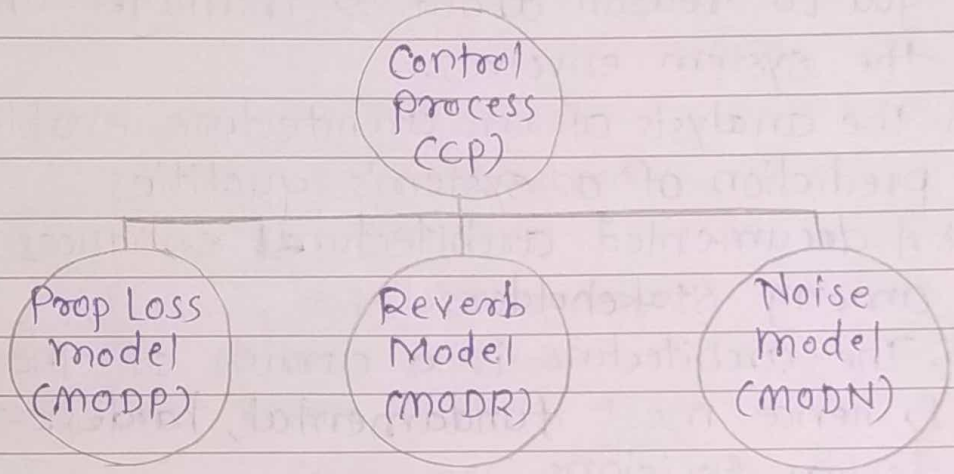
fig: Disciplines & phases of UP

2. Software Architecture

A. Define.

1. Software Architecture-

The s/w Architecture of a program or computing system is the structure or structures of the system which comprises of s/w element, the externally visible properties of those elements & the relationship among them.



2. Structure-

Structure is the set of elements itself, as they exist in s/w or h/w & View is representation of a set of elements & relations among them.

3. View-

A view is a representation of a coherent set of architectural element, as written by & read by system stakeholders.

B. Answer the following.

1. Why is S/W Architecture important ?

→ SA is important for a wide variety of technical & non technical reasons.

Below are some reasons which states its importance :

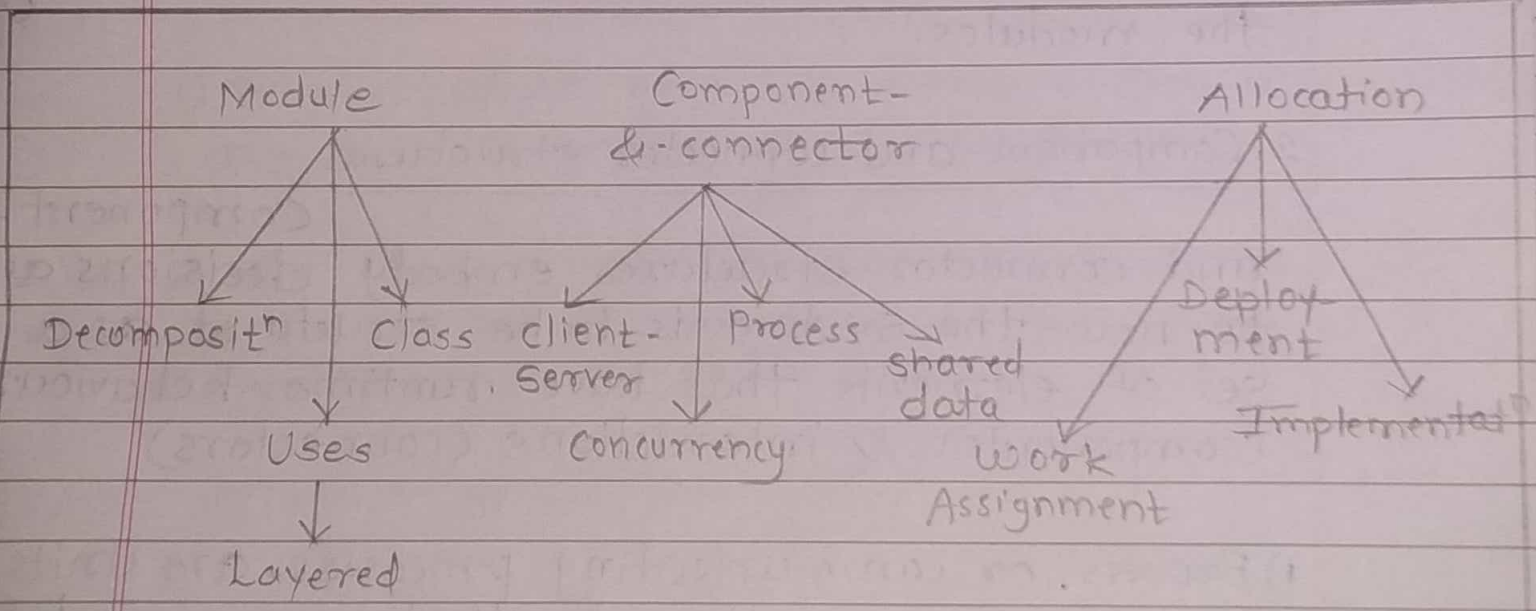
1. An architecture will inhibit or enable a system's driving quality attributes.
2. The decisions made in an architecture allow you to reason about & manage change as the system envolves.
3. The analysis of an architecture enables early prediction of a system's qualities.
4. A documented architecture enhances commⁿ among stakeholders.
5. The architecture is a carrier of the earliest & hence most fundamental, hardest-to-change design decisions.
6. It defines a set of constraints on subsequent implementation.
7. It dictates the structure of an organization, or vice versa
8. It can provide the basis for evolutionary prototyping.
9. It is the key artifact that allows the architect & project manager to reason about cost & schedule.
10. It can be created as a transferable, reusable model that forms the heart of a product line

