Chapter - 1

Software Process and Requirements

1.1 Software Crisis

1.2 Software Characteristics

1.3 Software quality attributes

1.4 Software process model

1.5 Process iteration

1.6 Process activities

✓ 1.7 Computer aided software engineering

✓ 1.8 Functional and non-functional requirements

1.9 User requirements

1.10 System requirements

1.11 Interface specification

1.12 Software requirements documents

✓ 1.13 Feasibility study

✓ 1.14 Requirement elicitation and analysis

✓ 1.15 Requirement validation and management

[20 - marks]

Questions

1> Why is it so difficult to gain clear understanding of what customer wants? What are the guidelines for requirement elicitation process? [4+4]

2> Explain details about current model of software process. Explain why waterfall model is not an accurate reflection of software development activities. [4+4]

3> What are typical software characteristics? What do you mean by software crisis? [4+4]

4> What are the reasons for software runways? Explain how both waterfall model and prototyping model can be accommodated in spiral model. [2+6]

5> Explain alternative model for waterfall model. [5]

6> Give your views on requirement engineering and requirement specification. [10]

7> What are different processes for requirements gathering? Explain at least three. Prepare a comparative chart of with pros and cons of each. [3+4.5+2.5]

8> What are major components of any feasibility study report? Explain with examples. The candidate matrix with recommendation in feasibility report is considered standard, justify with reason. [7+3]

9> What are techniques used for requirement gathering and analysis? Explain any three. [7]

10> What makes software development process a complex? The simple man-month measurement and additional workers assignment for delayed project does not work in software project, why? [7]

1.1 Software Crisis

Software crisis is defined as the failure of a software being developed or being used that leads to considerable loss of time, efforts and economy. Generally, the large, complicated and poorly specified software projects are vulnerable to crisis. The major reasons for software crisis are enlisted below :-

a) Development of powerful computer hardwares such that the programmers are unable to pace up with such hardware capabilities.
b) Poor quality of software resulting in malfunctioning.
c) Dissatisfaction amongst the uses of the software.
d) Projects running over budget or over time.
e) Difficulty in maintenance of software system.

For eg: Y2K problem is the well known example of software crisis. In 19's, softwares used two digit number to specify year i.e 20 for 1920 but the developers do not keep in mind that 2000 will also come in future which leads to dissatisfaction and inefficient software causing software crisis.

1.2 Software Characteristics

a) It should meet all the specifications of the users.
b) It should be easy to operate by users i.e. user friendly.
c) It should not produce bugs or faults while running.
d) The datas provided to it should be secured from external threat.
e) It should be easy to maintain and reuse.
f) It should be able to execute across all platforms.
g) It should be easy to upgrade.
h) It should be extensible.

1.3 Software Quality Attributes

→ Runtime Qualities
- a) Functionality
- b) Performance
- c) Security
- d) Availability
- e) Usability
- f) Interoperability

→ Non-runtime Qualities
- a) Modifiability
- b) Reusability
- c) Portability
- d) Integrability

1.4 Software Process Models

Software process model is the abstract representation of any software process considering a particular perspective.

1.4.1. Waterfall Model

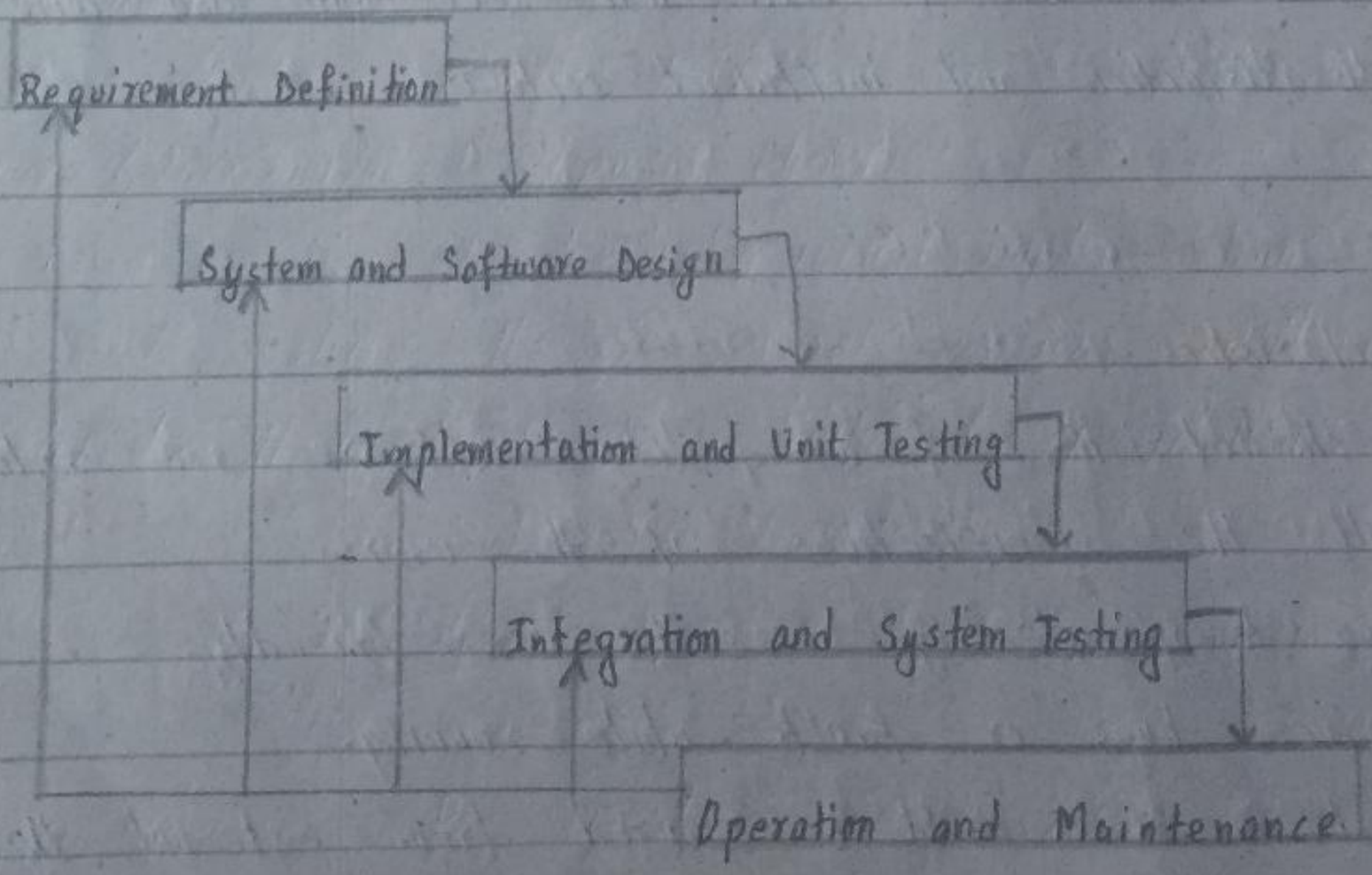
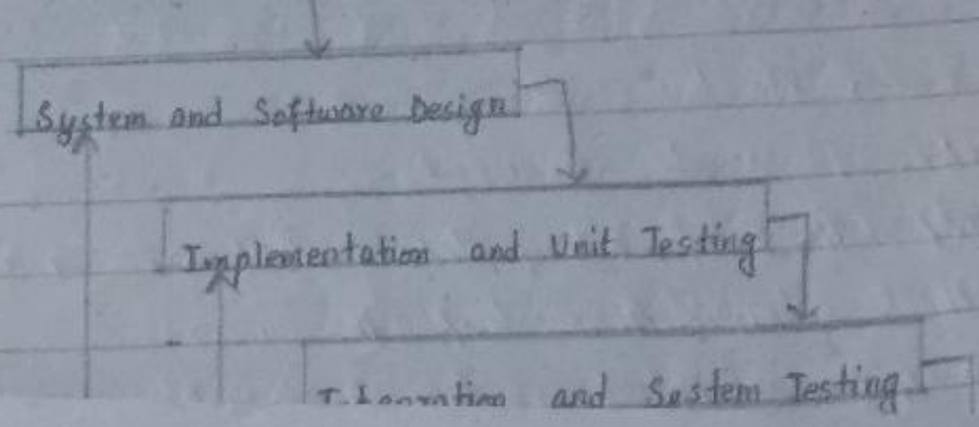


Fig. Waterfall model



The waterfall model is the general software process model as shown in above figure. It is called so because of the cascade from one phase to another. The following phase should not be started until the previous phase has finished.

The principal stages of waterfall model are explained below:-

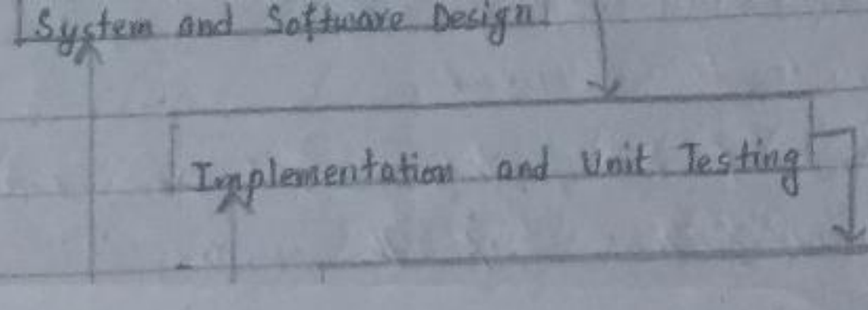
a) Requirement Definition: In this phase, the developer interacts with the users to establish the system requirements, which is then analyzed and defined detailly.

b) System and software design: In this phase, the requirements are partitioned as necessary to hardware or software system. It also involves identification and designing of software system abstractions and their relations.

c) Implementation and unit testing: In this phase, software design is realized as a set of program units and each units are implemented and tested whether they meet the requirement criteria.

d) Integration and system testing: All the program units are integrated to form complete system, which is tested to ensure fulfillment of user requirements and then delivered to its customer.

e) Operation and Maintenance: It is the longest phase in which software is installed and used practically. The errors not recovered earlier, if detected are maintained to improve the system.



Advantages

a) It reflects systematic way of software process.

b) It is useful for larger projects.

Disadvantages

a) It is impossible to partition into distinct stages.

b) It is difficult to respond to change in users requirements.

c) Waterfall model is not the accurate reflection of software development activities

In practice, the stages can overlap and feed information to each other. During practical design, the process is a sequence of iterations of these stages rather than linear model. But if iteration done as per the waterfall model, production cost and time significantly increases and the process becomes inefficient. Also, in practice, certain problems are left for future resolution and are ignored during designing which is totally against the protocol of waterfall model.

1.4.2 Evolutionary / Incremental Development Model

It is based on idea of developing initial implementation, exposing to user comments and evolving through several versions until complete system has been developed. It interleaves the activity of specification, development and validation.

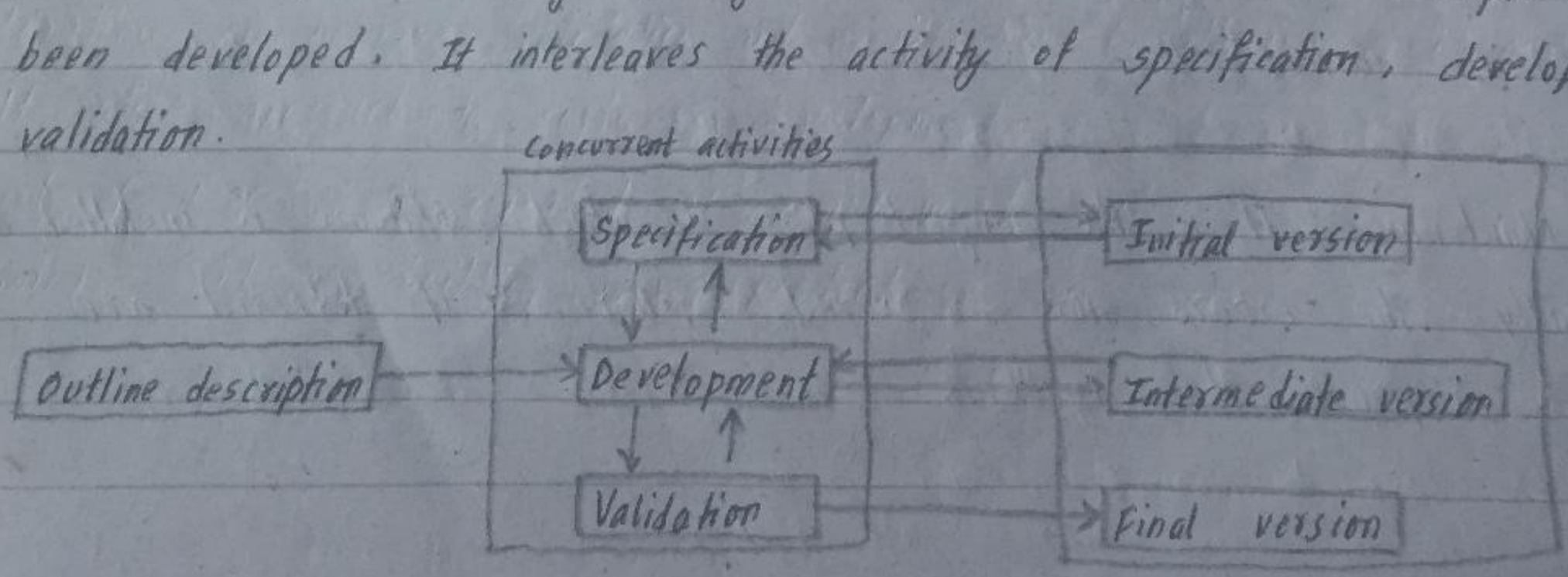


Fig. Evolutionary model

Advantages

a) The change in user requirements are easily accomodated.

b) It obtains user feedback during development process

c) It ensures rapid delivery.

Disadvantages

a) It is hard to identify common facilities needed by all increments.

b) It do not provide comple system specification at early stage.

1.4.3 Reuse oriented software engineering model / Component Based

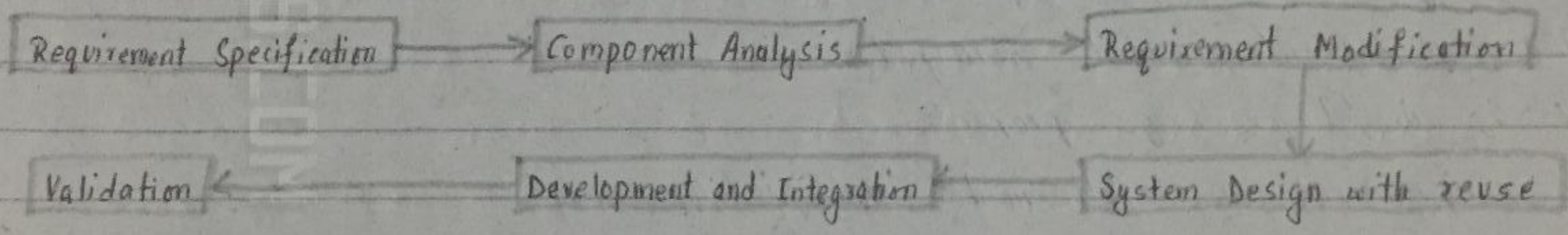


Fig: Reuse oriented model

Advantages

a) It reduces amount of software to be developed.

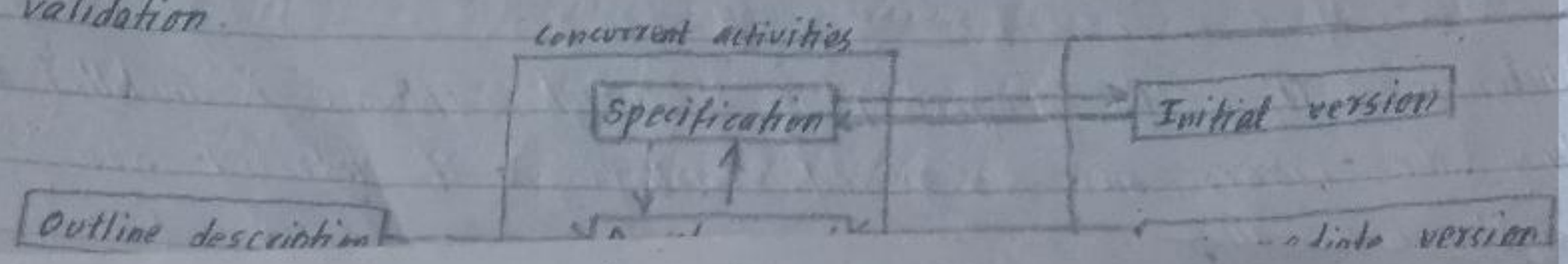
b) It reduces cost and risk.

c) It ensures rapid delivery of software.

isadvantages

a) It may lead to system that does not meet user requirements.

and validation.



