USN										

Internal Assessment Test II – April 2018

Sub:	Software Architectures							**Code:**	10IS81
Date:	16-04-18	Duration:	90 mins	Max Marks:	50	**Sem:**	VIII	**Branch:**	**ISE**

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

		Marks	OBE	
			CO	RBT
1.	Explain **Layers Architecture pattern**, with sketches and CRC cards.	[10]	CO3	L4
2.	List the components **of Pipes & Filters Architecture Pattern** & depict the dynamics behavior of it.	[10]	CO3	L1,L3
3.	List the steps to implement **Blackboard pattern** and Illustrate its behavior based on Speech recognition.	[10]	CO3	L1,L3
4.	Define **Broker Architecture pattern**. Explain with a diagram the objects involved in a Broker system.	[10]	CO3	L1,L4

USN										

Internal Assessment Test 1 – March 2018

Sub:	Software Architectures							**Code:**	10IS81
Date:	12-03-18	Duration:	90 mins	Max Marks:	50	**Sem:**	VIII	**Branch:**	**CSE, ISE**

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

		Marks	OBE	
			CO	RBT
1.	Explain **Layers Architecture pattern**, with sketches and CRC cards.	[10]	CO3	L4
2.	List the components **of Pipes & Filters Architecture Pattern** & depict the dynamics behavior of it.	[10]	CO3	L1,L3
3.	List the steps to implement **Blackboard pattern** and Illustrate its behavior based on Speech recognition.	[10]	CO3	L1,L3
4.	Define **Broker Architecture pattern**. Explain with a diagram the objects involved in a Broker system.	[10]	CO3	L1,L4

5.	Explain the dynamics of **MVC pattern**	[10]	CO3	L4
6.	Give the CRC cards for top level, intermediate level and bottom level **PAC pattern** agents and explain in brief.	[10]	CO3	L1,L4
7.	Enumerate the Implementation of a **Microkernel Pattern**.	[10]	CO4	L2

5.	Explain the dynamics of **MVC pattern**	[10]	CO3	L4
6.	Give the CRC cards for top level, intermediate level and bottom level **PAC pattern** agents and explain in brief.	[10]	CO3	L1,L4
7.	Enumerate the Implementation of a **Microkernel Pattern**.	[10]	CO4	L2

Scheme

Question #	Description	Marks Distribution		Max Marks
1	Layers Architecture pattern Diagram. Explanation & CRC cards	5M 5M	10M	10M
2	Pipes & Filters Architecture Pattern – components dynamics explanation	3M 5M 2M	10M	10M
3	Blackboard pattern diagram Explanation	5M 5M	10M	10M
4	Broker Architecture pattern Definition Four diagrams & Explanation	2M 8M	10M	10M
5	MVC pattern explanation Dynamics	2M 8M	10M	10M
6	PAC pattern – 3 Patterns Representation	3 M each 1M	10M	10M
7	Microkernel Pattern	10M	10M	10M

Solution

1. LAYERS 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.

   Example: Networking protocols are best example of layered architectures. Such a protocol consists of a set of rules and conventions that describes how computer programmer communicates across machine boundaries. The format, contacts and meaning of all messages are defined. The protocol specifies agreements at a variety of abstraction levels, ranging from the details of bit transmission to high level abstraction logic. Therefore the designers use secured sub protocols and arrange them in layers. Each layer deals with a specific aspect of communication and users the services of the next lower layer. (see diagram & explain more)

