USN										

IAT 2 – April 2018 Scheme of evaluation

Sub:	Software Engineering				Sub Code:	15CS42	Branch:	CSE
Date:	16/04/2018	Duration:	90 mins	Max Marks: 50	Sem/Sec:	4 (A,B,C)	OBE	

Answer **FOUR** FULL questions selecting AT LEAST ONE question **FROM EACH PART** — MARKS — CO — RBT

PART A

1 (a) With an example, explain Requirements based testing. [6] CO5 L2
(Requirement- 1M, tests- 5M)

(i) Requirements - based testing

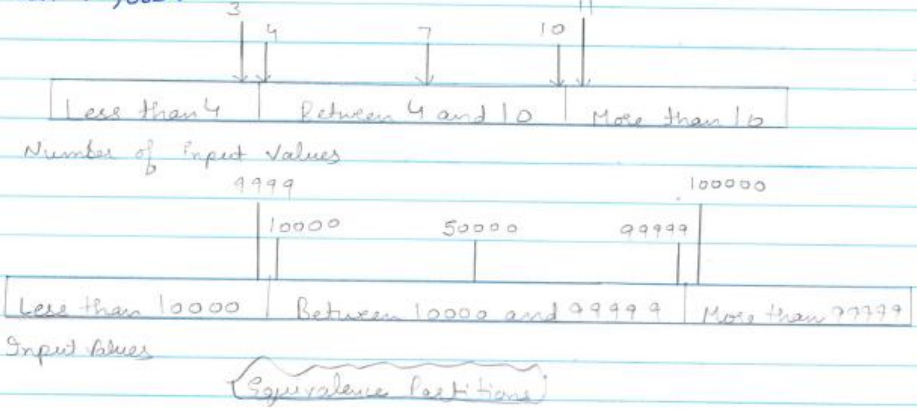
It involves examining each requirement and developing a test for it.
For example: MHC-PMS for requirements for drug allergies:

- If a patient is known to be allergic to any particular medication then presciption of that medication shall result in a warning message being issued to the system user.
- If a prescriber chooses to ignore an allergy warning, they shall provide a reason why this has been ignored.

Related requirements tests will be:

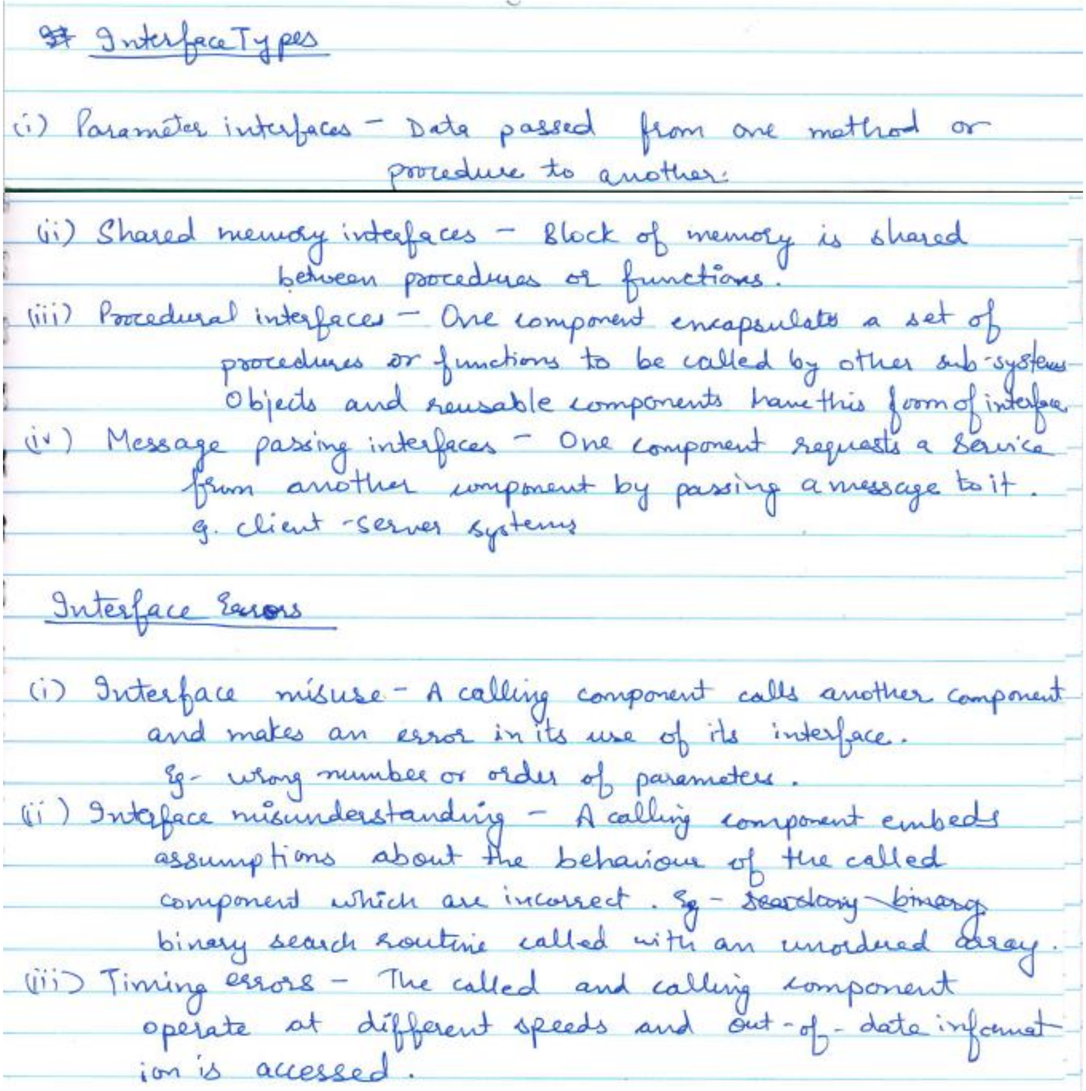
1. Set up a patient record with no known allergies. Prescribe medication for allergies that are known to exist. Check that a warning message is not issued by the system,
2. Set up a patient record with a known allergy. Prescribe the medication to that the patient is allergic to, and check that the warning is issued by the system.
3. Set up a patient record in which allergies to two or more drugs are recorded. Prescribe both of these drugs separately and check that the correct warning for each drug is issued.
4. Prescribe two drugs that the patient is allergic to. Check that two warnings are correctly issued.
5. Prescribe a drug that issues a warning and overrule that warning. Check that the system requires the user to provide information explaining why the warning was overruled.