1.4.4 Spiral Model

Software process is represented as spiral with some backtracking from one activity to another. Each loop of a spiral represents a phase. Each loop is divided into four sectors:

a) Objective setting
b) Risk assessment and reduction
c) Development and validation
d) Planning

Advantages

a) It recognizes risk explicitly.
b) The requirements can be managed.
c) It is useful for large projects.
d) It compromises waterfall model and prototype model.

Disadvantages.

a) It is not suitable for smaller project.
b) It is not suitable for changes that happen frequently.

1.5 Process Iteration

→ Iteration can improve validation and verification by allowing earlier quality feedback.
→ Alternative approach to software development.
→ Parts of process are repeated as requirement evolve.
→ Minimize risk of building wrong product.

→ Minimize risk of building wrong product.

1.6 Process Activities

A software process consists of a set of activities that leads to the production of software. All software process must include following activities:-

a) Software specification / Requirement engineering
b) Software design and implementation
c) Software validation
d) Software evolution

1.6.1 Software Specification

Software specification is the process of understanding and defining what services are required from the system and identifying the constraints needed for software development. It is a critical stage as error in this stage may lead to software crisis. The software specification process is depicted in the figure below:

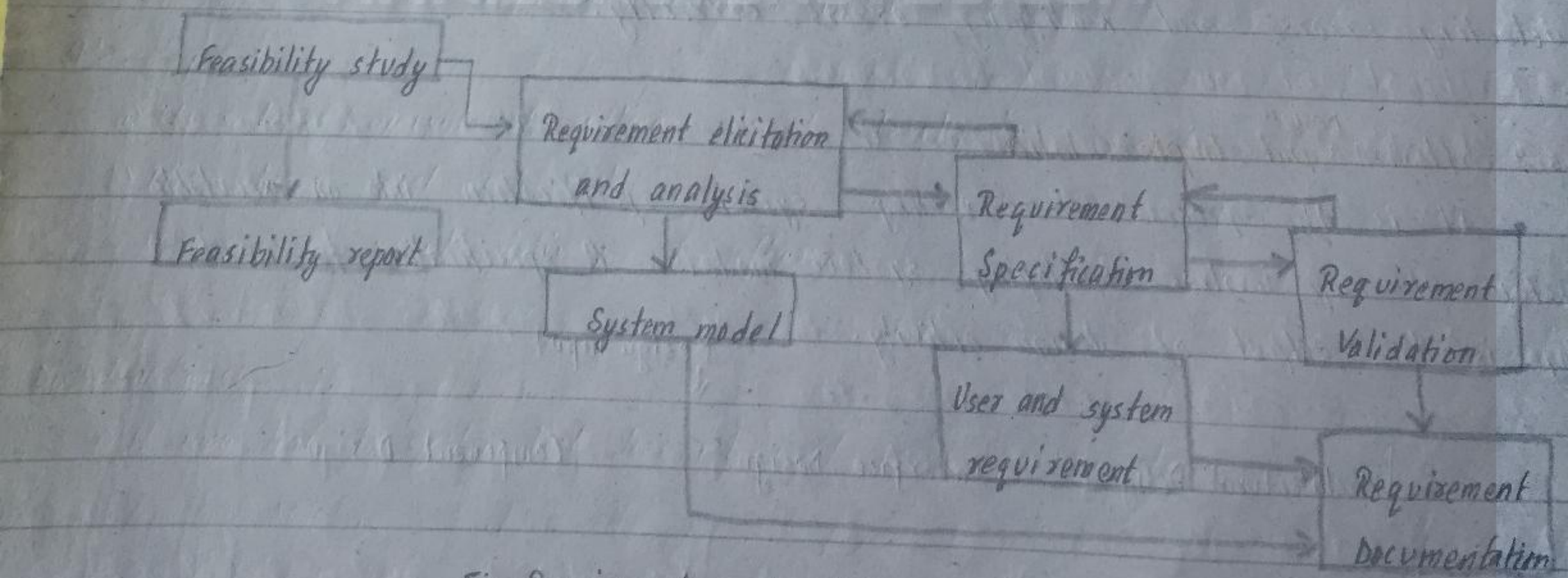


Fig. Requirement engineering process

a) Feasibility Study: It is an estimation of whether or not the user can be satisfied with current hardware and software technology and determine cost effectiveness of the system. It provides information to the developer whether or not they should move ahead for detail analysis.

b.

b) Requirement elicitation and analysis: It involves discussion with users and derive a system requirements and analyze them to form a system model or prototype.

c)

c) Requirement Specification: It involves making document relating to information gathered during analysis as a set of requirements. Requirements may be user or system requirement.

s.

d) Requirement Validation: It involves testing the completeness of requirements, detecting errors and modifying errors correctly.

1.6.2 Software Design and Implementation

It is the process of converting system specification into an executable system. It may involves developing system models. A specification for next stage is the output of design activity.

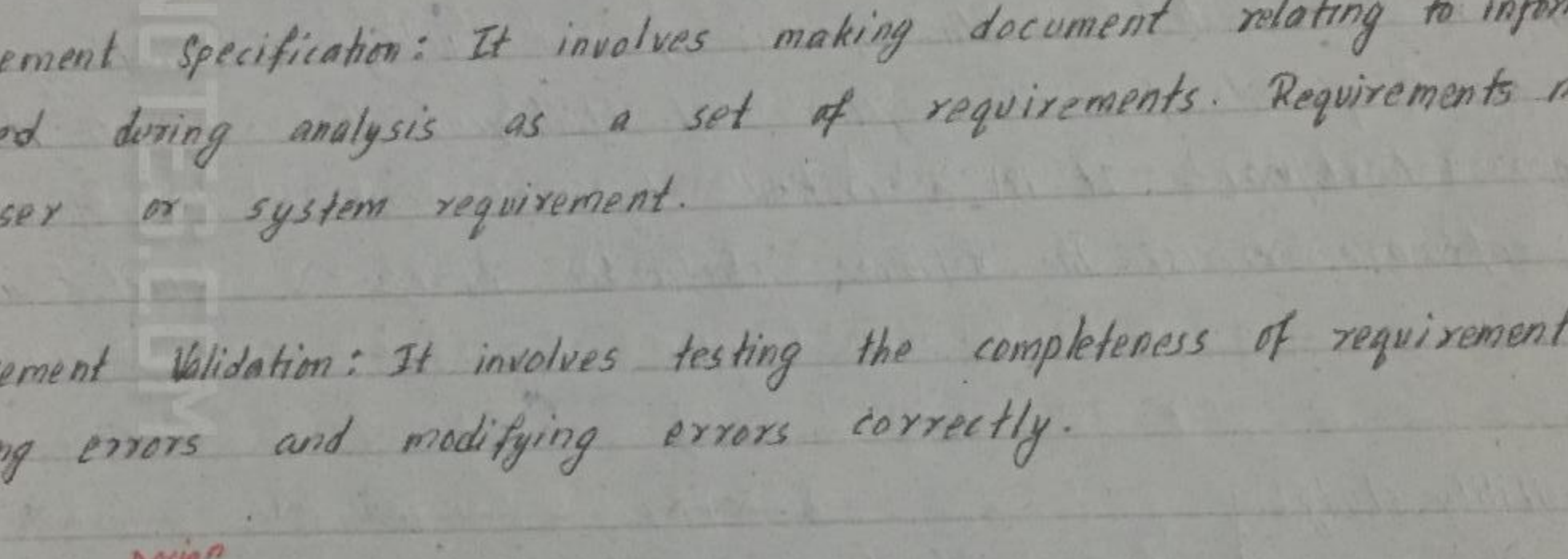


Fig. Software design process

a) Architectural Design: It involves identification of overall system and relationship between different components.

b) Interface Design: It involves defining the interfaces between system components.

c) Component Design: It involves designing the operation of each system component.

d) Database Design: It involves designing of system data structure and representing in a database.

1.6.3 Software Validation

Software validation is a test of whether or not system meets specification and user's expectation. It also involves checking processes.

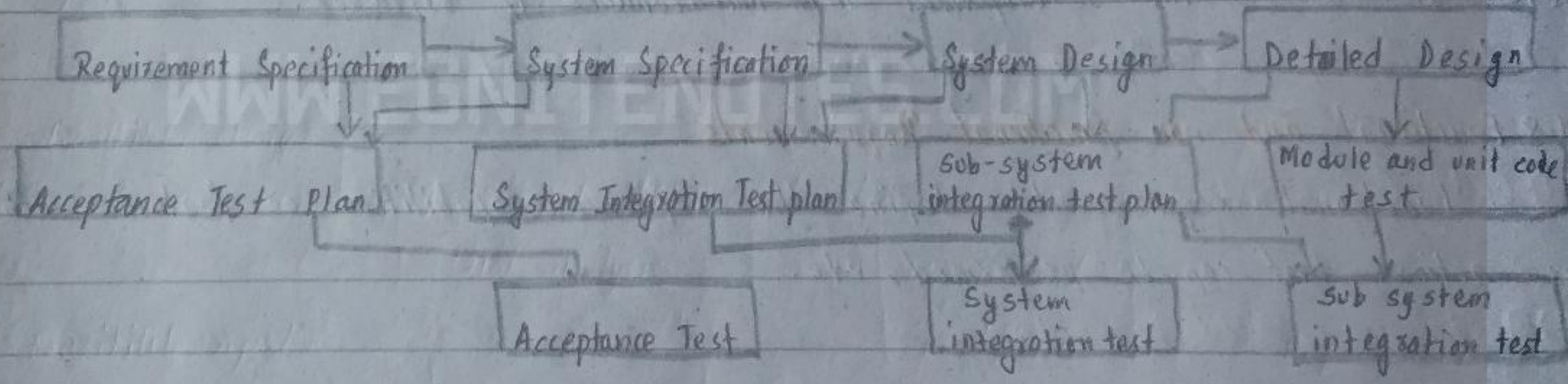


Fig: Testing Phase

a) Development Testing: Each component that make up the system is tested separately by the developer.

b) System Testing: The components are integrated and complete system is tested.

c) Acceptance Testing: The system is tested with data supplied by a customer.

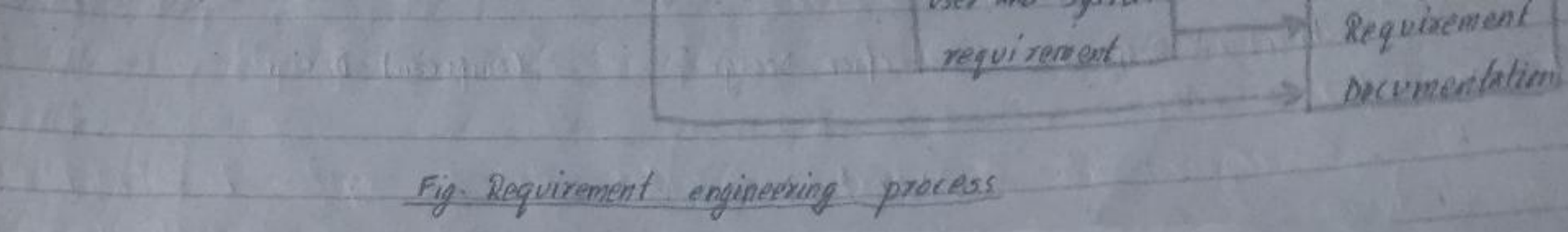
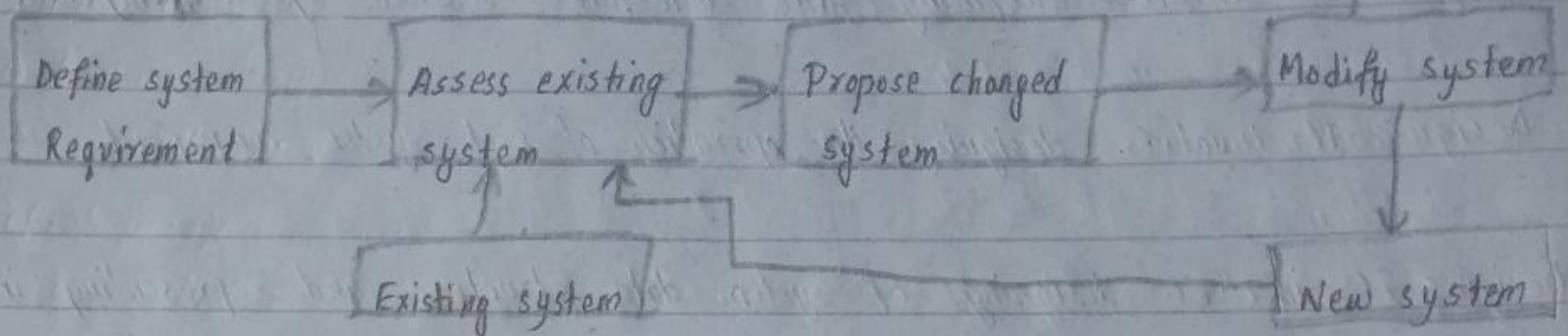


Fig. Requirement engineering process

1.6.4 Software Evolution

Software evolution is the process of maintaining software over time in response to changing needs and requirements of user.



1.7 Computer Aided Software Engineering (CASE)

CASE tools are the software that are used to support software process activities and includes design editor, debugger, compiler, system building tools, etc. It assists the development and maintenance of software.

1.8 Functional and non-functional Requirements

Functional requirements define the software capabilities or features such as actions that can be performed, etc.

Non-functional requirements define the software properties such as reliability, performance, security, response time, etc.

1.9 User Requirements

The user requirements of a system describes what the system is intended to do in terms of functional and non-functional requirements understandable by the system users. It can be defined using natural language, tables, etc.

Non-functional requirements define the software properties such as performance, security, response time, etc.

1.9 User Requirements

The user requirements of a system describes what the system is intended to do in terms of functional and non-functional requirements understandable by the system users. It can be defined using natural language, tables, etc.

1.10 System Requirements

System requirements are expanded versions of user requirements that are used by software engineers for system design. It includes how user requirements should be provided by the system. They may be defined using system models.

1.11 Interface Specification

1.12 Software Requirements Document/Specification (SRS)

SRS is an official statement of what the system developers should implement. It includes both user requirements and system requirements. It serves as an agreement between developers and customers on what the application will do.

The characteristics of a good SRS are:-

a) Complete
b) Consistent
c) Correct
d) Modifiable
e) Ranked
f) Testable
g) Traceable
h) Unambigious
i) Valid
j) Verifiable

1.13 Feasibility Study

The report should contains:-

1) Purpose
2) Statement of Problem
3) Criteria for feasibility
4) Analysis Procedure
5) Conclusion

1.9 User Requirements

The user requirements of a system describes what the system is intended to do in terms of functional and non-functional requirements understandable by the system users. It can i.e. i.e. .

#Requirement Gathering Technique

1) Interview
2) Questionnaire
3) Task Analysis
4) Domain Analysis
5) Brainstorming
6) Prototyping
7) Observation

#CASE Tools

1) Prototyping Tool
   → GUI development
   → Support mock up run of actual system

2) Structured analysis & design tool

3) Code Generation tool

4) Test case generation tool

Chapter - 2
System Modeling

2.1 System Modeling

System modeling is the process of developing abstract model of a system based on analyzed requirements such that each model presents a different view of the system.

2.2 Context Model

Context models are used to illustrate the boundaries between the system and the environment. Such system boundaries should be decided during the requirement elicitation and analysis process. It shows the interaction of external entities with the system

It consists of only one process i.e. overall system process and its relationships with other external entities.

Eg: Consider an ATM system. In context model, the entire system is represented as a single process. Here, the external entities interacting with the system may be security system, branch accounting system, branch counter system, maintenance system, usage database and account database.