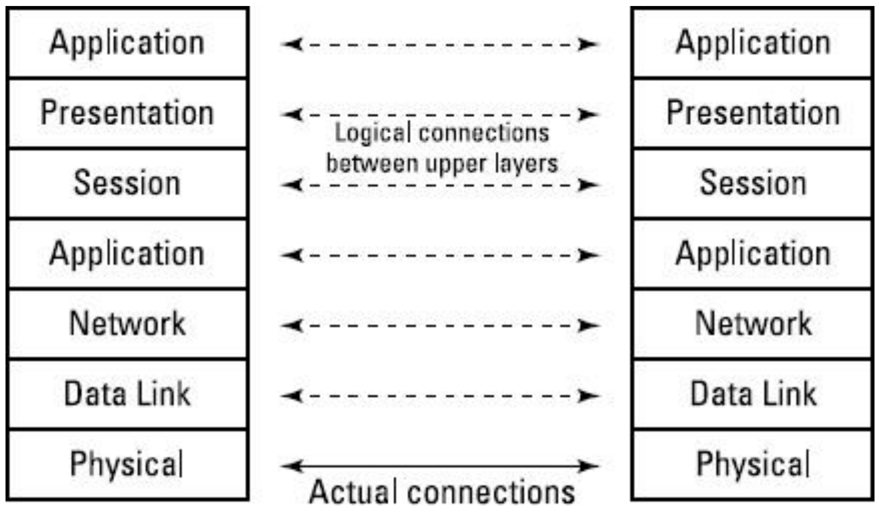
   Context: a large system that requires decomposition.

   Problem: THE SYSTEM WE ARE BUILDING IS DIVIDED BY MIX OF LOW AND HIGH LEVEL ISSUES, WHERE HIGH-LEVEL OPERATIONS RELY ON THE LOWER-LEVEL ONES. FOR EX, HIGH-LEVEL WILL BE INTERACTIVE TO USER AND LOW-LEVEL WILL BE CONCERNED WITH HARDWARE IMPLEMENTATION.

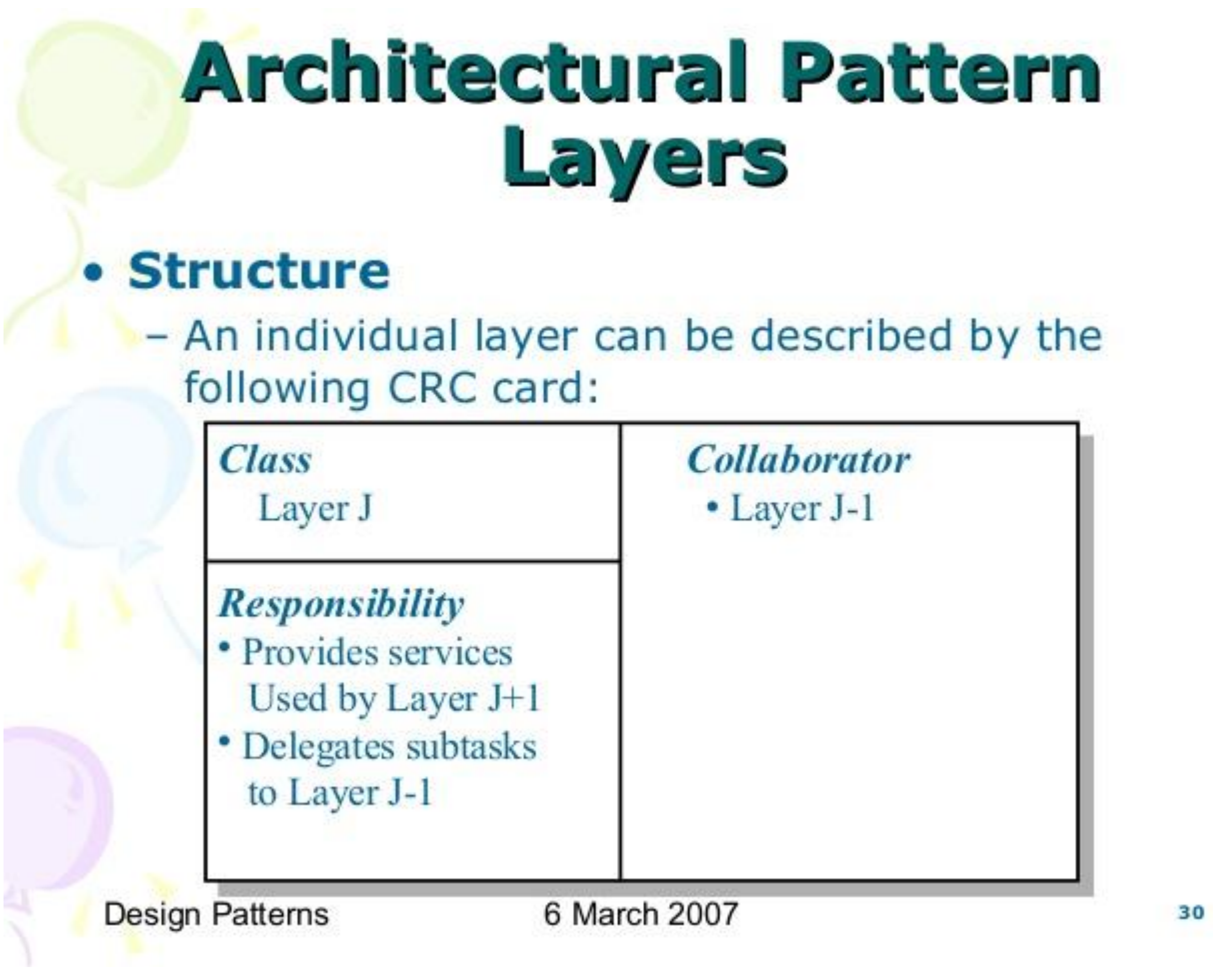
In such a case, we need to balance the following forces: Late source code changes should not ripple through the systems. They should be confined to one component and not affect others. Interfaces should be stable, and may even be impressed by a standards body. Parts of the system should be exchangeable (i.e, a particular layer can be changed). It may be necessary to build other systems at a later date with the same low-level issues as the system you are currently designing. Similar responsibilities should be grouped to help understandability and maintainability. There is no ‘standard’ component granularity. Complex components need further decomposition. Crossing component boundaries may impede performance, for example when a substantial amount of data must be transferred over several boundaries. The system will be built by a team of programmers, and works has to be subdivided along clear boundaries.