(b) A program specification states that the program accepts 4 to 10 inputs, which are 5-digit integers greater than 10,000. Show equivalence partitions for testing. [6] CO5 L3
(No. of inputs partitions3M, Input values partitions: 3M)



OR

2 (a) Explain various interface types and Interface Errors. [7] CO5 L1
(Interface types- 4M, Interface errors- 3M)



Interface Types

(i) Parameter interfaces – Data passed from one method or procedure to another.

(ii) Shared memory interfaces – Block of memory is shared between procedures or functions.

(iii) Procedural interfaces – One component encapsulates a set of procedures or functions to be called by other sub-systems. Objects and reusable components have this form of interface

(iv) Message passing interfaces – One component requests a service from another component by passing a message to it. g. client -server systems

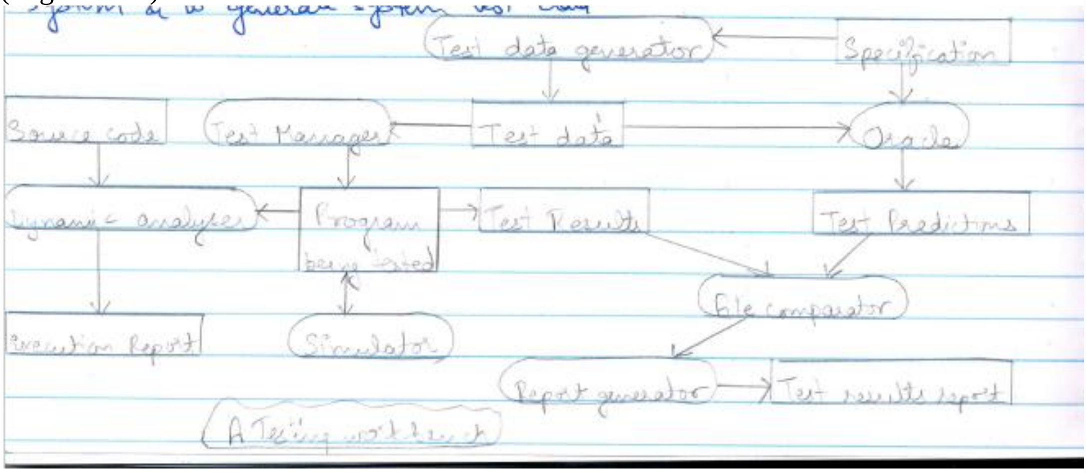
Interface Errors

(i) Interface misuse – A calling component calls another component and makes an error in its use of its interface. Eg- wrong number or order of parameters.

(ii) Interface misunderstanding – A calling component embeds assumptions about the behaviour of the called component which are incorrect. Eg – ~~searching~~ ~~binary~~ binary search routine called with an unordered array.

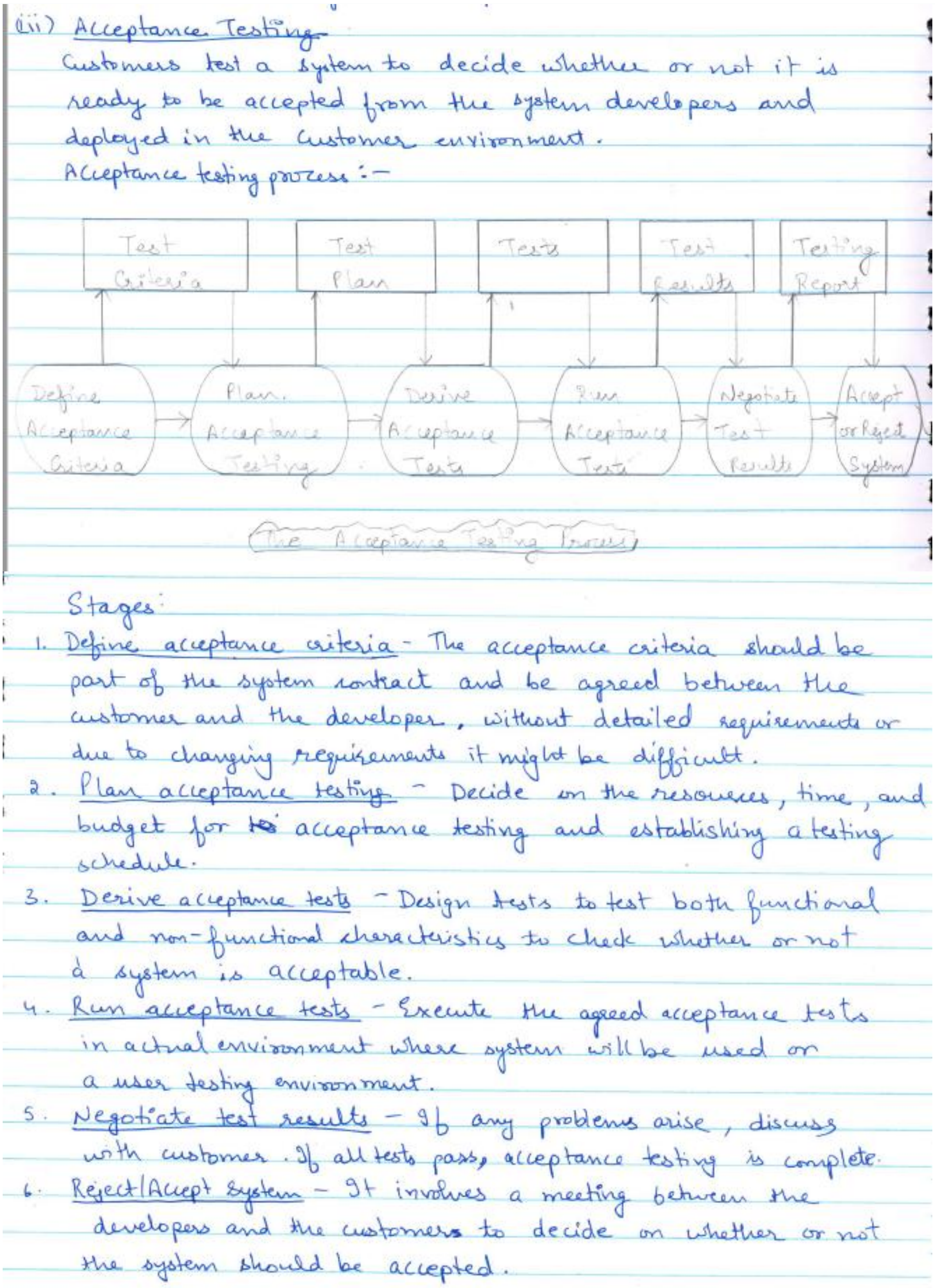
(iii) Timing errors – The called and calling component operate at different speeds and out-of-date information is accessed.