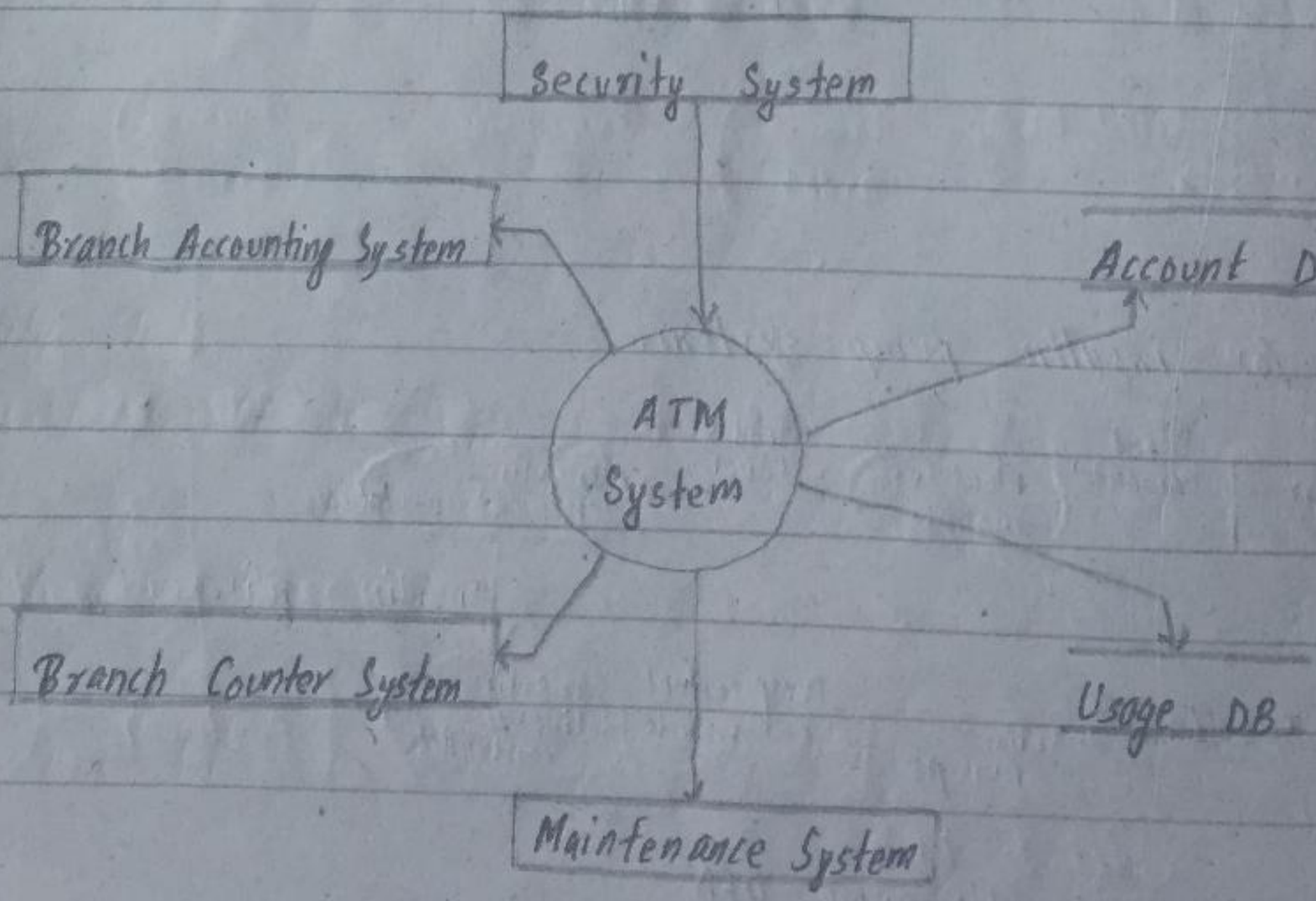


Fig. Context Diagram of ATM system

2.3 Behavioral Model

Behavioral model are used to describe the overall behaviour of the system. It may be data flow model (models the flow of data in the system) or state machine model (models how the system reacts to various events).

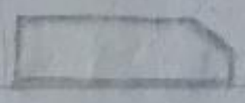
2.3.1 Data Flow Model

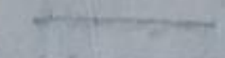
Data flow model shows a functional perspective of a system. It shows each transformation as a single function or process and how data flow through these processing. At each step, first data is transferred and we move to next step.

Each function consumes some input data and produces some o/p data. Data flow model is modelled using Data Flow Diagram (DFD). The symbols used in DFD are :-

a) External entity 

b) Process 

c) Output 

d) Data Store 

e) Data Flow 

Eg: A level 1 DFD for insulin pump system.

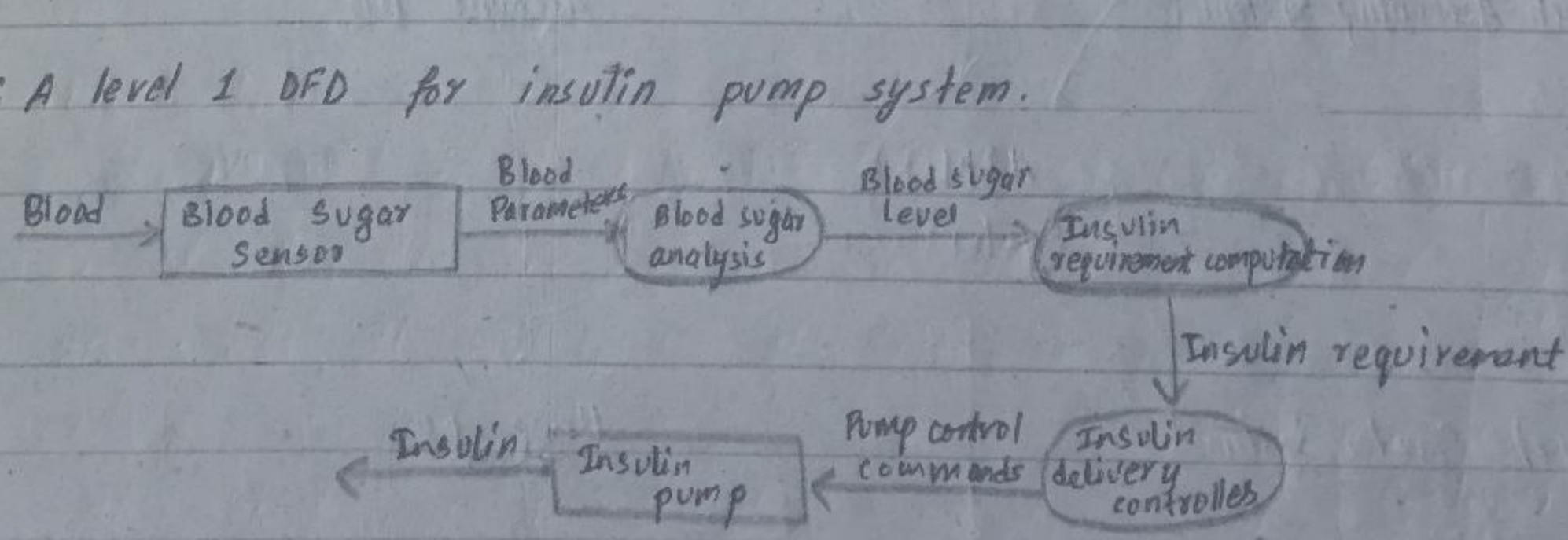


Fig. Insulin pump DFD

Blood sugar analysis

Blood sugar level

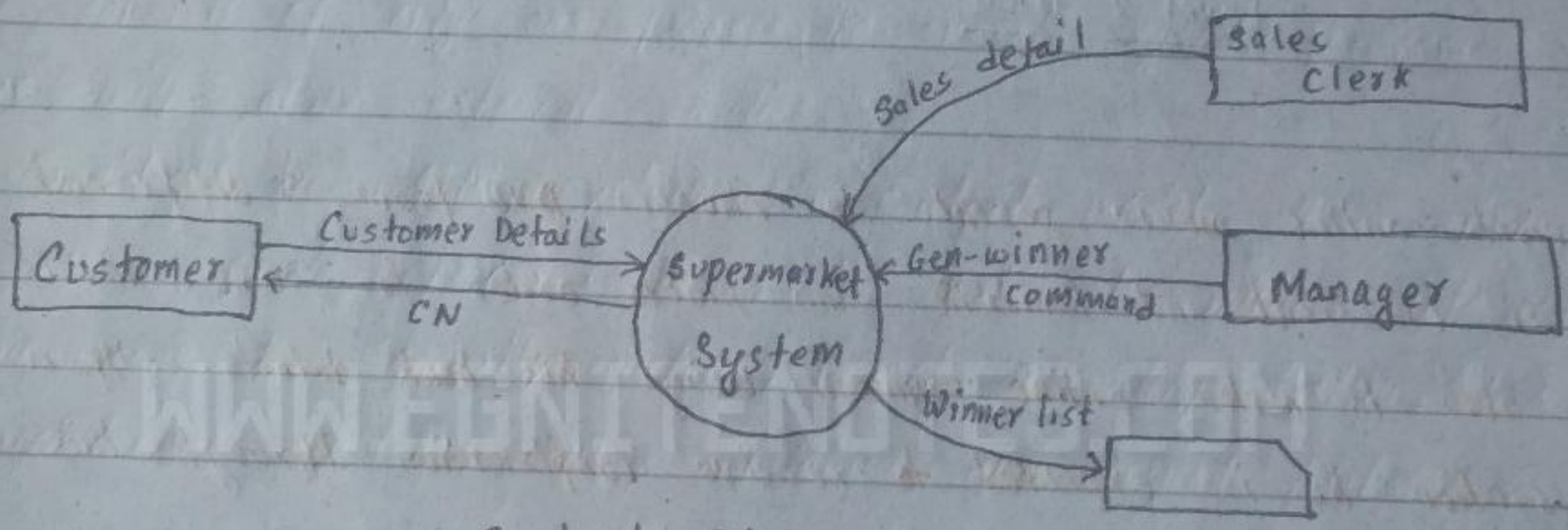
Insulin requirement computation

Insulin

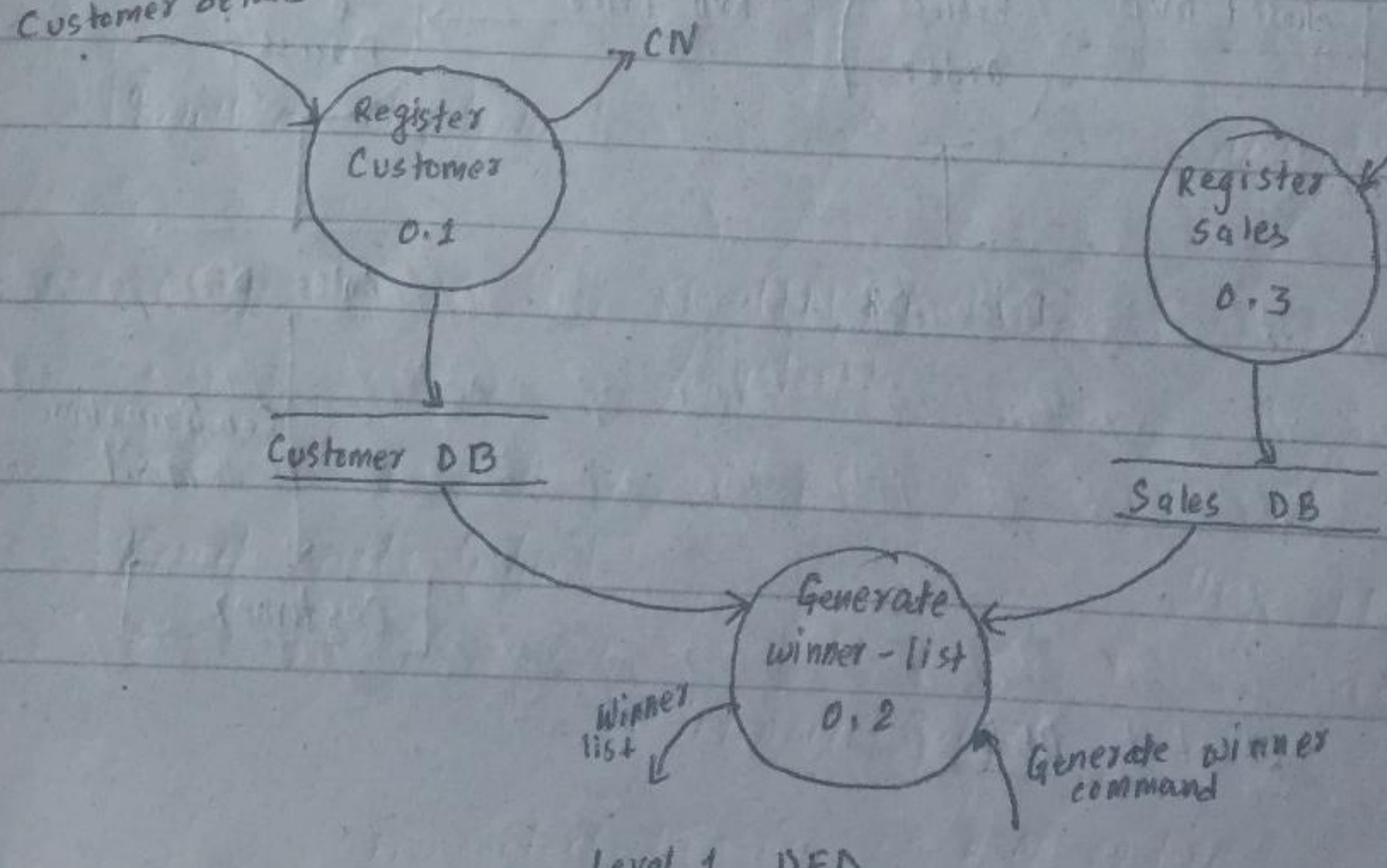
17

Q) A supermarket wants to develop following software. For this, customer needs to supply his/her residence address, telephone no and driving license number. Each customer who registers is assigned a unique customer number (CN). A customer can present his CN to the check out staff when he makes purpose. In this case, the value of his purphase is credited against his CN. At ~~customers~~ end of each year, supermarket intends to award surprise gifts to 10 customers who make highest total purchase. Also intends to award 22 carat gold coin to customer whose purchase exceeded Rs. 50000. The entries against CN are reset on the day after prize winners' lists are generated. Draw context diagram, level 1 DFD and level 2 DFD.