11. Architecture-based development focuses attention on the assembly of components, rather than simply on their creation.
12. It channels the ~~a~~creativity of developers, reducing design & system complexity.
13. It can be the foundation for training of a new team member.

2. Classify software structures.

→

Software Structures:



1. Module structures -

Modules structures embody decisions as to how the system is to be structured as a set code or data units that have to be constructed or procured. In any module structure, the elements are modules

of some kind. Modules represent a static way of considering the system.

1) Decomposition - shows how larger modules are decomposed into smaller ones recursively.
2) Uses - The units are modules, procedures or resources on the interfaces of modules & are related by the uses relation.
3) Layered - uses relations" structured into Layers.
4) Class, or generalization - shows the "inherits-from" or "is-an-instance-of" relations among the modules.

2. Component-and-connector structures -

Component-and-connector structures embody decisions as to how the system is to be structured as a set of elements that have runtime behaviour (components) & interactions (connectors).

1) Process, or communicating processes - are units are processes or threads that are connected with each other by communication, synchronization, and/or exclusion operations.

2) Concurrency - The units are component & the connectors are "logical threads" where a logical thread is a sequence of computatn that can be allocated to a separate physical thread.

3) shared data, or repository - This structure comprises components & connectors that create, store, & access persistent data.

4) Client-server - The components are the clients & servers, & the connectors are protocols & messages.

3. Allocation Structures -

Allocation structures embody decisions as to how the system will relate to non-s/w structures in its environment.

1) Implementatn - How s/w elements (usually modules) are mapped to the file structure(s).

2) Work assignment - Assigns responsibility for implementing & integrating the modules to development teams.

4. Introduction to Patterns

1. Define Pattern & Design Pattern.

→

① Pattern –

"Each pattern is a three-part rule, which expresses a relation between a certain context, a problem, & a solution.

or

The pattern is, in short, at the same time a thing, which happens in the world; & the rule which tells us how to create that thing & when we must create it.

It is both a process & a thing: both description of a thing which is alive, & a description of the process which will generate that thing.

② Design Pattern –

The design patterns ~~describe object-oriented designs, they are~~ are "descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context"

2. What is GOF?

→ These authors are collectively known as "The Gang of Four (GOF)". According to these authors design patterns are primarily based on the following principles of object oriented design.

1. Program to an interface not an implementation
2. Favour object composition over inheritance.

GOF design pattern are divided into 3 categories-
1. Creational 2. Structural 3. Behavioral

3) Describe a template of Design Pattern.

→

① Pattern name -

The pattern's name & short summary of pattern conveys the essence of the pattern. A good name is important, because it will become part of your design vocabulary.

② Intent -

A situation in which the pattern may apply. A short statement that answers the following questions:
1. What does the design pattern do &
2. What particular design issue or problem does it address &

③ Also Known As -

Other well-known names for the pattern, if any.

④ Motivation -

A scenario that illustrates a design problem & how the class & object structures in the pattern solve the problem.

⑤ Applicability -
What are the situations in which the design pattern can be applied & How can you recognize these situations ?

⑥ Structure -
A graphical representation of the classes in the pattern using a notation based on the Object Modeling Technique (OMT)

⑦ Participants -
The classes &/or objects participating in the design pattern & their responsibilities.

⑧ Collaboration -
Shows how the participants collaborate to carry out their responsibilities.