(b) With a neat diagram, explain Test Automation. [5] CO4 L2
(Digram-5M)



PART B

3 (a) With neat diagram, explain six stages of acceptance testing process. [6] CO5 L2
(Diagram-4M, Explanation-2M)



(ii) Acceptance Testing

Customers test a system to decide whether or not it is ready to be accepted from the system developers and deployed in the customer environment.

Acceptance testing process :-

Test Criteria
Test Plan
Tests
Test Results
Testing Report
Define Acceptance Criteria
Plan Acceptance Testing
Derive Acceptance Tests
Run Acceptance Tests
Negotiate Test Results
Accept or Reject System

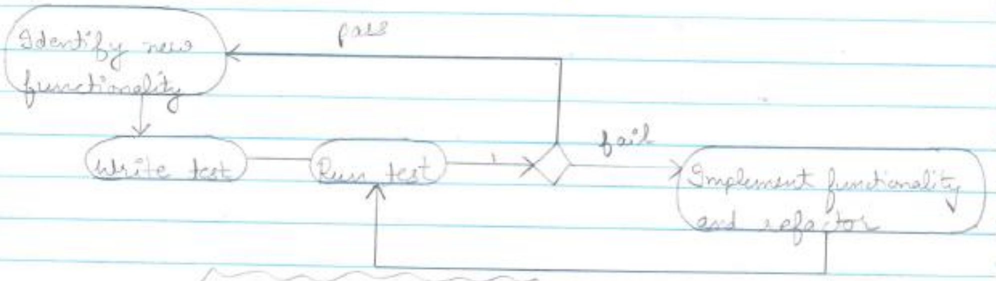
(The Acceptance Testing Process)

Stages:

1. Define acceptance criteria - The acceptance criteria should be part of the system contract and be agreed between the customer and the developer, without detailed requirements or due to changing requirements it might be difficult.
2. Plan acceptance testing - Decide on the resources, time, and budget for ~~to~~ acceptance testing and establishing a testing schedule.
3. Derive acceptance tests - Design tests to test both functional and non-functional characteristics to check whether or not a system is acceptable.
4. Run acceptance tests - Execute the agreed acceptance tests in actual environment where system will be used or a user testing environment.
5. Negotiate test results - If any problems arise, discuss with customer. If all tests pass, acceptance testing is complete.
6. Reject/Accept System - It involves a meeting between the developers and the customers to decide on whether or not the system should be accepted.

(b)	What is Test-driven development? State the benefits of Test-driven development. (Test-driven development exaplanation-2M, Benefits- 4M)	[6]	CO4	L2

- Test-driven development (TDD) is an approach to program development in which you inter-leave testing and code development.
- Tests are written before code and 'passing' the tests is the critical driver of development.
- You develop code incrementally, along with a test for that increment. You don't move on to the next increment until the code that you have developed passes its test.


Identify new functionality
pass
fail
Write test
Run test
Implement functionality and refactor

Test-driven development

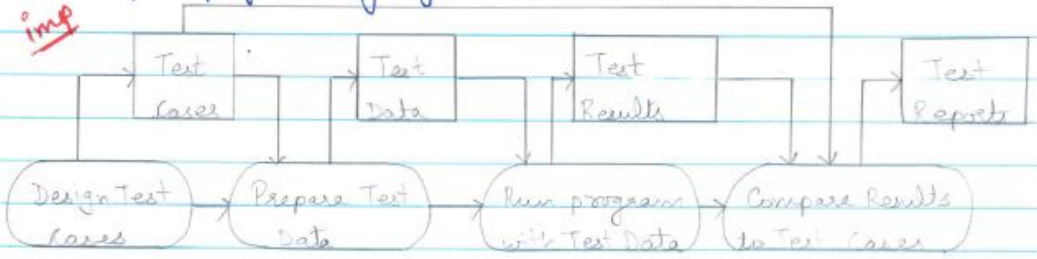
Benefits of TDD:

- Code coverage – Every code segment that ~~have~~ you write has at least one associated test so all code written has at least one test.
- Regression testing – A regression test suite is developed incrementally as a program is developed which can be run to test for any new introduced bugs because of changes.
- Simplified debugging – When a test fails, it should be obvious where the problem lies. The newly written code needs to be checked and modified.
- System documentation – The tests themselves are a form of documentation that describe what the code should be doing.

OR

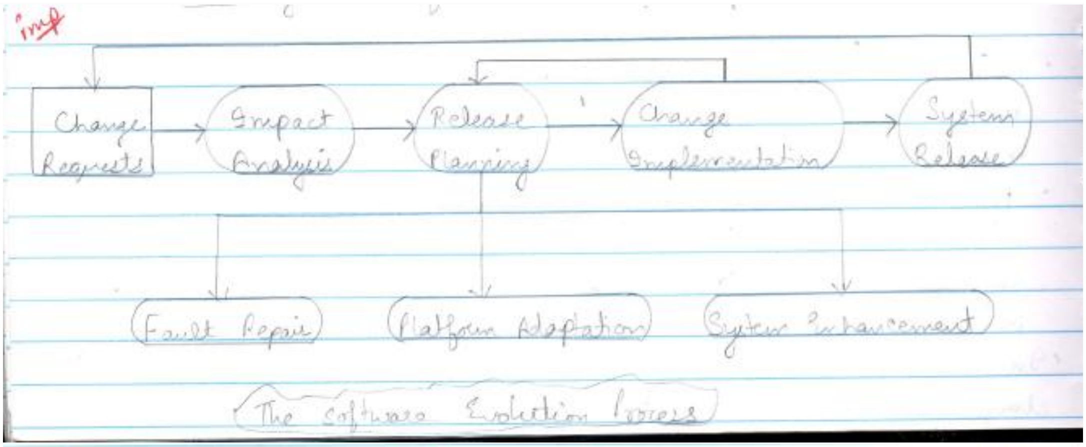
4 (a) With a block diagram, explain a model of software testing process. (Diagram-5M, Explanation-1M) [6] CO5 L2

imp



A model of Software Testing Process

(b) With a neat diagram, explain the software evolution process. (Diagram-5M, Explanation-1M) [6] CO5 L2



The Software Evolution Process

The cost and impact of proposed changes are assessed. If the proposed changes are accepted, a new release of the system is planned. During release planning, all proposed changes (fault repair, adaptation, and new functionality) are considered. A decision is then made on which changes to implement in the next version of the system. The changes are implemented and validated and a new version of the system is released. The process then iterates with a new set of changes proposed for the next release.