Solution: Structure your system into an appropriate number of layers and place them on top of each other. Lowest layer is called 1 (base of our system), the highest is called layer N. i.e, Layer 1, ……. Layer J-1, Layer J, ….. Layer N. Most of the services that layer J Provides are composed of services provided by layer J-1. In other words, the services of each layer implement a strategy for combining the services of the layer below in a meaningful way. In addition, layer J’s services may depend on other services in layer J.

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



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.



Architectural Pattern Layers

- **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	

Design Patterns 6 March 2007 30

In more detail, it might look something like this:

Dynamics: Scenario I is probably the best-known one. A client Issues a request to Layer N. Since Layer N cannot carry out the request on its own. It calls the next Layer N - 1 for supporting subtasks. Layer N - I provides these. In the process sending further requests to Layer N-2 and so on until Layer I is reached. Here, the lowest-level services are finally performed. If necessary, replies to the different requests are passed back up from Layer 1 to Layer 2, from Layer 2 to Layer 3, and so on until the final reply arrives at Layer N. Scenario II illustrates bottom-up communication-a chain of actions starts at Layer 1, for example when a device driver detects input. The driver translates the input into an internal format and reports it to Layer 2 which starts interpreting it, and so on. In this way data moves up through the layers until it arrives at the highest layer. While top-down information and control flow are often described as 'requests'. Bottom-up calls can be termed 'notifications'. Scenario III describes the situation where requests only travel through a subset of the layers. A toplevel request may only go to the next lower level N- 1 if this level can satisfy the request. An example of this is where level N- 1 acts as a cache and a request from level N can be satisfied without being sent all the way down to Layer 1 and from here to a remote server. Scenario IV An event is detected in Layer 1, but stops at Layer 3 instead of travelling all the way up to Layer N. In a communication protocol, for example, a resend request may arrive from an impatient client who requested data some time ago. In the meantime the server has already sent the answer, and the answer and the re-send request cross. In this case, Layer 3 of the server side may notice this and intercept the re-send request without further action. Scenario V involves two stacks of N layers communicating with each other. This scenario is well-known from communication protocols where the stacks are known as 'protocol stacks'. In the following diagram, Layer N of the left stack issues a request. 'The request moves down through the layers until it reaches Layer