4) Classify Pattern.

→

We classify patterns into 3 categories -

1) Architectural Patterns -
① An architectural pattern expresses a fundamental structural organization schema for s/w systems.
② It provides a set of predefined subsystems, specifies their responsibilities, & includes rules & guidelines for organizing the relationships between them.

③ Example - Model-View-Controller pattern. It provides a structure for interactive software systems.

2) Design Patterns -

① A design pattern provides a scheme for refining the subsystems or components of a s/w system, or the relationships betⁿ them.
② It describes a commonly-recurring structure of communicating components that solves a general design problem within particular context.
③ Design patterns are medium-scale patterns.

3) Idioms -

① An idiom is a low-level pattern specific to a programming language.
② An idiom describes how to implement particular aspects of components or the relatⁿships between them using the features of the given language.
③ Idioms address aspects of both design and implementation.
④ Example - The C++ community uses reference-counting idioms to manage dynamically allocated resources; Smalltalk provides a garbage collection mechanism, so has no need for such idioms.

5) Give different types of Behavioral patterns

→

There are 11 behavioural design patterns defined in the GOF design patterns.

① Template Method - Used to create a template method stub & defer some of the steps of implementation to the subclasses.

- ② Mediator - Used to provide a centralized communication medium between different objects in a system.

③ Chain of Responsibility - Used to achieve loose coupling in s/w design where a request from the client is passed to a chain of objects to process them.

- ④ Observer - Useful when you are interested in the state of an object & want to get notified whenever their is any change.

- ⑤ Strategy - Strategy pattern is used when we have multiple algorithm for a specific task & client decides the actual implementation to be used at runtime.

- ⑥ Command - Command pattern is used ~~to when an object to~~ implement lose coupling in a request-response model.

(7) State - State design pattern is used when an object change it's behaviour based on it's internal state

(8) Visitor - Visitor pattern is used when we have to perform an operat" on a group of similar kind of Objects.

(9) Interpreter - defines a grammatical representation for a language & provides an interpreter to deal with this grammar.

(10) Iterator - Used to provide a standard way to traverse through a group of objects.

(11) Memento - The memento design pattern is used when we want to save the state of an object so that we can restore later on.

6) How we can use patterns in s/w architecture ?

→ following are few factors which show how we can use patterns in s/w architecture.

1) Patterns & mental building blocks -

(1) Patterns are useful building blocks for dealing with limited & specific aspects when developing a s/w system.

(2) S/w architecture techniques such as inheritance & polymorphism, do not address the solution of specific problems.

③ Most of the existing analysis & design methods also fail at this level.

2) Constructing Heterogeneous Architecture -

① A pattern system describes patterns uniformly, classifies them & shows how they are interwoven with each other.

② Patterns systems also help you to find the right pattern to solve a problem or to identify alternative solutions to it.

③ Pattern systems help us to use the power that the entirely of patterns provide

3) Patterns versus Methods -

A good pattern description includes guidelines for its implementation that you can consider as a micro-method for creating the solutn to a specific problem.

4) Implementing patterns -

① Another aspect that arises from the integration of patterns with s/w architecture is a paradigm for implementing them.

② Many current s/w patterns are tempting to conclude that the only way we can implement a pattern effectively is in an object-oriented programming language.

7) How to prepare a pattern ?

→

3. Architecture Styles

1) Define Architectural style.

→ ① An Architectural style defines a family of such systems in terms of a pattern of structural organization.

② An architectural style determines the vocabulary of components & connectors that can be used in instances of that style, together with a set of constraints on how they can be combined.

2) Describes pipes & filters in detail.

→ ① In a pipes & filter style each component has a set of inputs & a set of outputs.

② A component reads streams of data on its inputs & produces streams of data on its outputs, delivering a complete instance of the result in a standard order.

③ This is usually accomplished by applying a local transformation to the input streams & computing incrementally so output begins before input is consumed. Hence, components are termed "filters".

④ The connectors of this style serve as conduits for the streams, transmitting outputs of one filter to inputs of another. Hence the connectors are termed "pipes".

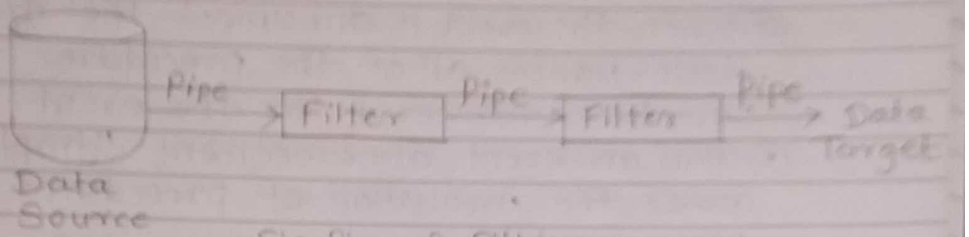


Fig Pipes & Filters

Advantages -

① They support reuse
② They naturally support concurrent execution

Disadvantages -

① They may be hampered by having to maintain correspondences betⁿ two separate, but related streams.
② Systems may not be appropriate for long-running computations.