PART C

5 (a) Define "Program Evolution Dynamics". Discuss Lehman laws for program evolution dynamics. [10] CO5 L2

(Definition-2M, Laws- 8x1M)

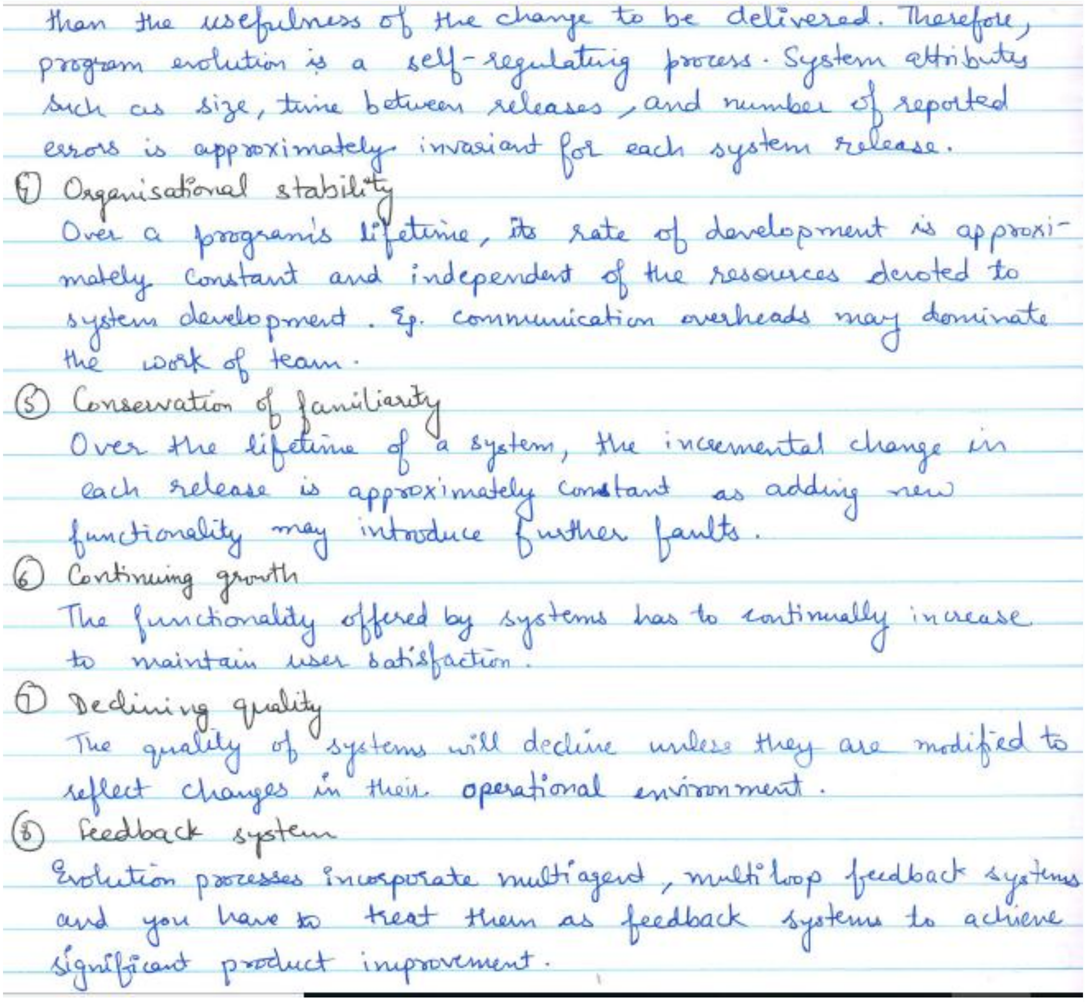
Program Evolution dynamics is the study of the processes of system change.

Lehman's Laws

① Continuing change
A program that is used in a real-world environment must necessarily change, or else become progressively less useful in that environment.

② Increasing complexity
As an evolving program changes, its structure tends to become more complex. Extra resources must be devoted to ~~pre~~ preserve and simplify the structure.

③ Large program evolution
Once system exceeds minimal size, it becomes more complex, hard to understand leading to more likely for programmers to make errors. A large change may introduce more new faults



than the usefulness of the change to be delivered. Therefore, program evolution is a self-regulating process. System attributes such as size, time between releases, and number of reported errors is approximately invariant for each system release.

④ Organisational stability

Over a program's lifetime, its rate of development is approximately constant and independent of the resources devoted to system development. Eg. communication overheads may dominate the work of team.

⑤ Conservation of familiarity

Over the lifetime of a system, the incremental change in each release is approximately constant as adding new functionality may introduce further faults.

⑥ Continuing growth

The functionality offered by systems has to continually increase to maintain user satisfaction.

⑦ Declining quality

The quality of systems will decline unless they are modified to reflect changes in their operational environment.