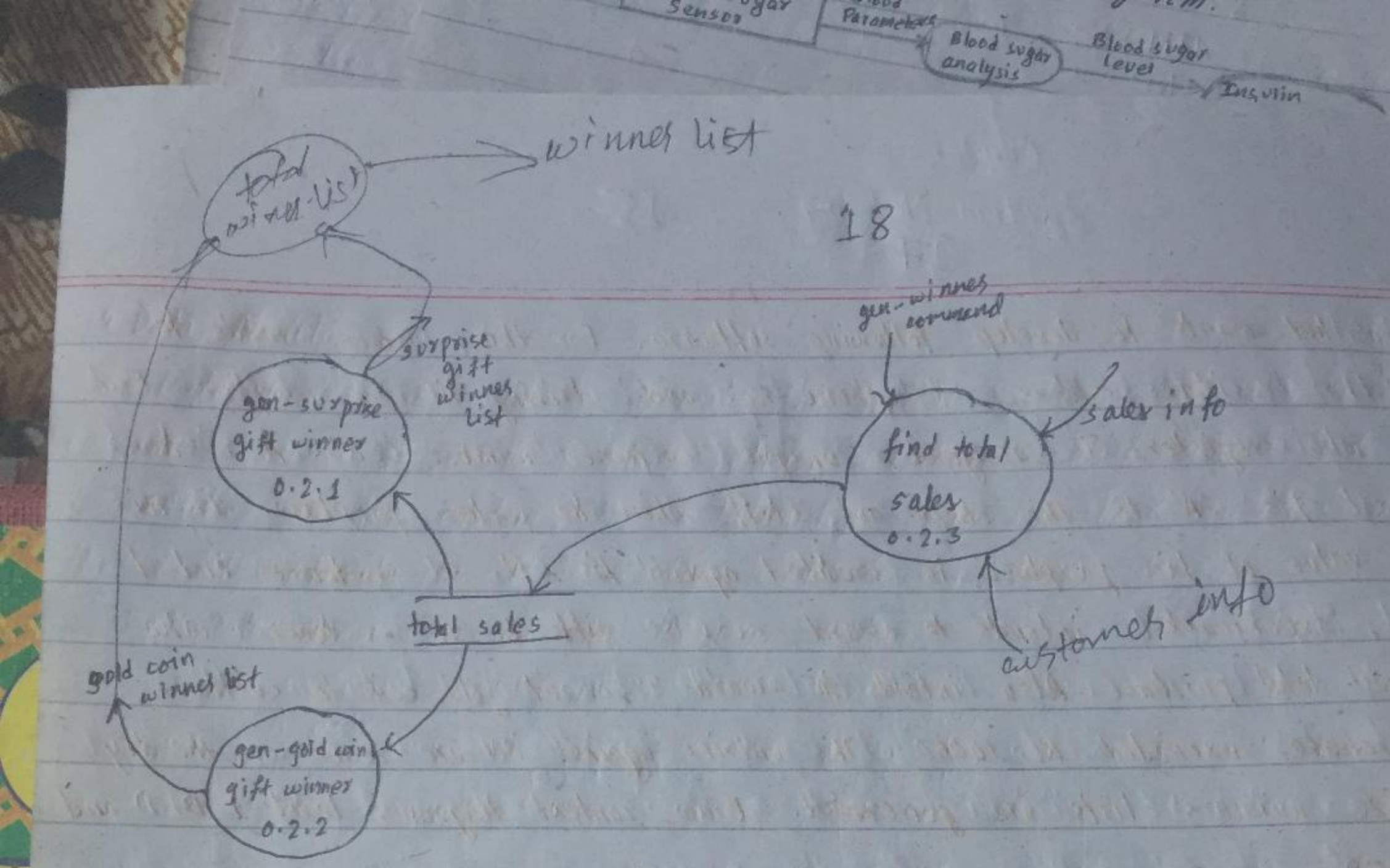
Ans →



Context Diagram

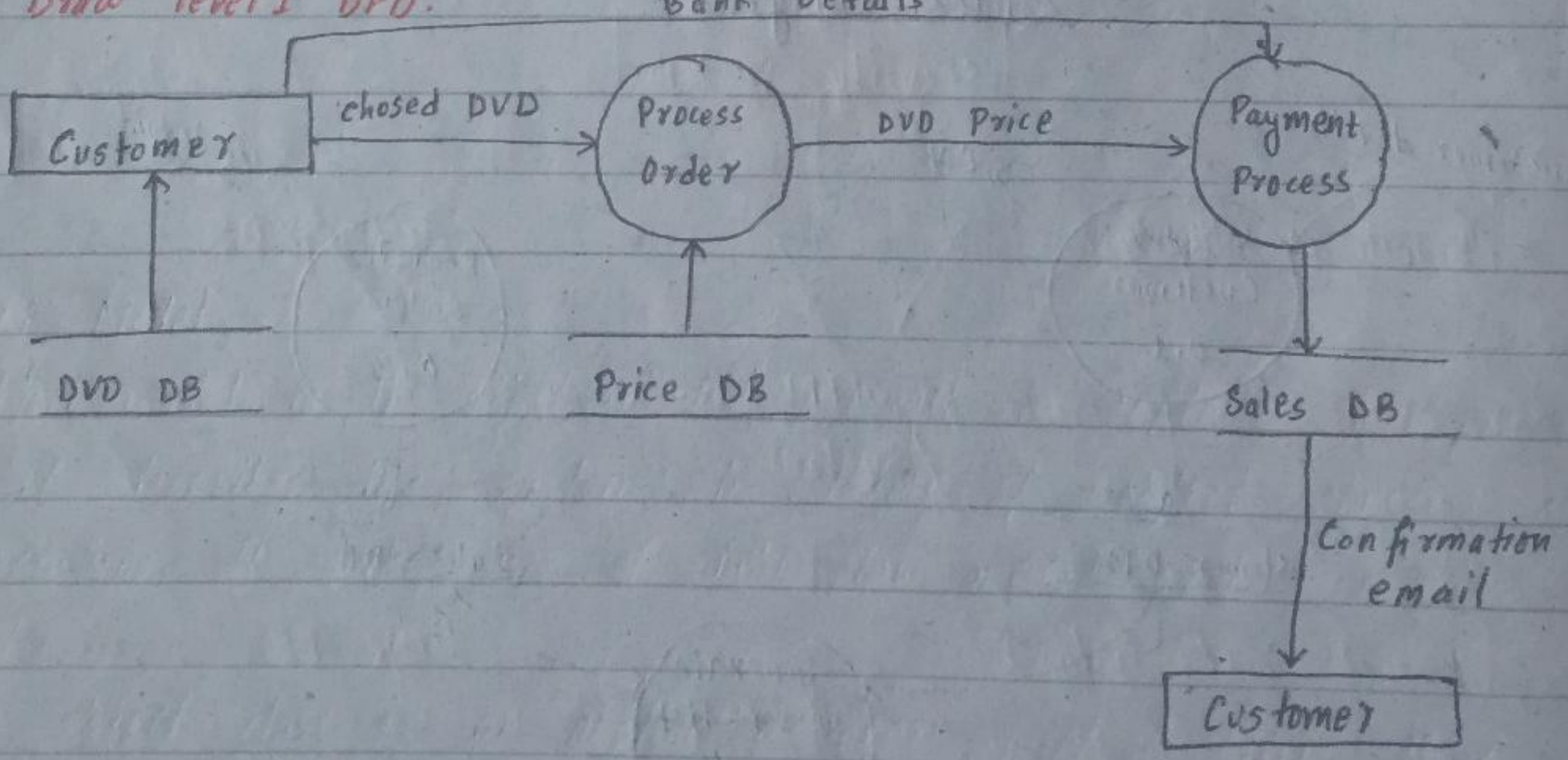


Level 1 DFD

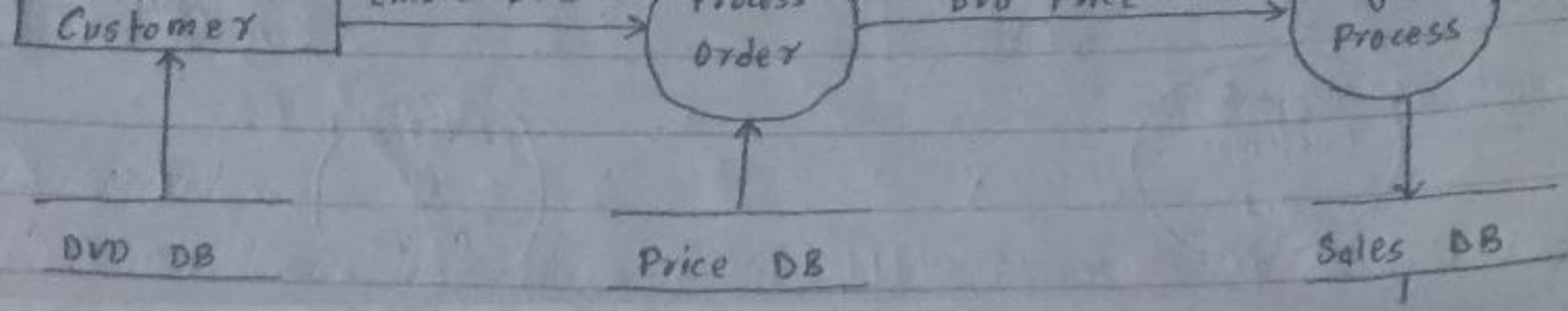


Level 2 DFD

Q2) A customer visits online movie portal. He chooses DVD from 3 different categories: Sci-Fi, Classical and Romantic and places order for same. He is supposed to be able to make online payment using his bank details. Upon successful transaction he is expected to receive confirmation through his e-mail. Draw level 1 DFD.



Level 1 DFD



2.3.1.1 Shortcomings of Data Flow Model

a) A single bubble may not explain whole functionality of the process.

b) Control aspects are not mentioned.

c) For a single problem, DFD may vary according to the thinking of an analyst.

d) It does not provide guidance to decompose a function into sub-functions.

2.3.1.2 Balancing DFD

Balancing DFD means preserving all the incoming and outgoing data flows from a process in the parent diagram at the next level of decomposition.

2.3.2 State Machine Model

This model is used to depict the behaviour of the system in response to external and internal events. They are often used in modeling real time systems.

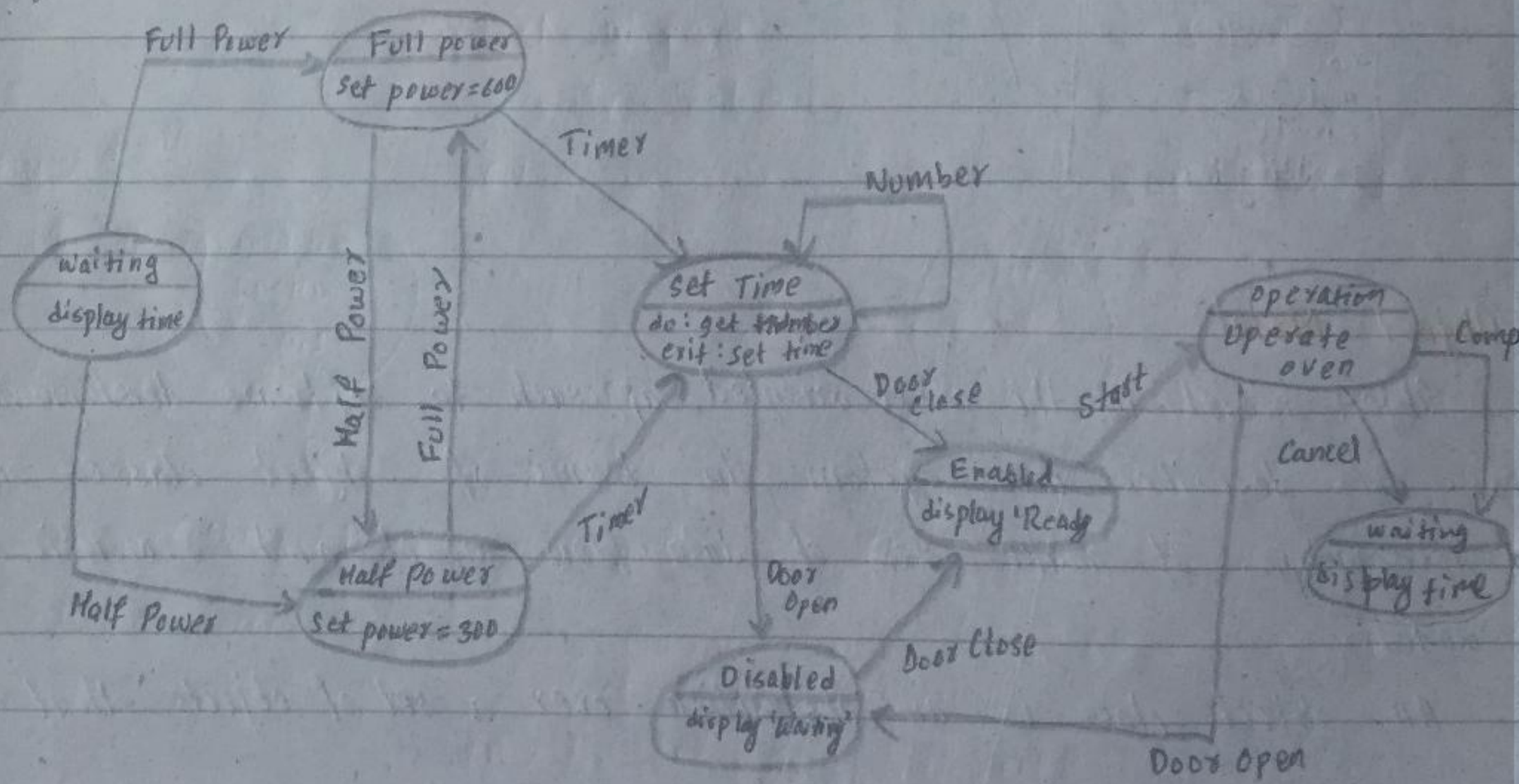
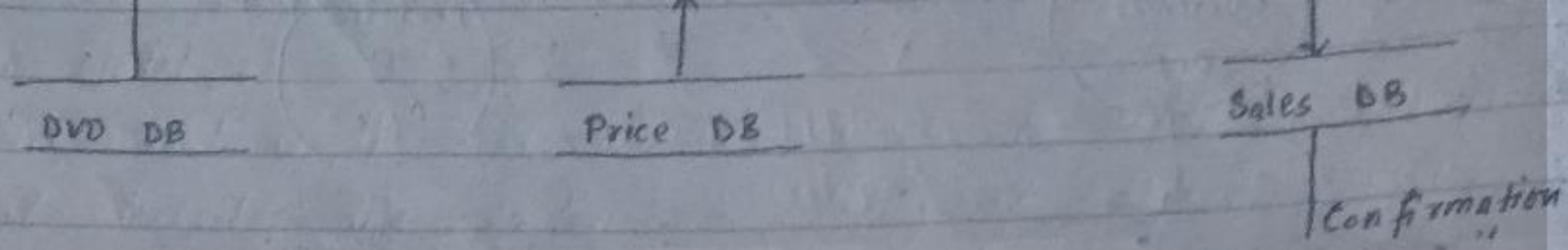


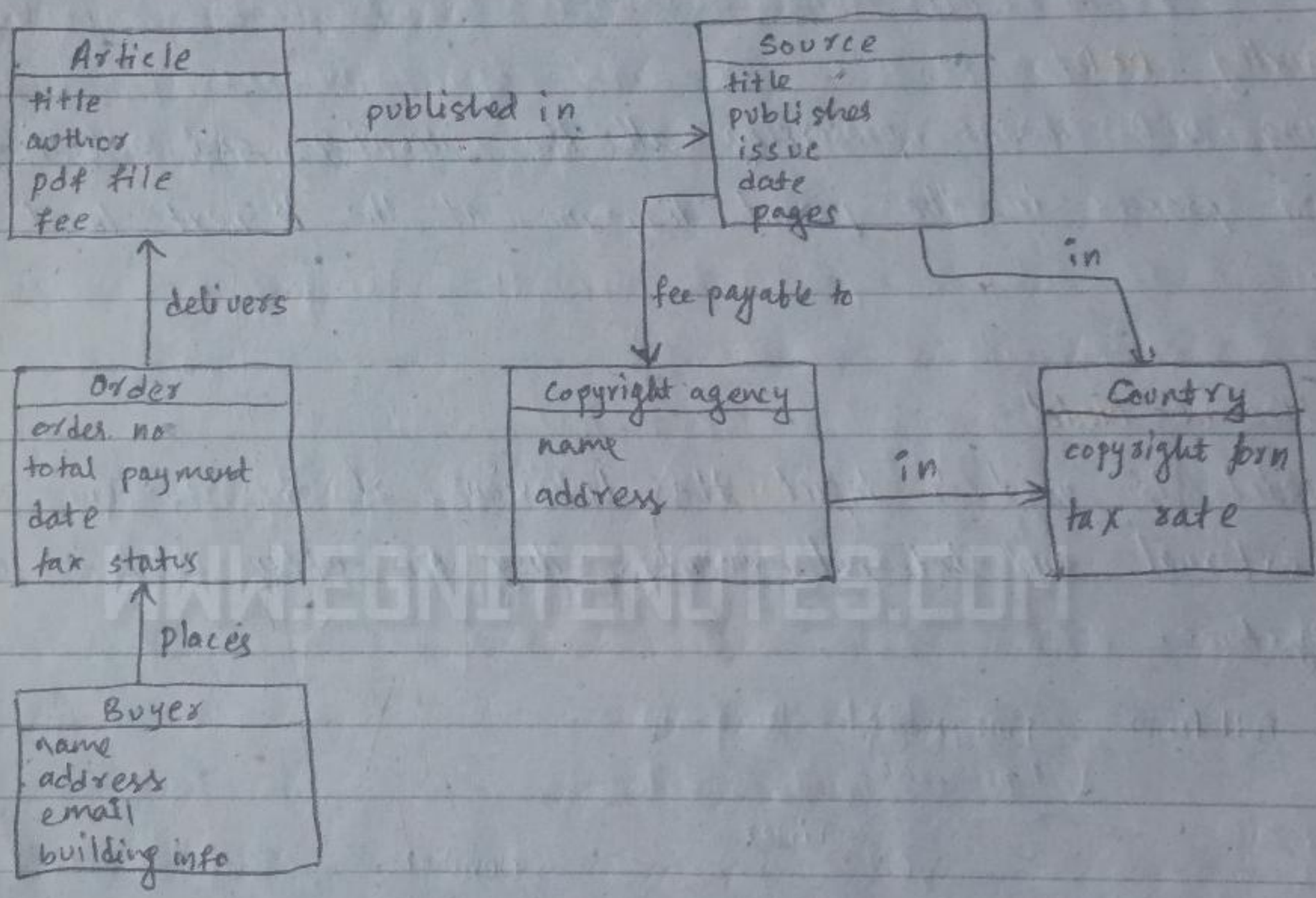
Fig. State Machine Model for microwave oven



2.4 Data Model

Data model is used to describe the logical structure of data processed by the system. It is an entity relation attribute model which sets out the entities in the system, the relationship between these entities and the entity attributes. It is mostly used for database design.

Eg: Data semantic model for library system.



2.5 Object Model

Object model is the object oriented approach to software development process. It describes the system in terms of object classes and their associations. It makes use of data flow model and semantic data model.

An object class is an abstraction over a set of objects that

Object model is the object oriented approach to software development process. It describes the system in terms of object classes and their associations. It makes use of data flow model and semantic data model.

An object class is an abstraction over a set of objects that

identifies common attributes and services provided by each object. It models real world entities using object class.

Object models may be:

a) Inheritance model

b) Aggregation model

c) Interaction model

2.6 Structured Methods

A structured method is a systematic way of producing system models by providing a framework. They define a process that may be used to derive models and a set of protocols that apply to the models. CASE tools support system modelling as a part of a structured method.

2.6.1 Shortcomings of structured methods

a) They do not model non-functional requirements.

b) They do not include information about whether a method is appropriate for a given problem.

c) They may produce too much documentation.

d) The system models may become detailed and difficult to understand for users.

Chapter - 3
Architectural Design

~~3.1 Architectural Design~~

Questions

1) Why architectural design is necessary before specifications are written? Explain client server architecture with example. [4+5]

2) Compare client server and distributed object architecture. [5]

3) How modular decomposition is practiced in system design? Illustrate with example of level-2 DFD. [4+6]

4) Provide brief comparison of multiprocessor and client-server architecture. [4]

5) What is role of reference architecture in system design process? Justify with example. [6]

6) Compare distributed object and multiprocessor architecture. [4]

7) What is architectural design? Why is it important in s/w engineering? Explain multiprocessor architecture with example. [2+3+5]

8) What are basic concept of modular programming? Explain with example. [5]

3.1 Architectural Design

→ Architectural design is the process of identification of the sub-systems of a system and establishment of a framework for their control and communication.