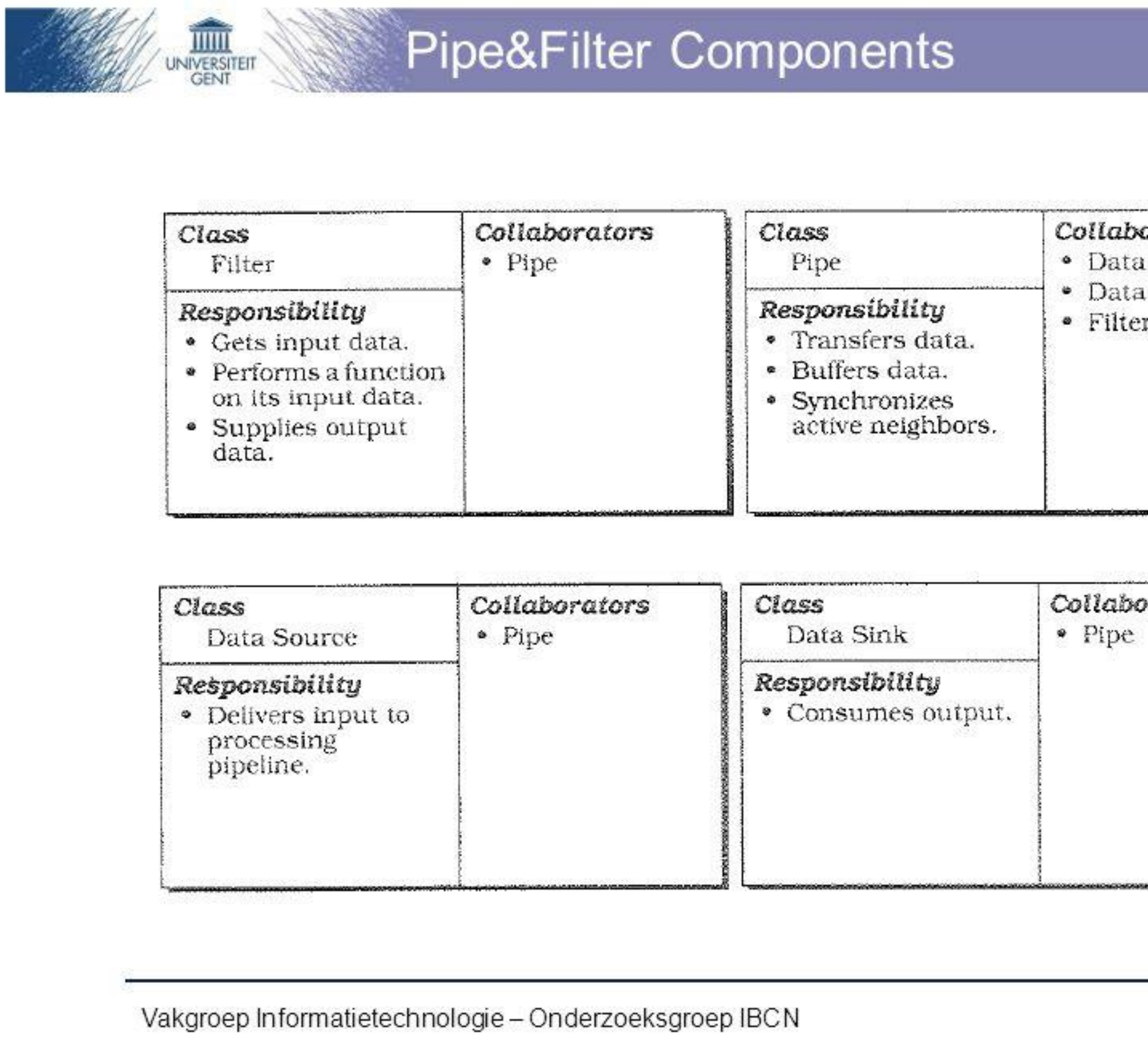
1, is sent to Layer 1 of the right stack, and there moves up through the layers of the right stack. The response to the request follows the reverse path until it arrives at Layer N of the left stack.