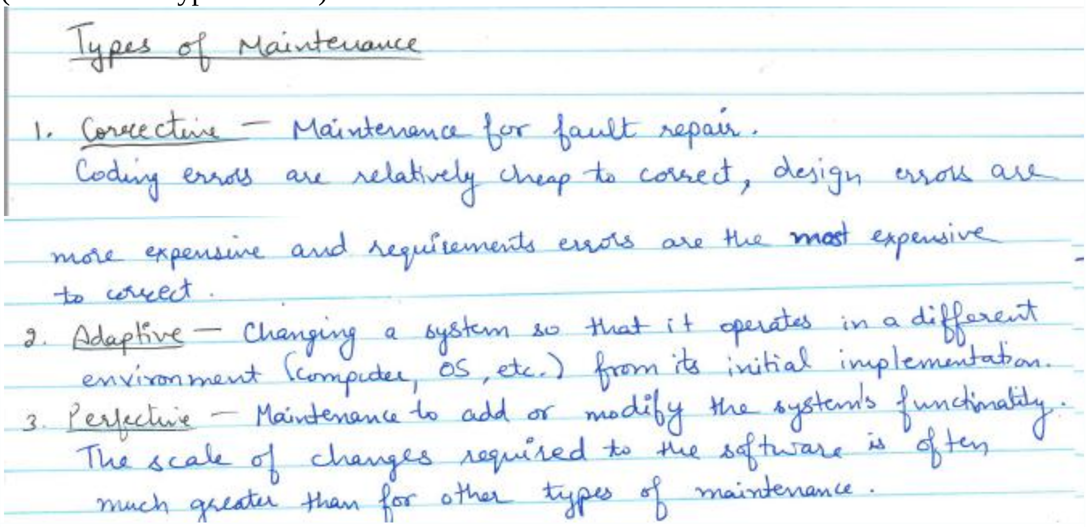
⑧ Feedback system

Evolution processes incorporate multiagent, multiloop feedback systems and you have to treat them as feedback systems to achieve significant product improvement.

OR

6 (a) Define three types of software maintenance. [3] CO5 L1

(Maintenance types- 3x1M)



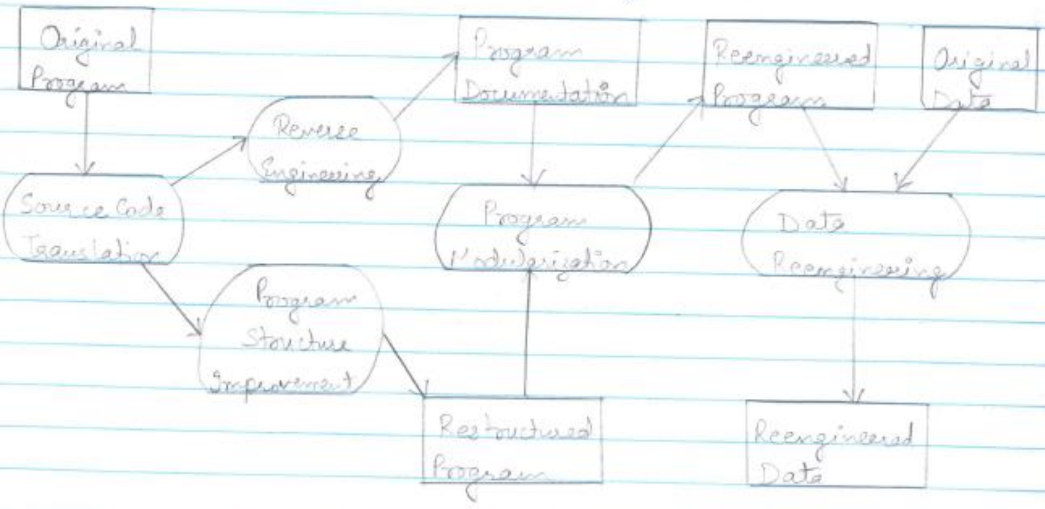
Types of Maintenance

1. Corrective — Maintenance for fault repair.
Coding errors are relatively cheap to correct, design errors are more expensive and requirements errors are the **most** expensive to correct.
2. Adaptive — Changing a system so that it operates in a different environment (computer, OS, etc.) from its initial implementation.
3. Perfective — Maintenance to add or modify the system's functionality.
The scale of changes required to the software is often much greater than for other types of maintenance.

(b) With a neat diagram, explain activities of re-engineering process. [7] CO4 L2

(Diagram- 5M, explanation-2M)

not present . cost may be high .



The Re-engineering process

- System reengineering is re-structuring or re-writing part or all of a legacy system without changing its functionality.

Activities:-

(i) Source code translation – Tool to convert program from an old programming language to a more modern version of same language or to a different language.

(ii) Reverse engineering – usually it is automated process. The program is analyzed and information extracted from it. This helps to document its organization & functionality.

iii) Program structure improvement – It can be partially automated. The control structure of the program is analysed and modified to make it easier to read and understand.

(iv) Program modularization – Related parts of the program are grouped together and where appropriate, redundancy is removed. This is manual process.

(v) Data reengineering – The data processed by the program is changed to reflect program changes. This means redefining database schemes and converting existing databases to new structure. This involves finding and correcting mistakes, removing duplicate records etc.

All activities may not be necessary.

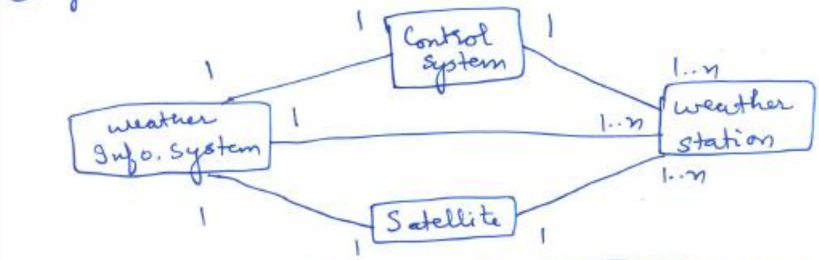
PART D

7 (a)	Explain with diagrams, activities involved in Object-oriented design using UML. (context diagram-2M, Interaction- 2M, Architectural design- 2M, Object class identification-3M, Sequence diagram- 2M, State diagram-3M, Interface specification-2M)	[16]	CO1	L3

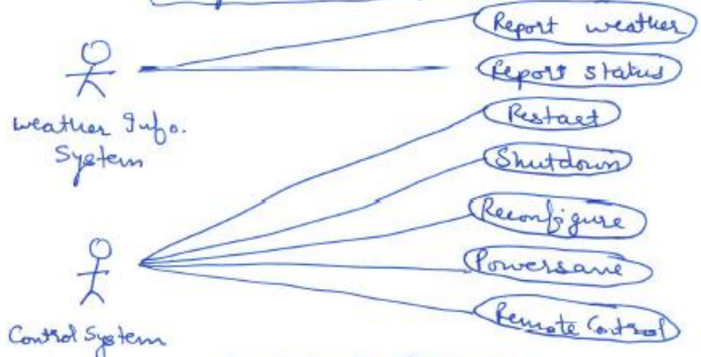
Object - oriented design using UML

Common activities in these processes include :-

① System context and iteractions :