3) What is Implicit Invocation ?

→ ① Traditionally, in a system in which the component interfaces provide a collection of procedures & functions, components interact with each other by explicitely invoking those routines.
② The idea behind the implicit invocation is that instead of invoking a procedure directly, a component can announce one or more events.
③ Other components in the system can register an interest in an event by associating a procedure with the event.

④ When the event is announced the system itself invokes all of the procedures that have been registered for the event.
⑤ Thus an event announcement "implicitely" causes the invocation of procedures in other modules.

Advantages-
① It provides strong support for reuse.
② Implicit invocation eases system evolution.

Disadvantages-
① Components relinquish control over the computation performed by the system.
② Another problem is with exchange of data.

4) Write a short note on Blackboard model.
→ ① Blackboard architecture style is an artificial intelligence approach which handles complex problem, where the solution is the sum of its parts.
② Blackboard architecture style has a blackboard component which acts as a central data repository.
③ It is used in location-locomotion, data interpretation & environmental changes for solving the problem.
④ This model is usually presented with 3 major parts:

(i) The knowledge sources (KS) ⇒ Separate independent parcels of application dependent knowledge.
(ii) The blackboard Data Structure ⇒ problem-solving state data, organized into an application-dependent hierarchy.
(iii) Control ⇒ driven entirely by state of blackboard

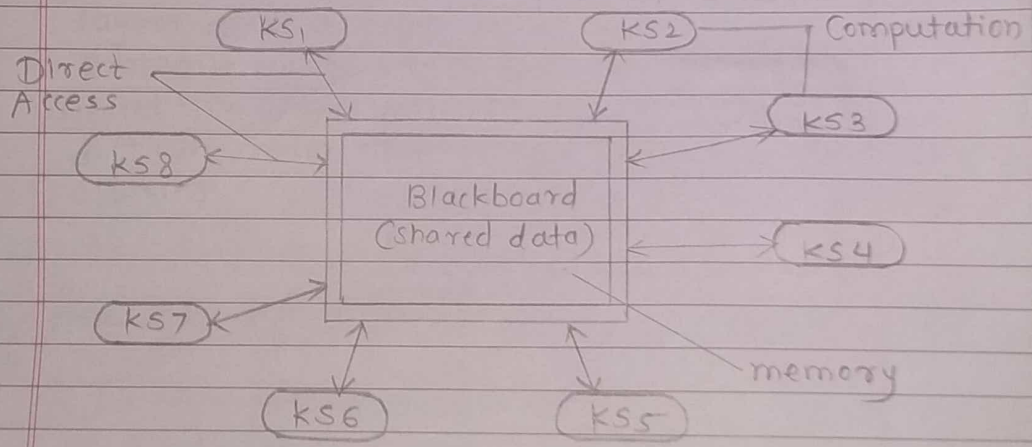


Fig. The blackboard model

Advantages - ① Concurrency ③ Reusability
② Scalability

Disadvantages -
① The structural change of blackboard may have a significant impact on all of its agent.
② This model suffers some problems in synchronization of multiple agents.

5) Explain Layered System a type of Architectural style.

→ ① A layered system is organized hierarchically, each layer providing service to the layer above it & serving as a client to the layer below it.

② In some layered systems inner layers are hidden from all except the adjacent outer layer, except for certain functions carefully selected for export.

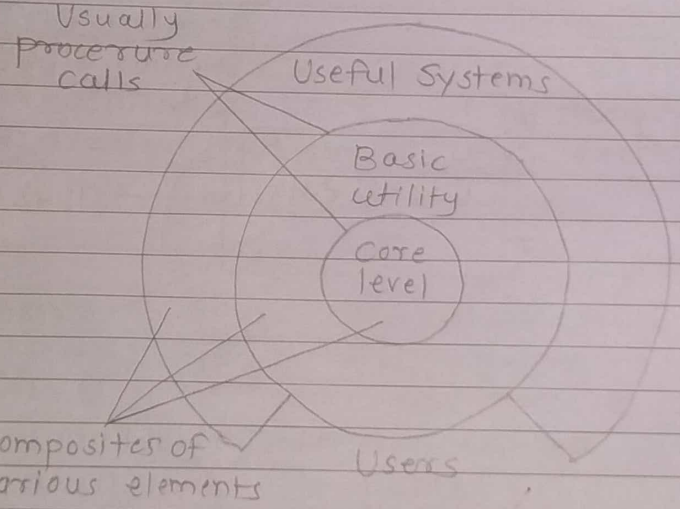


fig. Layered Systems

③ In this application area each layer provides a substrate for communication at some level of abstraction.