2. 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.
   Example: Suppose we have defined a new programming language called Mocha [Modular Object Computation with Hypothetical Algorithms]. Our task is to build a portable compiler for this language. To support existing and future hardware platforms we define an intermediate language AuLait [Another Universal Language for Intermediate Translation] running on a virtual machine Cup (Concurrent Uniform Processor). Conceptually, translation from Mocha to AuLait consists of the phases lexical analysis, syntax analysis, semantic analysis, intermediate-code generation (AuLait), and optionally intermediate-code optimization. Each stage has well-defined input and output data.

Context: Processing data streams.

Problem: Imagine you are building a system that must process or transform a stream of input data. Implementing such a system as a single component may not be feasible for several reasons: the system has to be built by several developers, the global system task decomposes naturally into several processing stages, and the requirements are likely to change. The design of the system-especially the interconnection of the processing steps-has to consider the following forces: Future system enhancements should be possible by exchanging processing steps or by recombination of steps, even by users. Small processing steps are easier to reuse in different contexts than larger contexts. Non-adjacent processing steps do not share information. Different sources of input data exists, such as a network connection or a hardware sensor providing temperature readings, for example. It should be possible to present or store the final results in various ways. Explicit storage of intermediate results for further processing the steps, for example running them in parallel or quasi-parallel. You may not want to rule out multi-processing the steps

Solution: The pipes and filters architectural pattern divides the task of a system into several sequential processing steps (connected by the dataflow through the system). Each step is implemented by a filter component, which consumes and delivers data incrementally to achieve low battery and parallel processing.
 The input to a system is provided by a data source such as a text file. The output flows into a data sink such as a file, terminal, and so on. The data source, the filters, and the data sink are connected sequentially by pipes. Each pipe implements the data flow between adjacent processing steps. The sequence of filters combined by pipes is called a processing pipeline.
Structure: Filter component: o Filter components are the processing units of the pipeline. o A filter enriches, refines or transforms its input data. It enriches data by computing and adding information, refines data by concentrating or extracting information, and transforms data by delivering the data in some other representation. o It is responsible for the following activities: The subsequent pipeline element pulls output data from the filter. (passive filter)
The previous pipeline element pushes new input data to the filter. (passive filter) Most commonly, the filter is active in a loop, pulling its input from and pushing its output down the pipeline. (active filter)



Pipe&Filter Components

Class Filter	*Collaborators*
Responsibility • Gets input data. • Performs a function on its input data. • Supplies output data.	• Pipe

Class Pipe	*Collabo*
Responsibility • Transfers data. • Buffers data. • Synchronizes active neighbors.	• Data • Data • Filter

Class Data Source	*Collaborators*
Responsibility • Delivers input to processing pipeline.	• Pipe

Class Data Sink	*Collabo*
Responsibility • Consumes output.	• Pipe

Vakgroep Informatietechnologie – Onderzoeksgroep IBCN

3. The blackboard architectural pattern is useful for problems for which no deterministic solution strategies are known. In blackboard several specialized subsystems assemble their knowledge to build a possibly partial or approximate solution.
4.
5. Example: Consider a software system for speech recognition. The input to the system is speech recorded as a waveform. The system not only accepts single words, but also whole sentences that are restricted to the syntax and vocabulary needed for a specific application, such as a database query. The desired output is a machine representation of the corresponding English phrases.
6.
7.
8. Context: An immediate domain in which no closed approach to a solution is known or feasible
9.
10. Problem: A blackboard pattern tackle problems that do not have a feasible deterministic solution for the transformation of raw data into high level data structures, such as diagrams, tables, or English phrases. Examples of domains in which such problems occur are:- vision, image recognition, and speech recognition. They are characterized by problems that when decomposed into sub problems,

spans several fields of expertise. The Solution to the partial problems requires different representations and paradigm.