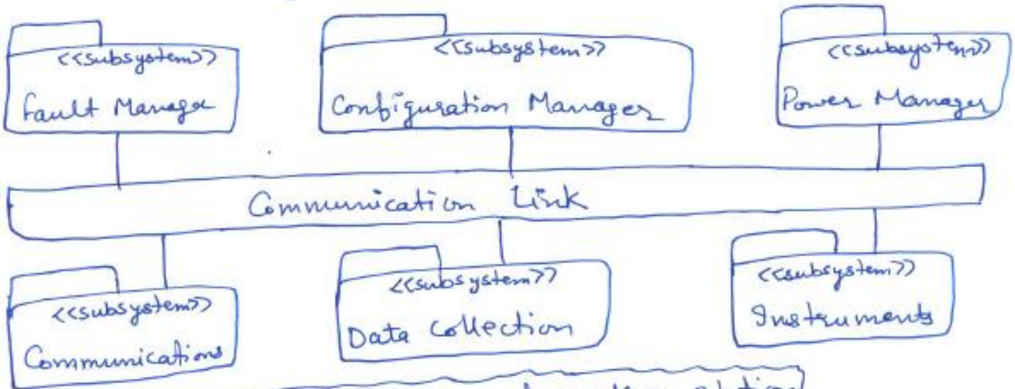


System context for weather station

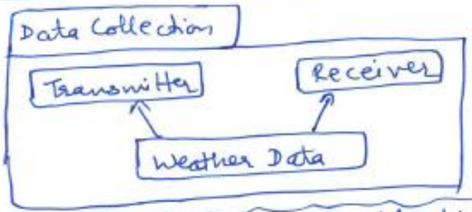


weather station use cases

② Architectural design :

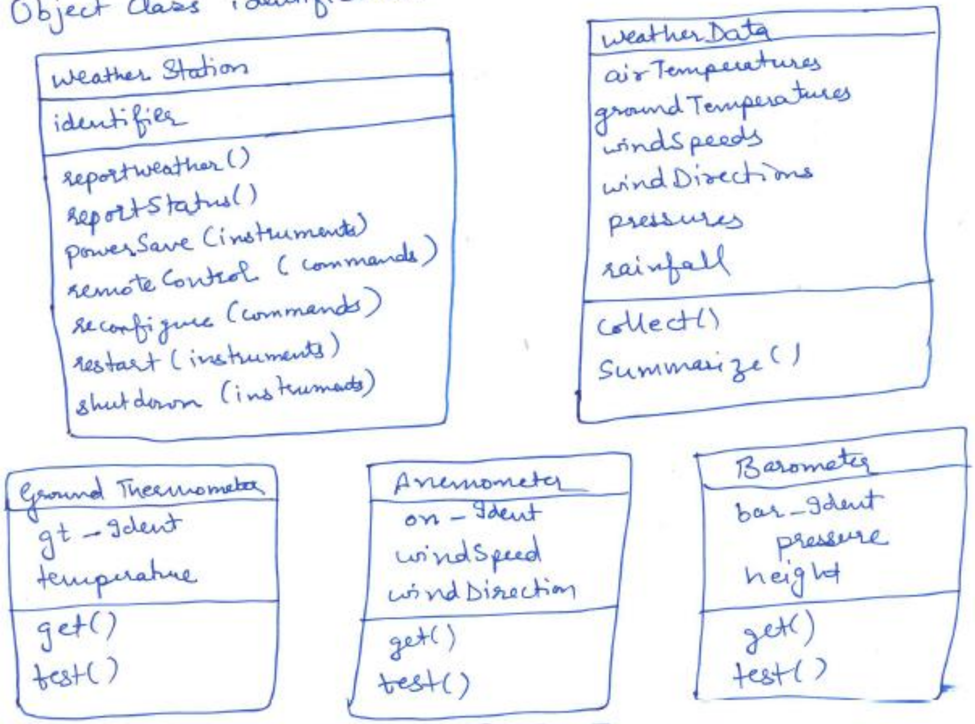


High-level architecture of weather station



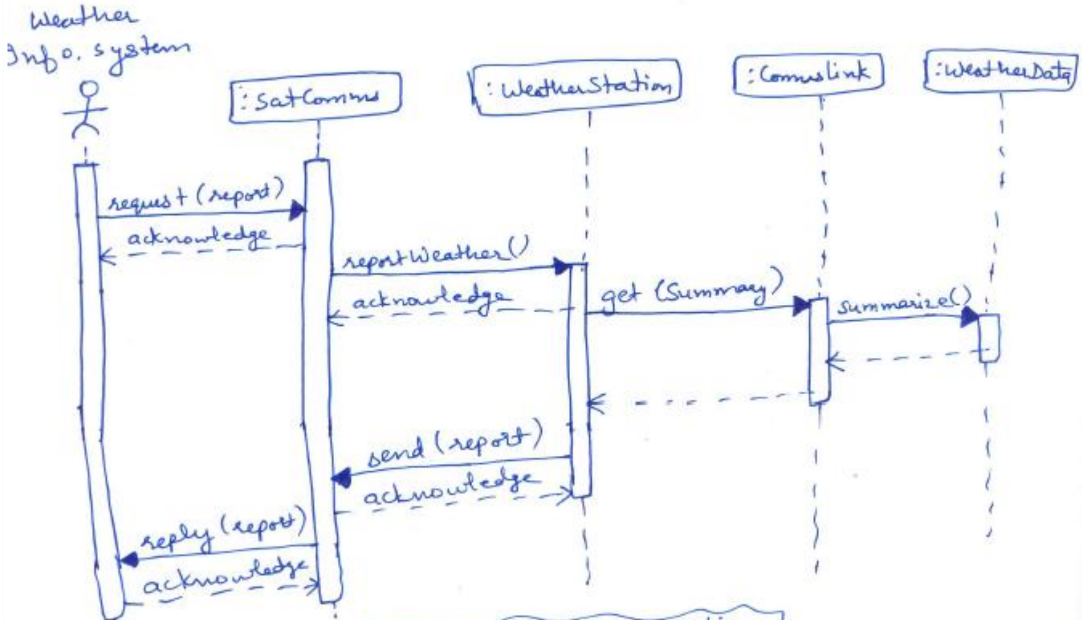
Architecture of Data Collection system

③ Object Class identification:



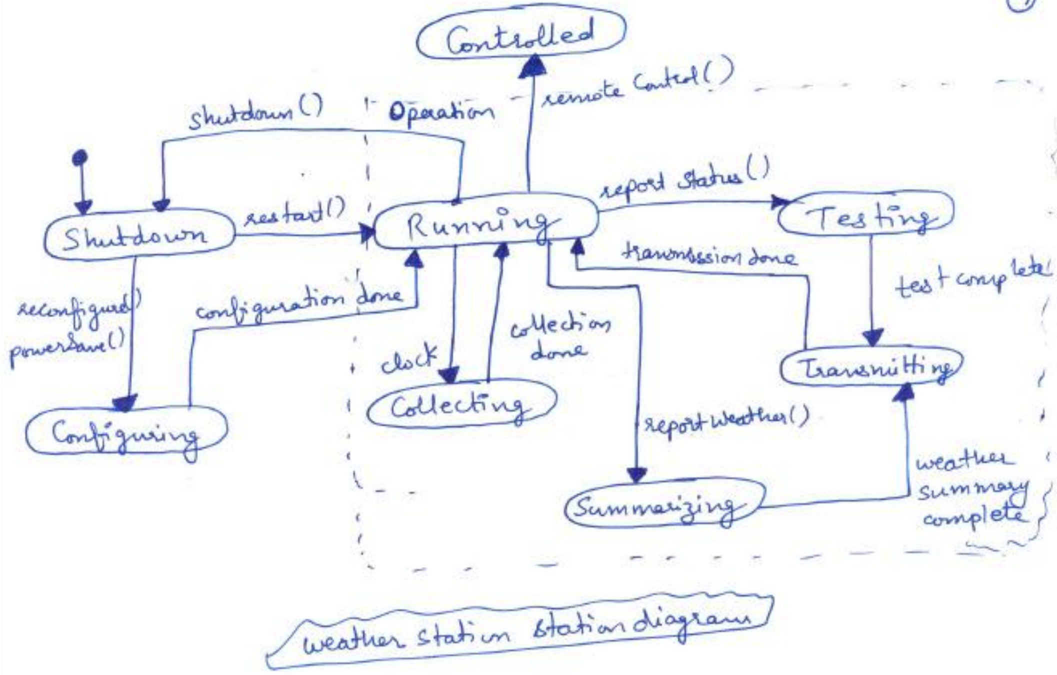
weather Station object classes

④ Design models:

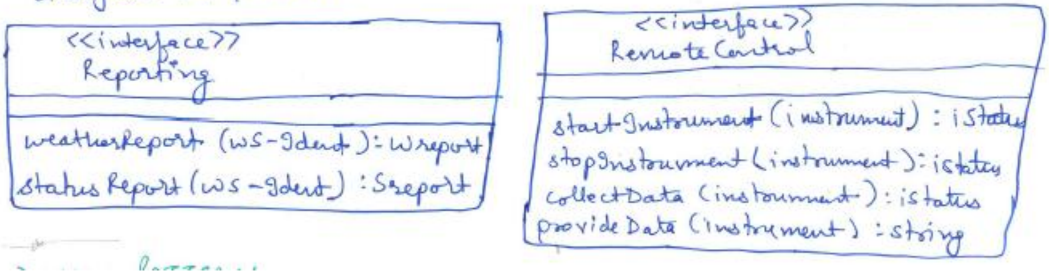


Sequence diagram describing data collection

7



3 Interface specification:
specify interfaces so that objects and subsystems can be designed in parallel.



<<interface>>
Reporting

weatherReport (WS-Ident): Wreport
status Report (WS-Ident) :Sreport

<<interface>>
Remote Control

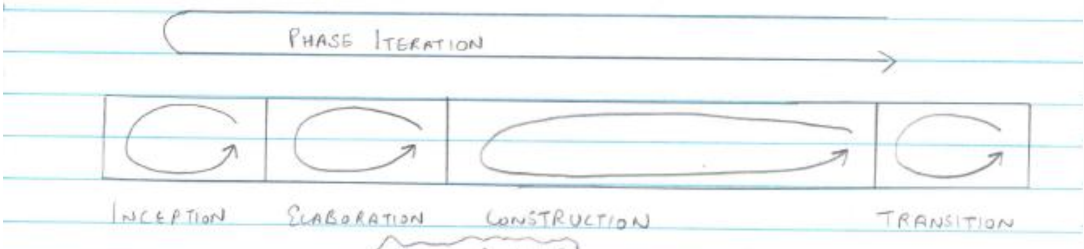
start-Instrument (instrument) : iStatus
stopInstrument (instrument): istatus
collectData (instrument): istatus
provide Data (instrument) : string

OR

8 (a)	With a neat block diagram, explain the phases of Rational Unified Process. (Diagram-4M, explanation-2M)	[6]	CO1	L2

Dynamic Perspective – Phases

Phases are closely related to business rather than technical concerns:



Phases in RUP

(i) Inception : Incept the idea of the business for system and the system will contribade to business significantly. Identify / define external entities & there interactions with system.

(ii) Elaboration : Understand problem domain, establish architectural framework for system, develop project plan, identify key project risks and finally create requirements model (Use cases) for the system.

(iii) Construction : It involves system design, programming and testing. Parts of system are developed in parallel and integrated during this phase. Working software system and associated documentation is ready for delivery to users.

(iv) Transition : Moving the system from the development community to the user community and making it work in a real environment.

Each phase or all the phases may be iterative.

(b) What are design patterns? Explain with example. [7] CO1 L2
(Diagram-4M,expalanation-3M)

Design pattern is a description of the problem and the essence of its solution, so that the solution may be reused in different settings. It allows reusing abstract knowledge about a problem and its solution. It is not a detailed specification.

Essential elements of design patterns :-

- Name – a meaningful pattern identifier.
- Problem description – describes when pattern may be applied.
- Solution description – not a concrete design but a template for a design solution that can be instantiated in different ways.
- Consequences – the results and trade-offs of applying the pattern

For example:-

Pattern name : Observer

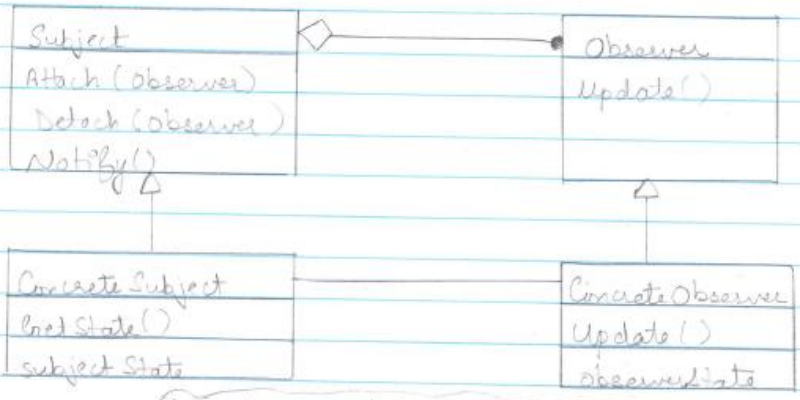
Description : separates the object that must be displayed from the different forms of presentation. If object changes, all displays are automatically notified and updated to reflect the change.