→ It establishes a basic structural framework that can identifies the components of a system and their relationship.

→ It serves as a design plan that is used to negotiate system requirements.

→ It helps in complexity management.

→ It can decide whether system can meet critical requirements like performance, reliability, etc as system analysis is done.

→ It helps to structure and organize the specifications.

3.2 System Organization

→ It includes the basic strategy used to structure the system.

→ The types of organizational styles are:-

1) Repository model

2) Client Server model

3) Layered model / Abstract machine model

→ Repository is a shared database that holds large amount of data of a system.

→ In layered model, system is organized into layers, each of which provides a set of services. Eg: OSI reference model.

3.3 Modular Decomposition

→ Modular decomposition is the process of decomposing sub-systems into modules.

→ It can be practiced in following ways:

1) Object oriented decomposition

→ A system is decomposed into a set of communicating objects.

→ It is concerned with object classes, attributes and operations.

→ On implementation, objects are created from classes.

→ Objects are loosely coupled, so its implementation can be modified without affecting other objects.

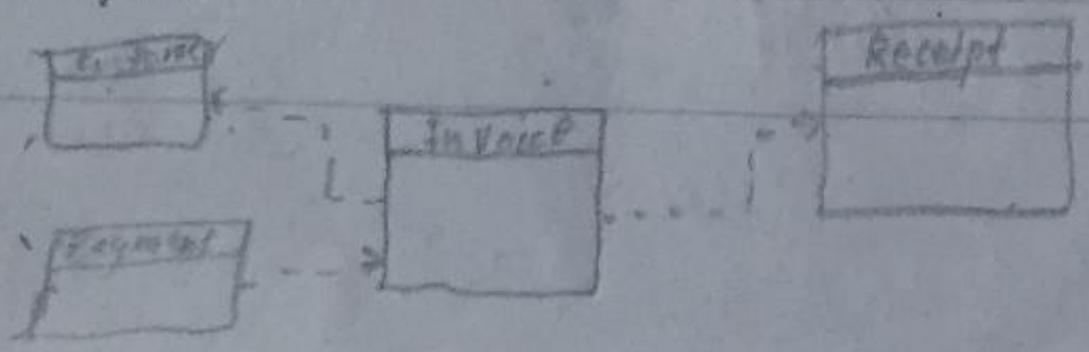
→ Objects can be reused and represent real world entities.

2) Function oriented decomposition

→ Each processing step is implement as a transform.

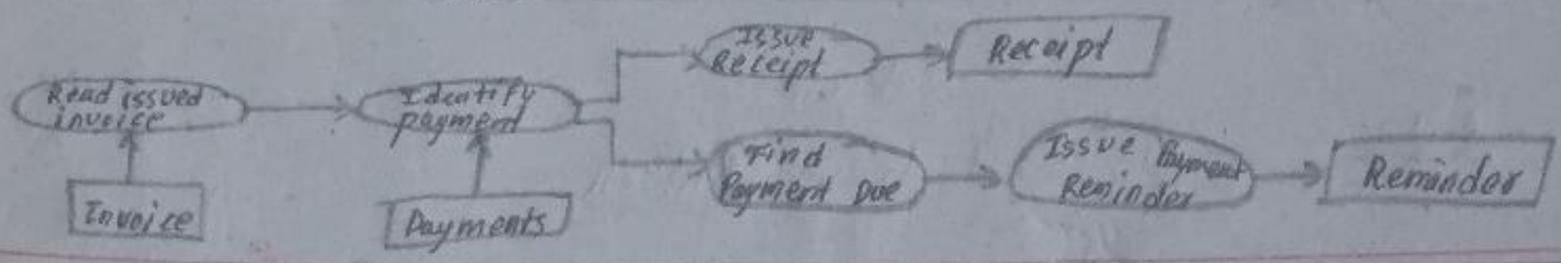
→ Input data flows through these transforms until converted to output.

→ Data flow model



affecting other objects.
→ Objects can be reused and represent real world entities.

2) Function oriented decomposition
→ Each processing step is implement as a transform.
→ Input data flows through these transforms until converted to output.
→ Data flow model

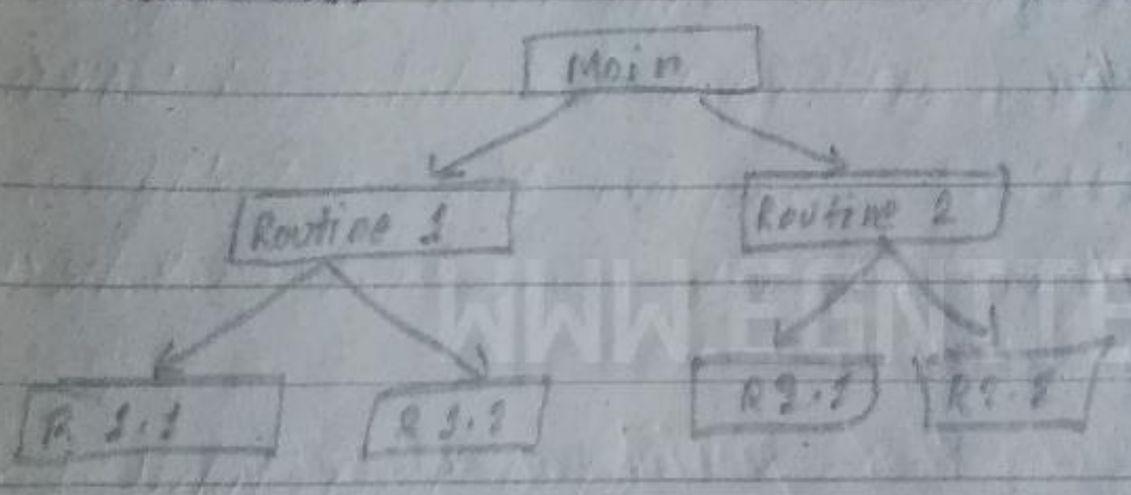


3.4 Control Styles

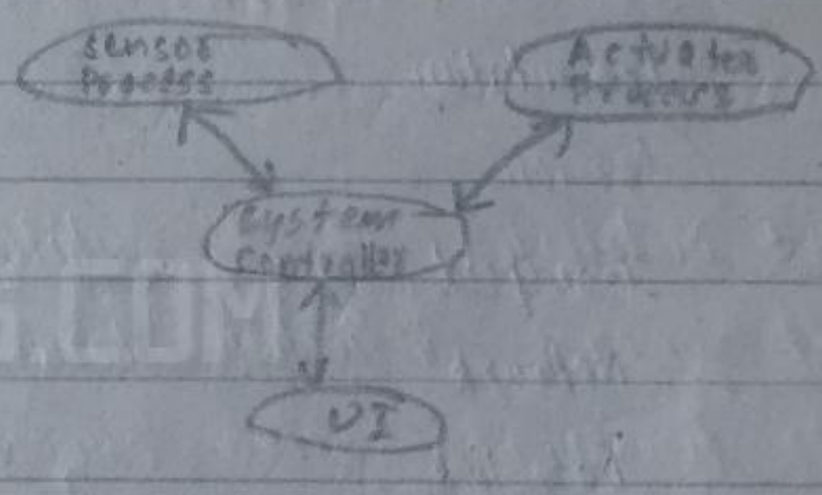
→ Control models at architectural level is concerned with control flow between sub systems.
→ To work as a system, sub systems must be controlled so that their services are delivered to right place at right time.
→ Control styles are used in conjunction with structural style.

3.4.1 Centralized Control

→ One sub system is designated as system controller that manages execution of other sub-systems.
→ It may be call-return model (sequential execution) and manager model (parallel execution)



Call return model



Manager model

→ Control decisions are determined by values of some system state variables.

3.4.2 Event-driven Control

→ Control decisions are driven by externally generated events.
→ In broadcast model, event is broadcast to all sub system, and sub system with handler for that event responds to it.
→ In interrupt driven model, external interrupts are detected by interrupt handler and are passed to some component for processing.

→ Decomposition
→ Each processing step is implement as a transform.
→ Input data flows through those transforms until converted to output.
→ Data flow model

3.6 Reference Architecture

→ The architectural models that are specific to a particular application domain is called domain specific architecture.
→ Reference architecture is an abstract model that informs designers about the general structure of the system domain.
→ It includes all the features that the system might incorporate.
→ It is used to communicate domain concepts and evaluate possible architecture
→ It is not a route to implementation.

For eg: OSI Reference model for open system interconnection.

Application		Application
Presentation		Presentation
Session		Session
Transport		Transport
Network		Network
Data link		Data link
Physical	→	Physical

→ The OSI model defines an implementation standard so that systems could communicate with each other.
→ It provides a basis for abstract structure and systematic implementation of communication between systems.

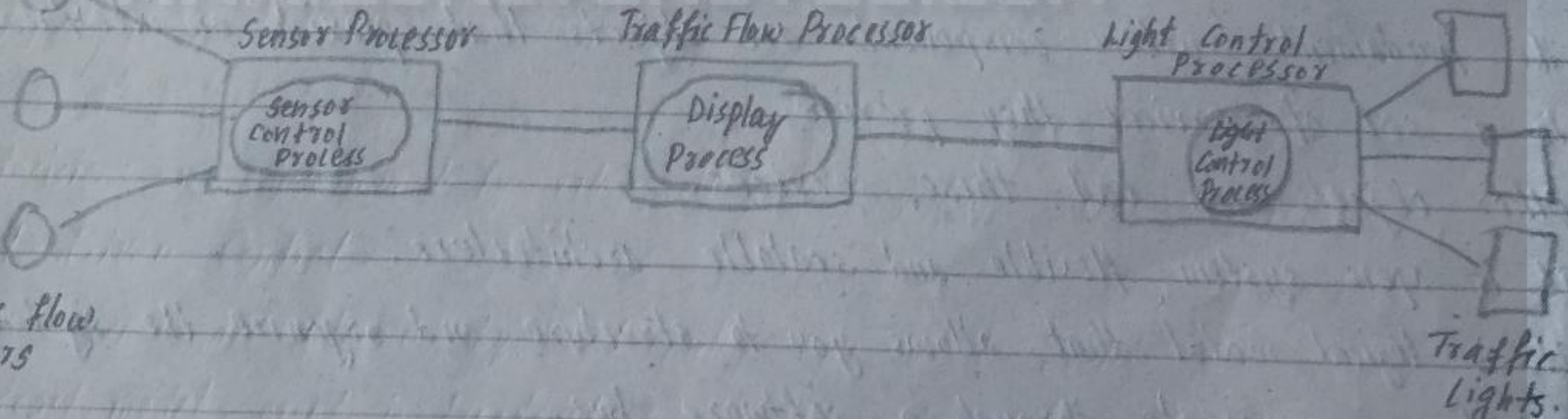
basis for abstract structure and systematic implementation
communication between systems.

3.6 Distributed System Architecture

→ Distributed system is a system in which the information processing is distributed over several computers rather than confining in a single machine.
→ It is composed of loosely coupled independent parts, each of which may interact with users directly or with other system parts.

3.6.1 Multiprocessor Architecture

→ It is the simplest model of distributed system in which software system consists of a number of processes which may execute on separate processors.
→ It is common in real time systems.
→ Use of multiprocessors improve performance.
→ The processes are distributed to processors by pre-determination or under the control of dispatcher.
→ Eg: A multiprocessor traffic control system



→ It reflects physical structure of the system.

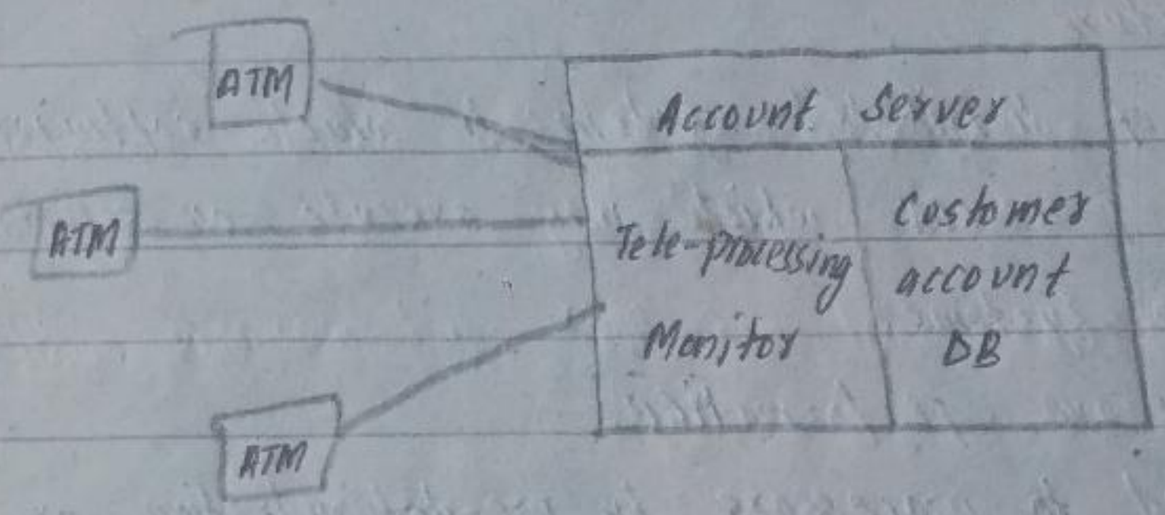
3.6.2 Client-server architecture

→ An application is modelled as a set of services that are provided by the servers and a set of clients that use the services.

→ Client and server are separate processes.

→ It reflects the logical structure of the application.

→ Eg: Client-server ATM system



3.6.3 Distributed Object Architecture

→ It removes distinction between client and server.

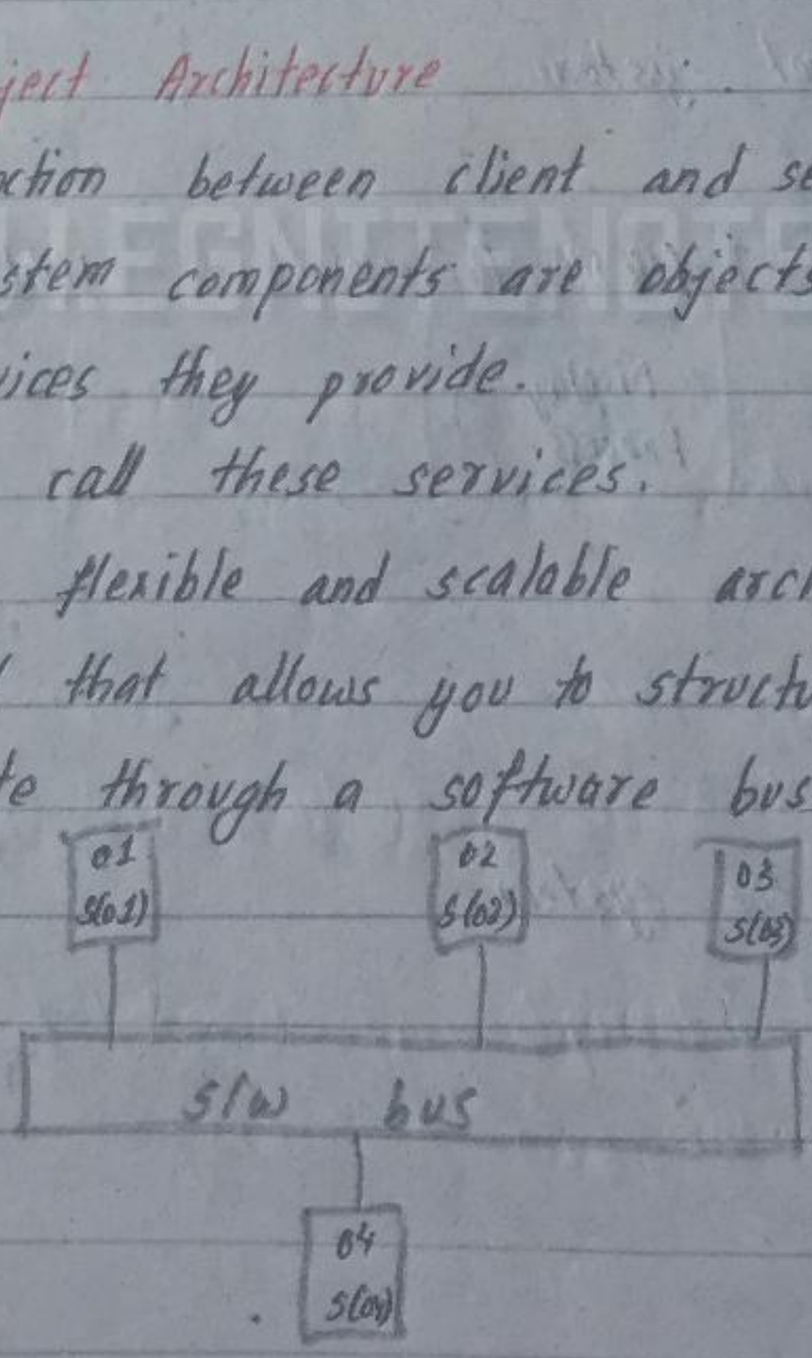
→ The fundament system components are objects that provide an interface to a set of services they provide.