11. The following forces influence solutions to problems of this kind: A complete search of the solution space is not feasible in a reasonable time. Since the domain is immature, we may need to experiment with different algorithms for the same subtask. Hence, individual modules should be easily exchangeable. There are different algorithms that solve partial problems. Input as well as intermediate and final results, have different representation, and the algorithms are implemented according to different paradigms. An algorithm usually works on the results of other algorithms. Uncertain data and approximate solutions are involved. Employing dis fact algorithms induces potential parallelism.
12.
13. Solution: The idea behind the blackboard architecture is a collection of independent programs that work co operatively on a common data structure. Each program is meant for solving a particular part of overall task. These programs are independent of each other they do not call each other, nor there is a predefined sequence for their activation. Instead, the direction taken by the system is mainly determined by the current state of progress. A central control component evaluates the current state of processing and coordinates these programs. These data directed control regime is referred to as opportunistic problem solving. The set of all possible solutions is called solution space and is organized into levels of abstraction. The name 'blackboard' was chosen because it is reminiscent of the situation in which human experts sit in front of a real blackboard and work together to solve a problem. Each expert separately evaluates the current state of the solution, and may go up to the blackboard at any time and add, change or delete information. Humans usually decide themselves who has the next access to the blackboard.
14.
15. Structure: Divide your system into a component called a blackboard, a collection of knowledge sources, and a control component. Blackboard: o Blackboard is the central data store. o Elements of the solution space and control data are stored here. o Set of all data elements that can appear on the blackboard are referred to as vocabulary. o Blackboard provides an interface that enables all knowledge sources to read form and write to it. o We use the terms hypothesis or blackboard entry for solutions that are constructed during the problem solving process and put on the blackboard. o A hypothesis has several attributes, ex: its abstraction level.
16. dge applications strategy. The basis for this strategy is the data on the blackboard.

o The following figure illustrates the relationship between the three components of the blackboard architecture.

Knowledge source calls inspect()to check the current solutions on the blackboard Update() is used to make changes to the data on the blackboard Control component runs a loop that monitors changes on the blackboard and decides what actions to take next. nextSource() is responsible for this decision.

17.
18. Dynamics: The following scenario illustrates the behavior of the Blackboard architecture. It is based on our speech recognition example: The main loop of the Control component is started. Control calls the nextsource ( ) procedure to select the next knowledge source. nextsource ( ) first determines which knowledge sources are potential contributors by observing the blackboard. nextsource() invokes the condition-part of each candidate knowledge source. The Control component chooses a knowledge source to invoke, and a hypothesis or a set of hypotheses to be worked on.

Implementation: Define the problem pecify the domain of the problem Scrutinize the input to the system Define the o/p of the system Detail how the user interacts with the system. Define the solution space for the problem Specify exactly what constitutes a top level solution List the different abstraction levels of solutions Organize solutions into one or more abstraction hierarchy. Find subdivisions of complete solutions that can be worked on independently. Divide the solution process into steps. Define how solutions are transformed into higher level solutions. Describe how to predict hypothesis at the same abstraction levels. Detail how to verify predicted hypothesis by finding support for them in other levels. Specify the kind of knowledge that can be uses to exclude parts of the solution space. Divide the knowledge into

specialized knowledge These subtasks often correspond to areas of specialization. Define the vocabulary of the blackboard Elaborate your first definition of the solution space and the abstraction levels of your solution. Find a representation for solutions that allows all knowledge sources to read from and contribute to the blackboard. The vocabulary of the blackboard cannot be defined of knowledge sources and the control component. Specify the control of the system. Control component implements an opportunistic problem-solving strategy that determines which knowledge sources are allowed to make changes to the blackboard. The aim of this strategy is to construct a hypothesis that is acceptable as a result. The following mechanisms optimizes the evaluation of knowledge sources, and so increase the effectiveness and performance of control strategy. Classifying changes to the blackboard into two types. One type specify all blackboard change that may imply new set of applicable knowledge sources, and the other specifies all blackboard changes that do not. Associating categories of blackboard changes with sets of possibly applicable knowledge sources. Focusing of control. The focus contains either partial results on the blackboard or knowledge sources that should be preferred over others. Creating on queue in which knowledge sources classified as applicable wait for their execution. Implement the knowledge sources
Split the knowledge sources into condition parts and action-parts according to the needs of the control component. We can implement different knowledge sources in the same system using different technologies

Variants: Production systems: used in oops language. Here the subroutines are represented as condition-action rules, and data is globally available in working. Repository: it is a generalization of blackboard pattern the central data structure of this variant is called a repository.

19.