Problem description : This pattern may be used in all situations where more than one display format for state information is required.

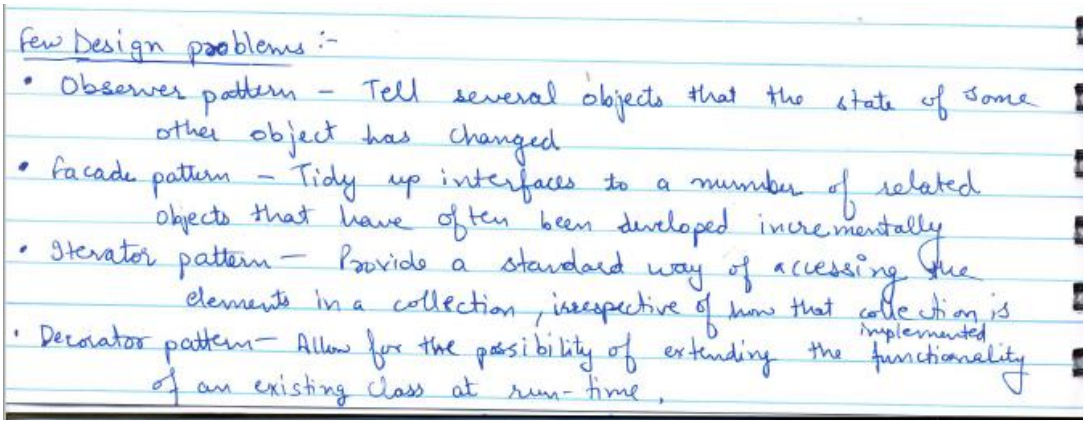
Solution description : Two abstract objects, Subject and Observer are used which include general operations applicable in all situations. Two concrete objects, Concrete Subject and Concrete Object inherit properties of these abstract objects. The state to be displayed is maintained in Concrete Subject which inherits operations from Subject allowing it to add and remove Observers (each observer corresponds to a display) and to issue a notification when the state has changed.

The ConcreteObserver maintains a copy of the state of ConcreteSubject and implements the Update () ~~&~~ interface of observer. The ~~ConcreteObject~~ ConcreteObserver automatically displays the state and reflects changes whenever state is updated. (UML model shown in figure below).

Consequences : The subject only knows the abstract Observer and doesnot know details of concrete objects. Changes to subject may cause a set of linked updates to observers to be generated, some of which may not be necessary.



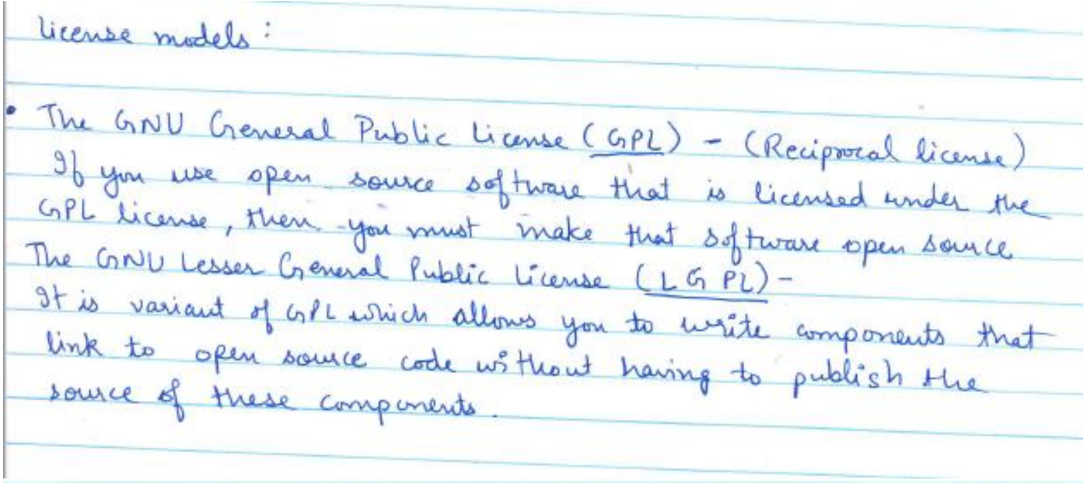
(A UML model of Observe Pattern)



Few Design problems :-

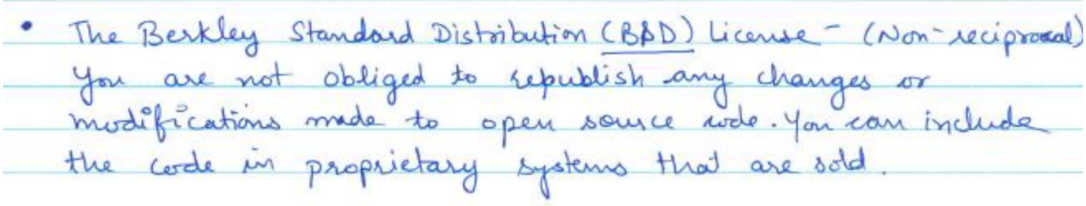
- Observer pattern – Tell several objects that the state of some other object has changed
- Facade pattern – Tidy up interfaces to a number of related objects that have often been developed incrementally
- Iterator pattern – Provide a standard way of accessing the elements in a collection, irrespective of how that collection is implemented
- Decorator pattern – Allow for the possibility of extending the functionality of an existing class at run-time.

(c) State general models of Open Source Licenses. [3]
(License- 3x1M)



license models :

- The GNU General Public License (GPL) – (Reciprocal license)
If you use open source software that is licensed under the GPL license, then you must make that software open source
The GNU Lesser General Public License (LGPL) –
It is variant of GPL which allows you to write components that link to open source code without having to publish the source of these components.



- The Berkley Standard Distribution (BSD) License – (Non-reciprocal)
You are not obliged to republish any changes or modifications made to open source code. You can include the code in proprietary systems that are sold.

CO5 L1