→ Other objects can call these services.

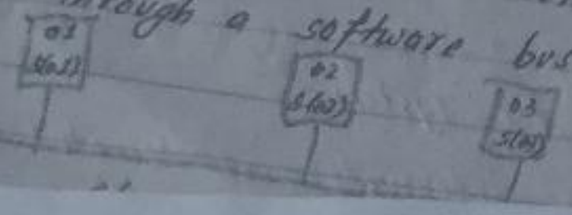
→ It is open system flexible and scalable architecture.

→ It is logical model that allows you to structure and organize the system.

→ Objects communicate through a software bus.



→ all these services.
→ It is open system flexible and scalable architecture.
→ It is logical model that allows you to structure and organise the system.
→ Objects communicate through a software bus.



Chapter - 4
Real Time Software Design

4.1 System Design
4.2 Real time operating system
4.3 Monitoring and control system
4.4 Data Acquisition System

[5-marks]

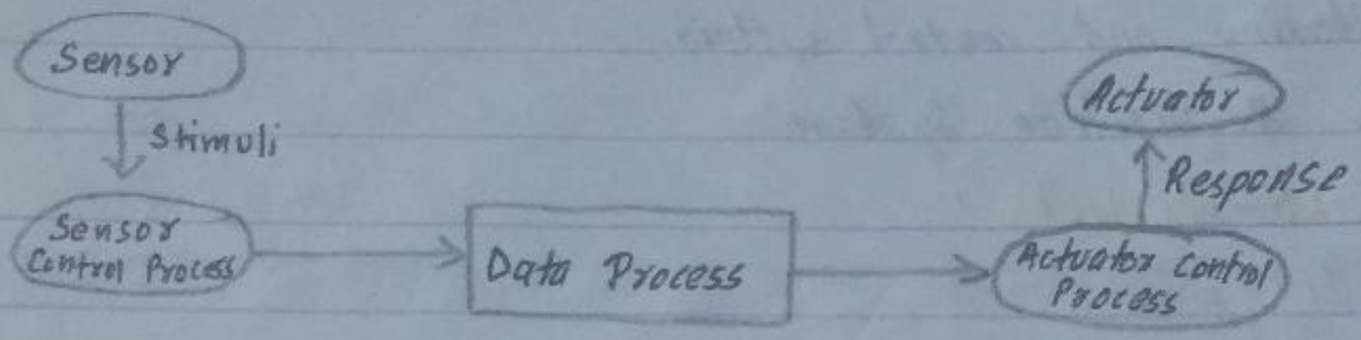
Questions

1) How do real time software and OS differ from non-real time software and OS? Describe data acquisition system. [4+4]
2) What specific considerations are to be made while designing typical software to be operated in real time environment? Explain. [5]
3) Define real time system. Explain real time OS and its components. [1+4]

4.1 Real Time System

→ A real time system is a software system in which the system functioning depends on the results produced by system due to an event and the time of result production.
→ A soft real time system is a system whose operation is degraded if results are not produced in specified time.
→ A hard real time system is a system whose operation is incorrect if results are not produced in specified time.
→ Timely response to the input stimulus is the key for real time system.
→ The behaviour of real time system can be defined by stimuli received, associated response and the time for response production.
→ Stimuli can be periodic (occur at predictable time interval) and aperiodic (occur irregularly).

→ Real time system are designed as a set of concurrent and cooperating processes.
→ The execution platform is real time OS.
→ The general stimulus - response architecture model is shown below:-



4.2 System Design

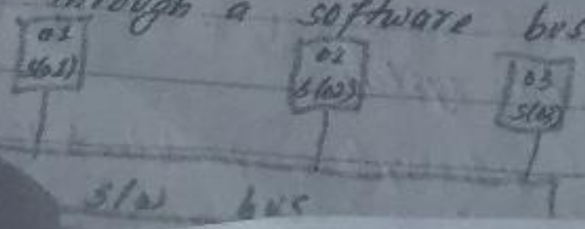
→ The interleaved stages in the system design process are as follows:-
1) Identify the stimuli to be processed and the required response.
2) For each stimulus and response, identify timing constraints.
3) Aggregate stimulus and response processing into a number of concurrent processes.
4) For each process, design algorithms that meet timing requirements.
5) Design a scheduling system.
6) Integrate using real time operating system.

→ State machine modelling is used to model real time systems.
→ This model assumes, at any time, system is in one of a number of possible states. When stimulus is received, this may cause transition to different state.

4.3 Real time Operating Systems (RTOS)

→ RTOS is a specialized OS which manages the processes in real time system.
→ It manages process and allocates memory in RTS.
→ It does not include facilities like file management.

system flexible and scalable architecture.
model that allows you to structure and organize the system.
unicate through a software bus.



4.3.1 Components of RTOS

1) Executive Components

a) Real time clock → It provides information to schedule processes periodically.

b) Interrupt handler → It manages aperiodic requests for service.

c) Scheduler → It chooses next process which is to be run.

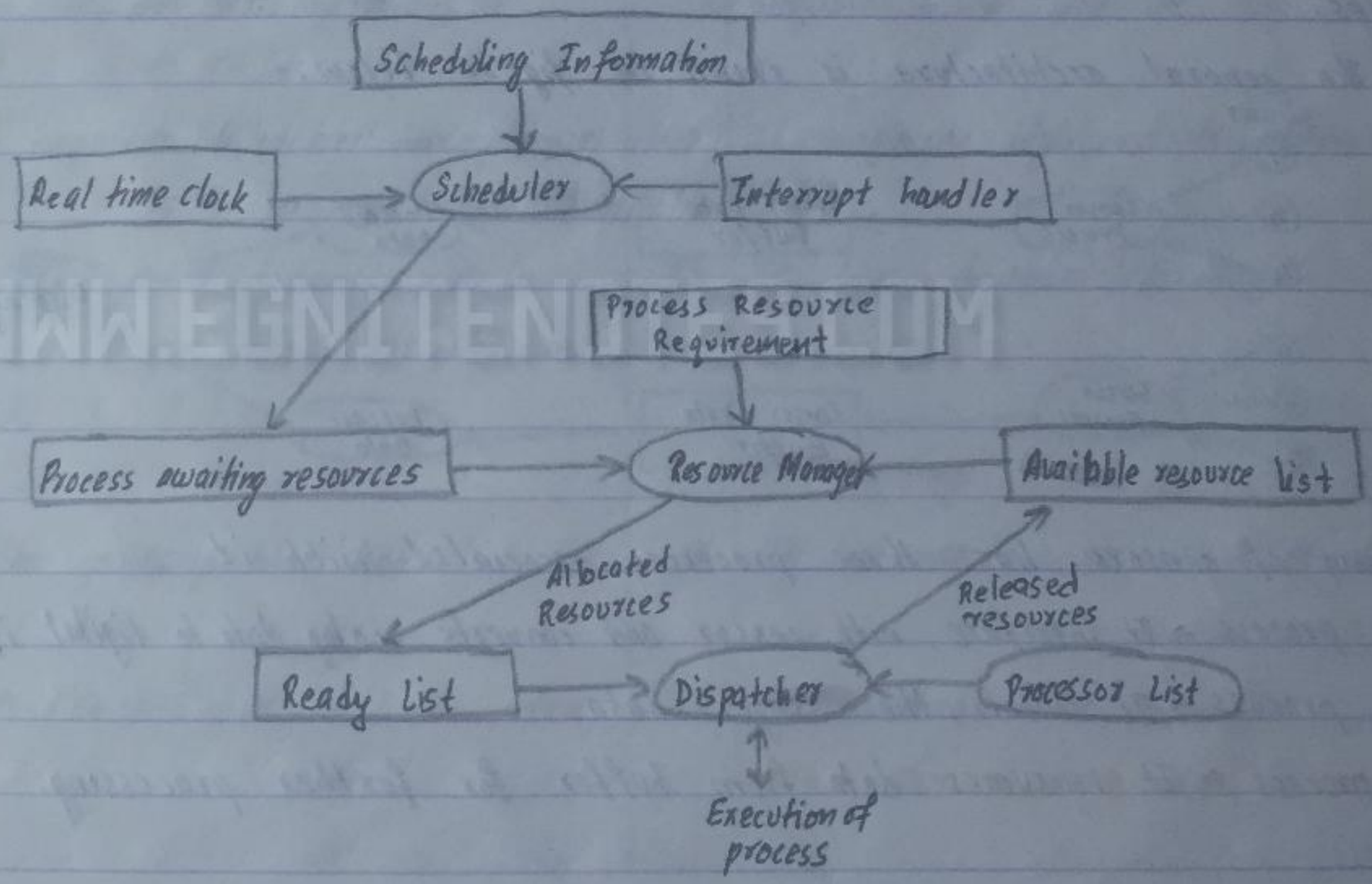
d) Resource Manager → It allocates memory and resources for scheduled process.

e) Dispatcher → It starts the execution of a process.

2) System Components

a) Responsible Manager → Responsible for dynamic re-configuration of real time system.

b) Fault Manager → Responsible to detect faults and take appropriate actions.



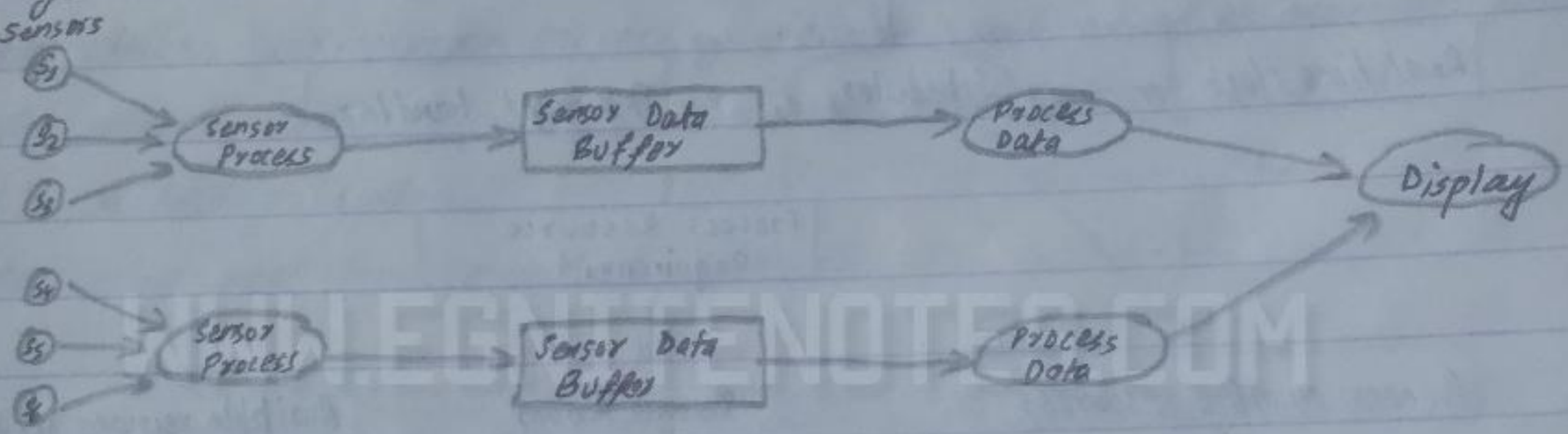
4.4 Monitoring and Control Systems

→ Important aspect of RTS.
→ They check sensors continuously and take actions depending on sensor value.
→ Monitoring system examine sensors and report their results.
→ Control system takes sensor values and control hardware actuators.

4.5 Data Acquisition System

Data acquisition system collects data from sensors for subsequent processing and analysis. This system generally comes into action when the sensor is collecting lots of data and its impossible to process them in real time. The key aspect is that the sensor reading should be collected by the system before its value changes.

The general architecture is shown in figure below:-



Each group of sensors has three processes associated with it.
a) Sensor process → It interface with sensor and converts analog data to digital if necessary.
b) Buffer process → It collects the sensor data.
c) Data process → It consumes data from buffer for further processing.

The specific considerations to be made are:-
a) Data acquisition may be faster than data processing. To smooth out such speed difference, circular ring buffer should be used.
b) Mutual exclusion must be implemented i.e. acquisition process and processing process should be prevented from accessing same element in buffer at the same time.
c) It must ensure that acquisition process does not try to add element in full buffer and processing process does not try to extract element from empty buffer.

architecture.
to structure and organize the system.
ftware bus.

03
3(e)

4.6 Differentiate RTOS and non-RTOS

	RTOS		non-RTOS (GPOS)
a)	RTOS is time critical.	a)	GPOS is not time critical.
b)	Generally, it is single purpose.	b)	It is multiple purpose system.
c)	Scheduling is also priority based.	c)	Scheduling is based on obtaining high throughput.
d)	It is designed for low end, stand alone device like ATM.	d)	It is designed for high end, general purpose device like PC.
e)	It requires low cost to implement.	e)	It requires high cost to implement.
f)	It has bounded latency i.e. a process will execute in specified time limit.	f)	It has unbounded latency
g)	It has preemptible kernel.	g)	It has non-preemptible kernel
h)	Eg: ModComp Classic, RSX-11, etc.	h)	Eg: Windows, Linux, etc.

that allows you to structure and organize the system.
te through a software bus.

Chapter - 5
Software Reuse

5.1 The reuse Landscape
5.2 Design patterns
5.3 Generator based reuse
5.4 Application frameworks
5.5 Application system reuse
[5-marks]

Questions

1) Write short notes on:
a) Generator based reuse [3]
2) Prepare brief notes on design pattern with statement of their benefits. [5]
3) Compare reuse framework and pattern generator. [4]
4) Compare application framework and component reuse. [4]
5) What are benefits and problems of software reuse? What factors need to be taken care of for software reuse planning? [5]

5.1 Software Reuse

→ Reuse based software engineering is a strategy of using already existed components, modules or softwares in the new developing software in other to gear up development process and increase software reliability.
→ In application system reuse, the whole system may be reused into other systems.
→ In component reuse, components of an application are reused.
→ In object and function reuse, a single function or object class is reused.

5.1.1 Benefits

1) Reused software which are already tested are more dependable as their faults are already found and fixed.
2) It reduces margin of error in project cost estimation.

3) It helps in faster development of software with less error.
4) The overall development cost is reduced.

5.1.2 Problems

1) The reused elements may be incompatible with system changes if source code are not available resulting increase in maintenance cost.
2) It may be difficult to integrate reused elements in the system.
3) It may be tedious to generate some necessary modifications.
4) It is a tedious task to find the reusable components and to determine whether it is suitable for current project or not.

5.1.3 Factors to consider while planning reuse

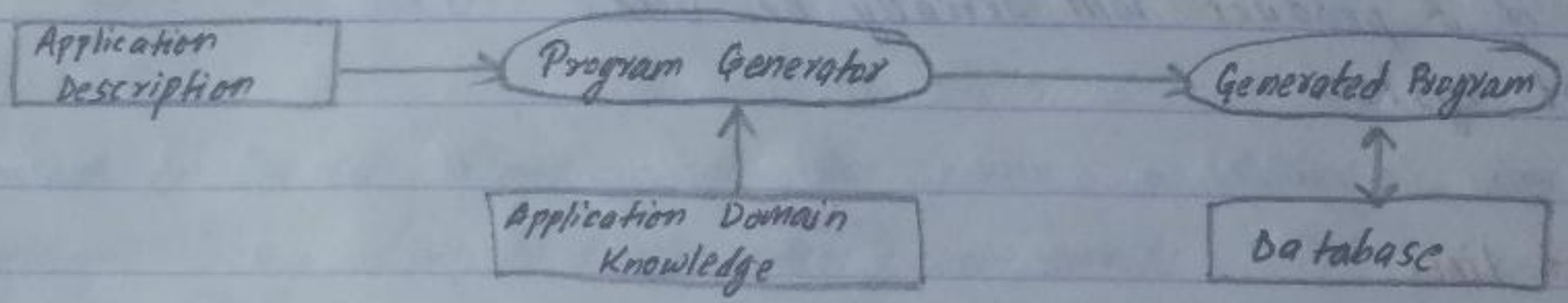
1) The development schedule for the software.
2) Expected lifetime of software
3) Background, skills and experience of development team.
4) Criticality of software and its non-functional requirements.
5) Application domain and platform

5.2 Design Patterns

→ Design patterns is a way of reusing abstract knowledge about a problem and its solution.
→ A pattern is a description of the problem and the essence of its solution.
→ It should be abstract so that it can reused in different setting.
→ The pattern must have following elements :-
  1) Name of pattern
  2) Problem description
  3) Solution
  4) Consequences
→ Design patterns implies concept reuse to specific problem.