④ Lower levels define lower levels of interact, the lowest typically being defined by h/w connections.

Page No.
Date

Advantages -

① Support design based on increasing levels of abstraction.
② Support enhancement
③ Support reuse

Disadvantages -

① Not all systems are easily structured in a layered fashion.
② It is difficult to define system independent layers since a layer must support the specific protocols as its upper & lower boundaries.
③ Find the right level of abstraction may be difficult, often this is the problem with evolving systems.

6) How are interpreters useful in programming ?

→

① In interpreter organization a virtual machine is produced in s/w.
② An interpreter includes the pseudo-program being interpreter & the interpretation engine itself.
③ The interpretation engine includes both the definition of the interpreter & the current state of its execution.
④ Interpreters are commonly used to build virtual machines that close the gap between the computing engine expected by the semantics of the program & the computing engine available in h/w.

⑤ Interpreter examples - Programming Language Compilers like Java & Smalltalk.

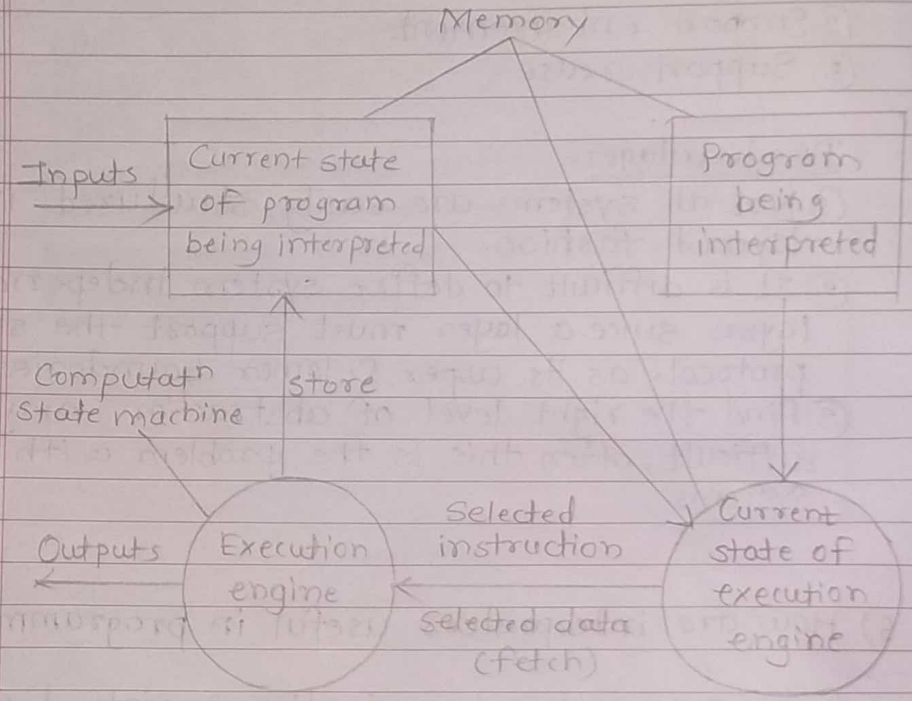


fig. Interpreter

Advantages -

① Simulation of non-implemented h/w; keeps cost of h/w affordable.

② Facilitates portability of appl^n or languages across a variety of platforms.

Disadvantages -

① Extra level of indirection slows down execut^n.

② Java has an option to compile code : JIT (Just In Time) compiler.

7) Can we combine different architectural styles? If yes, how & if no, why?

→ Yes.
There are different ways in which architectural styles can be combined.

① One way is through hierarchy. A component of a system organized in one architectural style may have an internal structure that is developed a completely different style.
For example, a pipe connector may be implemented internally as a FIFO queue accessed by insert & remove operations.

② A second way for styles to be combined is to permit a single component to use a mixture of architectural connectors.
Example is an "active database". This is a repository which activates external components through implicit invocation. In this organizat^{n} external components register interest in portions of the database.

③ A third way for styles to be combined is to completely elaborate one level of architectural description in a completely different architectural style.

8) What are pros & cons of Data Abstraction

→