5.3 Generator based reuse

→ In generator based reuse, the reusable knowledge is captured in a program generator system that can be programmed by domain experts.
→ It takes advantage of the fact that applications in same domain have common architecture and carry out comparable functions.
→ It makes reuse of standard programs and algorithms which are embedded in the generator and parameterised by user command to generate a program automatically.
→ Eg: Application generator for business data processing, parser generator for language processing, code generator in CASE tools, etc.
→ It is cost effective.
→ It's applicability is limited to small number of domains.



5.4 Application Framework

→ A framework is a subsystem design made up of a collection of abstract and concrete classes and the interface between them.
→ Frameworks are the large entities that can be reused.
→ In framework, subsystem are implemented by adding components and by providing implementations of abstract classes.
→ System infrastructure framework
→ Middleware integration framework
→ Enterprise application framework
→ Call backs are methods that are called in response to events recognized by the framework.
→ The problem is inherent complexity and time it takes to learn to use them.

5.5 Application System Reuse

→ It involves reusing entire application system either by configuring a system for specific environment or by integrating two or more systems to create a new application.

5.5.1 COTS (Commercial Off the shelf) product reuse

→ Software system that can be used without change by the buyer.
→ Large systems can be created by integrating a range of COTS system.
→ You have to make design choices like:
1) Which COTS product is most suitable?
2) How will data be exchanged?
3) What features of a product will actually be used?
→ Eg: E-procurement system

5.5.2 Software Product Lines

→ A product line is a set of applications with common application specific architecture.
→ New development involve reuse of core of application family with component configuration, adaptation to meet new demands, etc.
→ Eg: Enterprise Resource Planning (ERP) system

these services.
flexible and scalable architecture.
that allows you to structure and organize the system.
te through a software bus.

Chapter - 6

Component Based Software Engineering

6.1 Component and Component Model

6.2 CBSE Process

6.3 Component Composition

[3 - marks]

Questions

1) What are benefits of CBSE? How closely code generation feature of case tools are associated with CBSE? [3+6]

2) Write short notes on: CBSE [4]

6.1 CBSE

→ Component based s/w engineering is the process of defining, implementing and integrating loosely coupled independent components into systems.

→ It can produce complex and better software in less time.

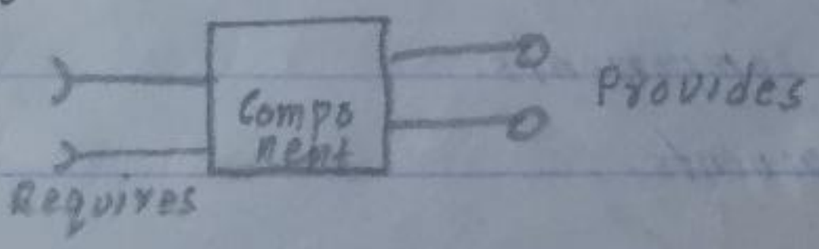
6.2 Components

→ Component is an independent s/w unit that can be integrated with other components to create a software system.

→ Components are defined by their interfaces.

→ A provides interface defines the service provided by the component.

→ A requires interface defines what services must be provided by other components to operate it correctly.



→ The characteristics of a component are :-

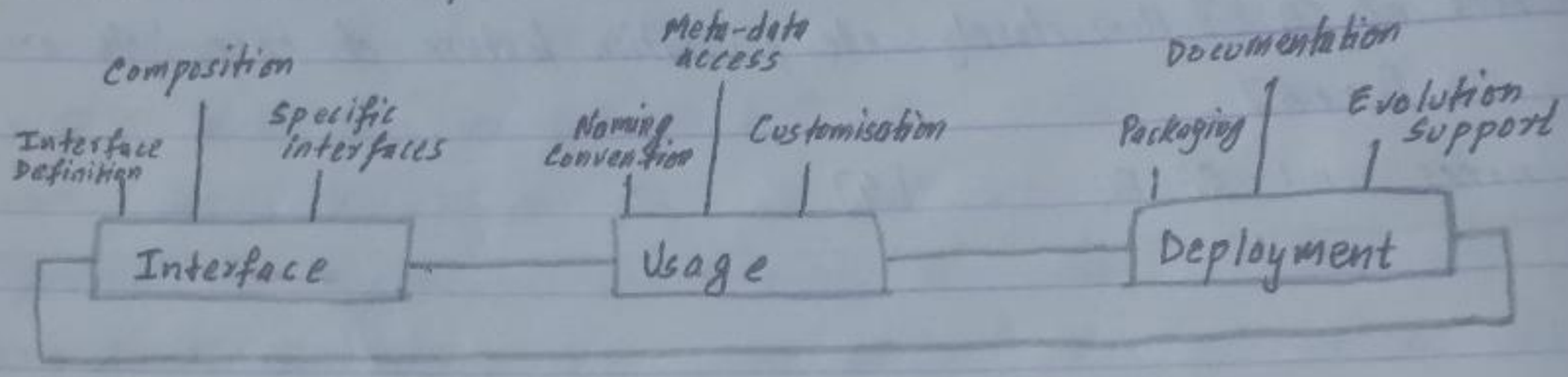
a) It should be independent entity.

b) It should follow standard component model.

c) All external interactions must use public interface.
d) It must be able to operate in stand-alone case.
e) It should be fully documented.
→ Component implementation are hidden

6.3 Component Model

→ Component model is a set of standards for components implementation, interface and deployment.
→ The basic elements of a component model is shown in figure below:-



→ Component model provides platform service (to communicate with each other) and support service.

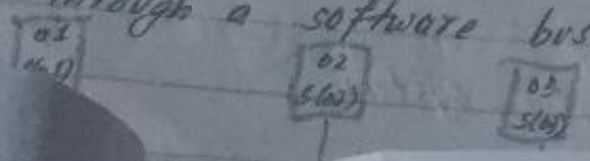
6.4 CBSE Process

→ CBSE emerged from failure of objects oriented development to support effective reuse.
→ CBSE requires knowledge, careful planning and methodological support.

→ CBSE process involves:
1) The user requirements are developed in outline.
2) Requirements are refined and modified based on components available.
3) Components are searched and requirements are modified.
4) Further searching for better components.
5) Integration of discovered components.

→ 2 types of CBSE process:
a) Development for reuse
b) Development with reuse

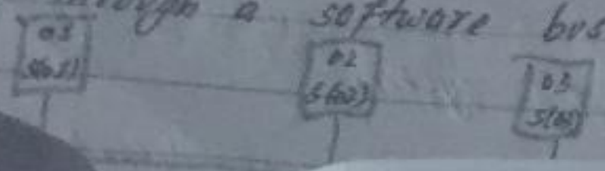
flexible and scalable architecture.
that allows you to structure and organize the system through a software bus.



6.5 Component Composition

→ Component composition is the process of integrating components to create a system or another component.
→ It is of following types:-
a) Sequential → The constituent components are executed in sequence.
b) Hierarchial → One component calls directly on services provided by another component.
c) Additive → Two or more components are put together to create new component.
→ In all cases, "glue code" should be written to link the components.
→ The linking component should have interfaces so that the components are compatible.
→ Incompatibility in interface can occur. It can be categorized as:
a) Parameter incompatibility
b) Operation incompatibility
c) Operation incompleteness
→ Such incompatibility can be tackled by writing adaptor component that reconciles the interfaces of two components being reused.

that allows you to structure and organize the system through a software bus.



Chapter - 7
Verification and Validation

7.1 Planning verification and validation
7.2 Software inspections
7.3 Verification and formal methods
7.4 Critical system verification and validation

[10 - marks]

Questions

1) Compare and contrast verification and validation. Define critical system. How does partitioning augments in V and V model? Explain with example. [4+2+2+2]
2) What is verification planning? Why is it required? Explain steps involved in it. [8]
3) Why software verification is needed before launching any system? [2]
4) Why program inspection are effective for discovering errors? What types of errors are unlikely to be discovered? [5+5]

7.1 Verification and Validation

→ Verification involves checking that the software meets to its specification i.e. functional and non-functional requirements.
→ Validation ensures that software system meets customer's expectations i.e. what customer expects it to do.
→ The major goals of verification and validation are:-
1) The extent to which the product is error free.
2) The extent to which the product is consistent.
3) The extent to which everything in the product is necessary.
4) The extent to which product is complete.
5) The extent to which the product performs.

7.1.1 Comparision of verification and validation

	Verification		Validation
1)	Verification is static practice of checking documents, design, code and programs.	1)	Validation is dynamic mechanism of validating and testing the actual product.
2)	It does not involve code execution.	2)	It involves code execution.
3)	It is human based checking of documents and files.	3)	It is computer based execution of program.
4)	It uses walkthroughs, inspections and reviews.	4)	It uses black box testing, gray box testing and white box testing.
5)	It is to check whether software meets specification.	5)	It is to check whether software meets customer's expectations.
6)	It is low level practice.	6)	It is high level practice.
7)	Target is requirement specification, software architecture, etc.	7)	Target is actual product i.e. unit, modules or final product.
8)	It is done by BA team.	8)	It is done by testing team.
9)	It is done before validation.	9)	It is done after verification.
10)	It describes whether o/p are according to i/p or not.	10)	It describes whether software is accepted by user or not.

ell these services.
lexible and scalable architecture.
hat allows you to structure and organize the system
hrough a software bus.

ntation

7.2 Planning V and V

→ Verification and validation planning means to establish a standards for the effective process of verifying and validating the software product.
→ Planning should be done in early phase of development process.
→ It depends on complexity and criticality of the software system.

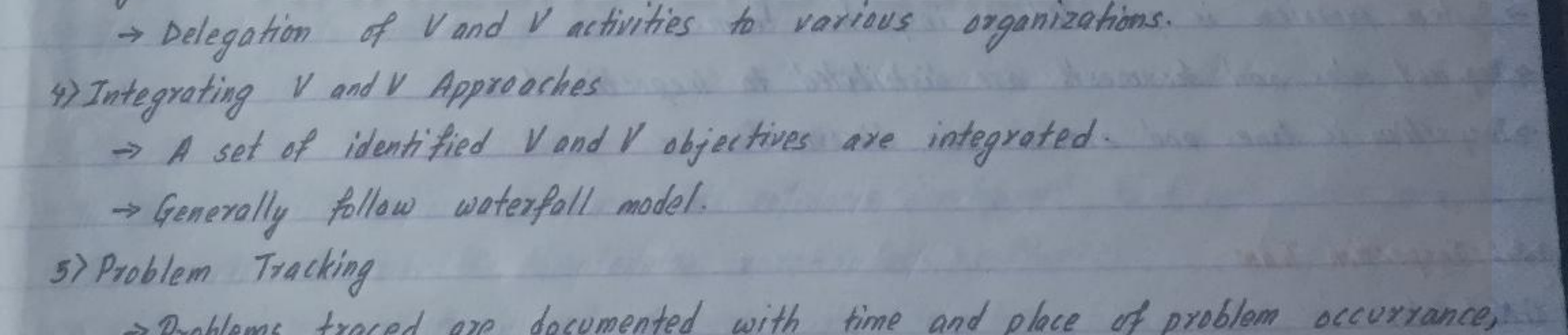
7.2.1 Necessity

1) It improves the software quality by detecting bugs.
2) It assures the software development is in right track.
3) It helps to reduce cost for verification and validation process.

7.2.2 Steps to be considered

1) Identification of V and V goals
   → The goals must be identified from requirements and specifications.
   → It must address attributes that corresponds to its user's expectation.
2) Selection of V and V techniques
   → The methods for the process should be selected for each of project's product.
3) Organizational Responsibility
   → Delegation of V and V activities to various organizations.
4) Integrating V and V Approaches
   → A set of identified V and V objectives are integrated.
   → Generally follow waterfall model.
5) Problem Tracking
   → Problems traced are documented with time and place of problem occurrance, problem evidence, solving priority, etc.

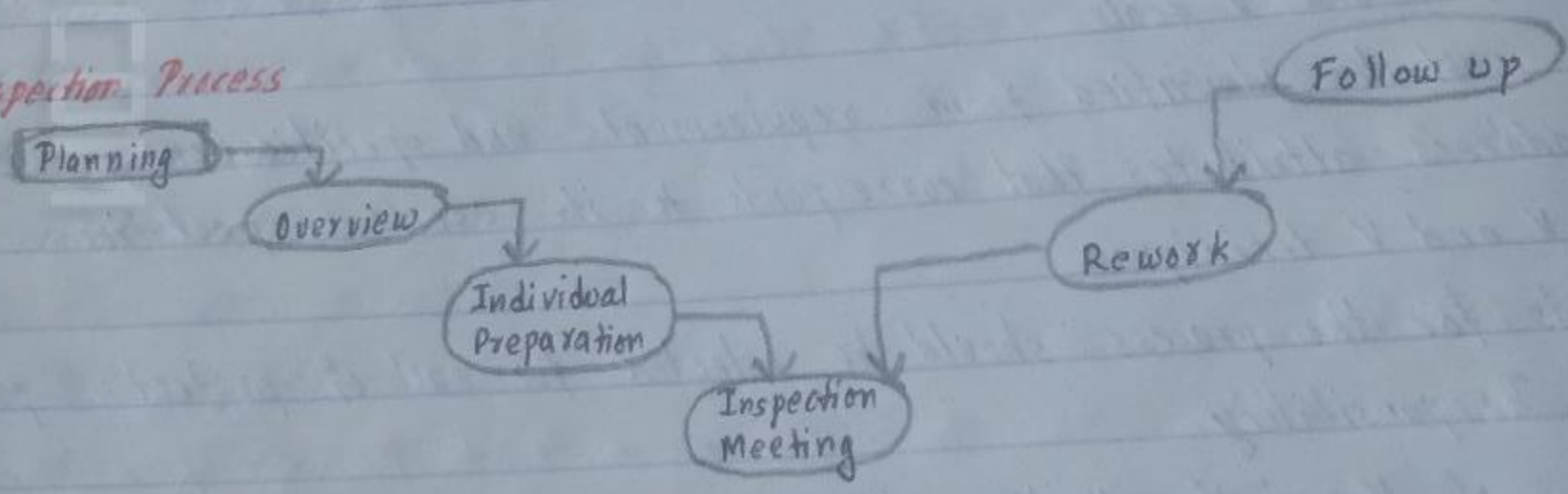
7.2.3 V model of development



7.3 Software Inspections

→ Software inspection is a process in which system is reviewed to detect errors before coding and in code before testing.
→ It is manual and static technique.
→ Inspection is effective to detect errors because of following reasons :-
1) Errors can mask other errors. Since inspection is static process, you do not need to concern with interactions between errors so that single inspection can detect many errors.
→ Advantages of inspections :-
1) Incomplete versions can be inspected without additional cost.
2) It considers inappropriate algorithms and poor programming styles.

7.3.1 Inspection Process



→ System overview is presented to inspection team.
→ Required codes and documents are distributed to inspection team.
→ Inspection is done and errors are discovered.

7.3.2 Inspection Team

1) Moderator
2) Inspector
3) Reader
4) Author

they provide.
these services.
ible and scalable architecture.
t allows you to structure and organise the systems
hrough a software bus.

7.3.3 Errors unlikely to be discovered

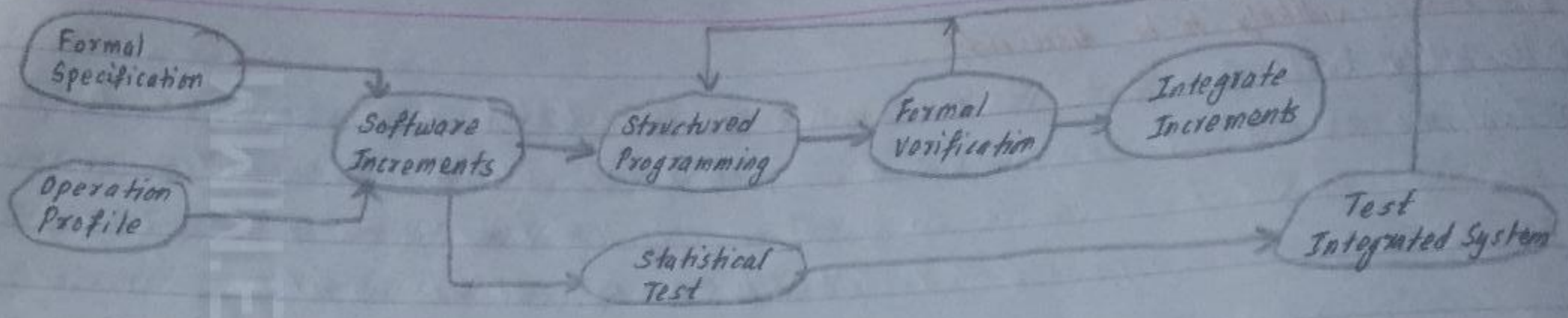
1) Usability Problems
2) Run time errors

7.4 Verification and Formal Methods

→ Formal methods are the analysis of the mathematical representation of the specification and software.
→ It is static verification technique.
→ It involves detailed mathematical analysis of specificatims, so the errors are more likely to be discovered.
→ It can detect implementation errors before testing.
→ It requires special notations which is difficult to understand.
→ It is very expensive process.

7.4.1 Cleanroom Software Development

→ It is a well documented approach of software development that uses formal methods.
→ The main objective is to develop zero-defect software.
→ It has replaced unit testing of components by inspection check.
→ It is based on five key strategies:
1) Formal specification
2) Incremental Development
3) Structured programming
4) Static verification
5) Statistical testing



7.5 Critical System

→ Critical system is a system whose failure or malfunction may result in hazardous impacts.

→ Critical system should be developed with less chance of fault.

→ It includes:

a) Reliability validation

b) Safety assurance

c) Safety assessment

d) Dependable cases

Chapter- 8

Software Testing and Cost Estimation

8.1 System Testing
8.2 Component Testing
8.3 Test case design
8.4 Test automation
8.5 Metrics for testing
8.6 Software Productivity
8.7 Estimation Technique
8.8 Algorithm. Cost Modelling
8.9 Project Duration and Staffing

[8-marks]

Questions

1) What is COCOMO? Calculate COCOMO effort, development time in month, average staffing and productivity that is estimated to be 49,200 lines of code. [3+5]

2) Establish chronology among component, release unit and integration testing. Write distinctive notes on their testing. [3+4]

3) Differentiate functional and structural testing. For 200 KLOC, calculate effort per month, development time, average staffing and productivity rate considering COCOMO-2 for reference. [5+3]

4) What is exception and error testing? [5]

5) Explain software testing metrics. [4]

6) Explain black box testing. [5]

7) Why unit test is not enough in system with multiple interacting units? What other tests are required? [8]

8) Differentiate black box and white box testing. [5]

9) Differentiate alpha and beta testing. Also write their importances. [8]

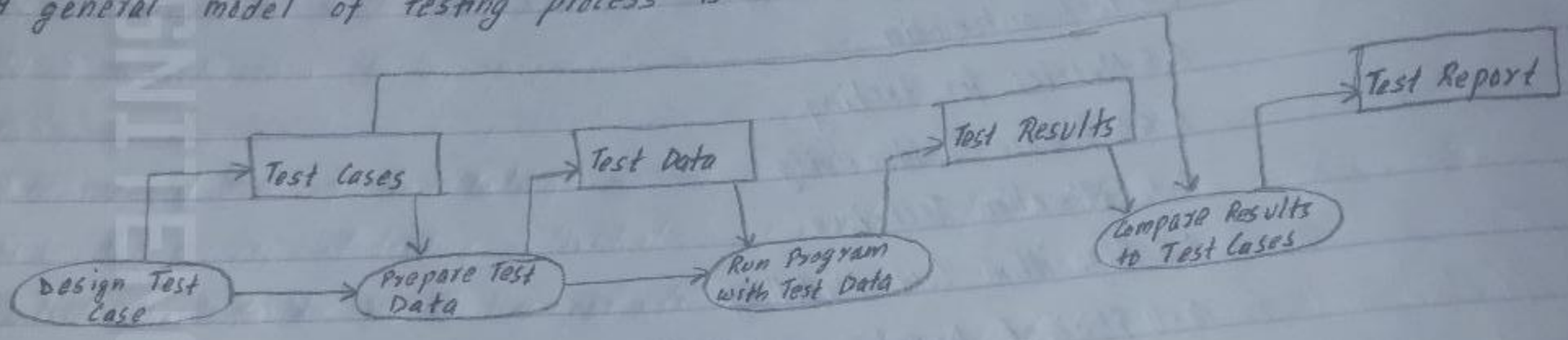
10) What problems may encountered when choosing top-down integration? What is regression testing? [10]

11) What are basic principles of testing? List characteristics of testability. List errors of black box testing. [10]

12) Why software fail after passing through acceptance test? [3]

8.1 Software Testing

→ Software testing is the process of exercising with the intent of finding errors prior to the delivery of product to the end user.
→ It provides high quality software, low maintenance cost and accurate results.
→ A general model of testing process is shown below:-



8.2 System Testing

→ It verifies whether the elements are properly integrated and perform specified function.
→ It is carried out by testing team.

8.2.1 Integration Testing

→ It involves combining the modules of a software to check whether the combination is working as per specification.
→ It verify functional, performance and reliability requirements.
→ Interprocess communication and shared data usages are tested.
→ It is performed after unit testing.
→ It reveals the errors occured due to integration of modules.

8.2.1.1 Big Bang Approach

→ All or most of the modules are coupled together and then integration testing is carried.

8.2.1.2 Bottom-up Testing

→ The lowest level components are tested first and are used to test higher level components. This process is repeated until top most component is tested.

these services.
ble and scalable architecture.
allows you to structure and organize the system
rough a software bus.

→ Helpful when all the modules are ready.
→ Bugs are more easy to find.
→ More efforts are needed.
→ Error in interface is found lately.

8.2.1.3 Top-Down Testing

→ The top level modules are tested first and the branches are tested step by step until the end of module.
→ Benefits:
1) We can test critical functions early.
2) We can test the interfaces easily.
3) It is easy to find missing branch.
→ Drawbacks:
1) The solution provides limited coverage in early stage.
2) The custom adapter should be developed in early stage.
3) Implementation cost is high.
4) Certain modules should be re-tested for integrity.

8.2.1.4 Regression Testing

→ When a new increment is integrated to previous one, the tests for previous increments must be rerun and the new tests should also be performed to ensure functionality of new system and to ensure that the previously fixed bug do not affect the other sections of modules. Such test is regression testing.
→ It determines whether change in one part affects other part of the software.
→ It tracks the correctness of program and quality of output.
→ It is performed by the testing team.

8.2.2 Release ~~[illegible]~~ / Functional Testing

→ A complete version of the system is tested before releasing to users.
→ It focuses on discovering bugs and to check whether the system meets the requirements of customer's.
→ It is concerned with functionality of software.
→ It is a black box testing process.

8.3 Component Testing / Unit Testing

→ It is the process of testing individual components of a software system to detect the faults in the components.
→ Developers of components are responsible for this test.
→ The components to be tested may be individual function or object class or composite components.
→ It focuses on testing component interface.

Black box and white box testing (structural)

	Black box testing		White box testing
1)	It focuses on functionality of system.	1)	It focuses on structure of system.
2)	Knowledge of programming is not necessary.	2)	Knowledge of programming is necessary.
3)	Knowledge of implementation is not required.	3)	Knowledge of implementation is required.
4)	Done by independent testing team.	4)	Done by software developer.
5)	It focuses on what is carried out by the system.	5)	It focuses on how it is carried out by the system.
6)	Code of software is not important.	6)	Code of software is important.
7)	It is external test.	7)	It is internal test.

services.
d scalable architecture.
s you to structure and organise the system.
a software bus.

Alpha test and Beta Test

	Alpha Test		Beta Test/Field Test
1)	Performed by developers.	1)	Performed by customers.
2)	It is not public.	2)	It is public.
3)	It is conducted for software application.	3)	It is conducted for product.
4)	Performed in virtual environment	4)	Performed in real environment.
5)	It involves both white box and black box testing.	6)	It involves black box testing only.
6)	Performed at the time of acceptance test.	6)	Performed at time of product marketing.
7)	Done at developer's area.	7)	Done at customer's area.

Failure of project after acceptance test

1) Poor user input
2) Vague requirements
3) Late failure warning
4) Communication breakdown
5) Hidden costs
6) Customer's conflict

Errors of black box testing

1) Small number of inputs can only be tested.
2) Test case is difficult to design.

Principle of testing

1) Testing an application completely is impossible.
2) Testing is context based.
3) Testing is to find out bugs.
4) Testing starts from requirement gathering
5) The defects seem to come from one or few areas of application.
6) Performing similar tests multiple time do not identify defects.
7) Absence of error does not mean it is free from defects.

Characteristics of testability of software

1) Operability
2) Observability
3) Controllability
4) Decomposability
5) Simplicity
6) Stability
7) Understandability.

* Cyclomatic complexity

8.4 Test Case Design

→ It involves designing of the test cases (inputs and outputs) to create a set of tests effective for validation and defect testing.

→ Steps involved in test case design are:-

1) Select a feature of the system that you are testing.
2) Select a set of inputs that execute that feature and document the expected outputs.

8) Consider program to determine nature of roots of a quadratic equation. Its input is a triple of positive integers (say a, b, c) and values may be from interval [0, 100]. The program o/p may have one of following words (not a quadratic equation, real root, imaginary root, equal roots). Design test cases to test this program.

→ The possible partitions can be:

1) No. of inputs

Less than 3	Equal to 3	Greater than 3

2) Value of each inputs

Negative numbers	Number bet^n 0 and 100	Number greater than 100.

Test cases are:-

Input (a,b,c)	Expected output
2, 4, 6	Imaginary root
1, 4, 4	Equal root
1, 5, 6	Real root
10, 19, 28, 57	Not a quadratic equation
59, 6	Not a quadratic equation
-5, 2, 3	Error
2, -9, 105	Error
5, 99, 175	Error

8.5 Test Automation

→ It is use of special software to control execution of tests and comparison of actual outcomes to predicted outcomes.
→ Reduces testing costs.
→ Two general approach are:-
  a) Code driven testing
  b) Graphical UI testing

8.6 Metrics for testing

→ Metric is a quantitative measure of degree to which system or component processes a given attribute.
→ It helps to estimate the progress, quality and health of software testing effort.
→ Testing metrix can be process metric, product metric or project metric.

→ Eg: Halstead metric to calculate testing effort.

$$PL = 1 / [(n1/2) * (N2/n2)]$$
$$e = V/PL$$

where, PL = program level
n1 = no. of distinct operations
n2 = no. of distinct operands
N2 = total no. of operands
V = program volume
e = effort

The % of overall testing effort to be allocated to a module is estimated as:

$$K = e(K) / \Sigma e(i)$$

8.7 Software Productivity

→ Software productivity is the ratio of amount of software produced to the number of labor and expenses of producing it.

→ It can be approached using:

a) Functionality metric (no. of function point per person-month)

b) Size metric (no. of LOC per person-month)

→ Functionality metric is independent of programming language used.

→ Size metric depends on programming language used.

8.8 Estimation Technique

1) Algorithm cost modelling

2) Expert judgement

3) Estimation by analogy

4) Parkinson's law

5) Pricing to win

8.9 Algorithm Cost Modelling

→ It uses mathematical formula to predict project costs based on estimates of project size, no. of engineers and product factors.

→ In general, algorithmic cost estimate can be expressed as:

$$effort = A \times size^{B} \times M$$

where, A = constant based on organizational practice

size = code size

B = multiplier = 1 to 1.5

→ Range of estimates should be developed (i.e. worst, expected and best)

8.9.1 COCOMO (Constructive Cost Modelling)

→ COCOMO is empirical model that is delivered by collecting data from large number of software projects.
→ It is well documented model.
→ It has been widely used and evaluated.

8.10 Project Duration and Staffing

→ From COCOMO model:

$$TDEV \text{ (calender time estimate)} = 3 \times (PM)^{(0.33 + 0.2 * (B - 1.01))}$$

where, PM = effort compulation
B = exponent

→ From COCOMO II model

$$TDEV = 3 \times (PM)^{(0.33 + 0.2 * (B - 1.01))} \times SCED\% / 100$$

where SCEDY. is Y. increase or decrease in nominal schedule.

Notes

1) $Effort = a_1 \times (KLOC)^{a_2}$ PM
→ Organic : $2.4 \times (KLOC)^{1.05}$
→ Semi detached : $3.0 \times (KLOC)^{1.22}$
→ Embedded : $3.6 \times (KLOC)^{1.20}$

2) Development Time $= b_1 \times (Effort)^{b_2}$ months
where, $b_1 = 2.5$

$$b_2 = \begin{cases} 0.38 & \text{(Organic)} \\ 0.35 & \text{(Semi-detached)} \\ 0.32 & \text{(Embedded)} \end{cases}$$

3) Productivity = LOC / PM
= LOC / Effort

4) Staffing = Effort / Development time

Chapter - 9
Quality Management

9.1 Quality concepts
9.2 SQA
9.3 Software Review
9.4 Formal Technical Review
9.5 Formal approaches to SQA
9.6 Statistical SQA
9.7 Software reliability
9.8 Framework for software metrics
9.9 Matrices for analysis and design model
9.10 ISO standards
9.11 CMMI
9.12 SQA plan
9.13 Software certification
[10-marks]

Questions

1) How does SEI CMM ensure quality of complex software? Differentiate ISO and CMM. [4+3]
2) How many levels are there in CMM? Explain. [2+5]
3) Why software quality standards are necessary? [2]
4) Explain statistical SQA. [4]
5) Explain SQA. [5]
6) Explain why software reliability is important. [4]
7) Explain SQA plan. [3]
8) How do you conduct formal technical review? Explain Garvin's quality dimensions. [6+4]

9.1 Software Quality

→ A software is said to have quality if it meets all the requirements specified by the user in initial phase and is easy to use for the users.

→ The factors for software quality are:-

1) Portable
2) Usable
3) Reusable
4) Correct
5) Maintainable

9.2 Software Quality Assurance (SQA)

→ Software quality assurance a planned and systemic pattern of actions that are required to ensure high quality softwares

→ SQA team includes engineers, managers, etc who must focus the software from the customer's point of view.

→ The various SQA activities includes:-

1) Preparing an SQA plan
2) Each process description should be reviewed.
3) Review to identify deviations from the specified process and verify that the correction has been made.
4) Record non-compliance items in the software and report them.

→ The main aim of SQA is to remove problems related to the software quality.

9.3 Software Reviews

→ Software reviews act as a filter during software development process.

→ Reviews can be done at various points during development.

→ It helps to uncover errors occured in different phases and can be removed easily during development.

→ It can be done in various ways formally or informally.

→ Formal Technical Review (FTR) is the most effective way.

9.4 Formal Technical Review / Walkthrough

→ Formal technical review is a software quality control activity performed by the software engineers in which the previous development phases are reviewed in a form of formal meeting.

→ The main objectives of FTR are :-

1) To uncover errors.
2) To verify software under review meets its requirement.
3) To ensure software is represented as per the standards.
4) To make projects manageable.

→ A FTR is conducted as a properly planned and controlled review meeting.

9.4.1 Review Meeting

→ It should be planned, controlled and attended by necessary representatives.
→ Walkthroughs and inspections are conducted for each component or small group of components to uncover errors.
→ The focus is on a work product

Conducting FTR

1) The producer, who developed work product, informs the project leader that work product is completed and need reviews.
2) The project leader calls review leader and generates copies of product materials.
3) The product materials are distributed to all reviewer for advance preparation.
4) The review leader also reviews the product and prepare agenda for review meeting.
5) The review meeting is attended by review leader, reviewers and producer.
6) One of the reviewers act as recorder who records issues raised.
7) The meeting begins with introduction of agenda and product by producer.
8) Walkthrough of work product is carried out in which producer explains the material while reviewers raise issues on them.
9) The discovered problems are recorded.
10) At the end, reviewers decide whether to accept product, reject product or accept provisionally.
11) The meeting is ended.
12) The summary report is published.

9.5 Formal Approaches to SQA

9.6 Statistical SQA

→ It reflects quantitative aspect of SQA

→ It involves following steps:-

1) Collection of information about software defects.

2) An attempt is made to trace out cause of each defect.

3) The vital few causes are isolated. According to Pareto principle, "80% of the defects can be traced to 20% of all possible causes."

4) Attempt to correct the problems that cause the defects.

→ Eg: Consider that an organization collects information on defects over a period of one year. Those defects can be categorized into few causes such as erroneous specification, misinterpretation of customer communication, violation of standard, error in data representation, error in design logic, incomplete testing, etc.

A table is developed showing critacality of the causes and the most vital causes are traced out and they are solved.

Now the new table is poped and simultaneous steps are carried out.

9.7. Software Reliability

→ Reliability denotes trustworthiness or dependability of a software product

→ It is the probability of product working correctly over a given time.

→ The reliability metrics are:-

1) Rate of occurrence of failure

2) Mean time to failure

3) Mean time to repair

4) Mean time between failures

5) Probability of failure on demand

6) Availability.

8.8 Metric for analysis and design model

→ Metric is a quantitative measure of degree to which a system possesses a given attribute.

→ Activities of measurement process includes formulation, collection, analysis, interpretation and feedback.

→ For analysis:

1) Functionality delivered
2) System size
3) Specification quality

→ For design:

1) Architectural
2) Component level
3) Interface
4) Specialized object oriented

9.9 ISO 9 Standard

→ It acts as reference for contract between independent parties.

→ It is a set of guidelines that addresses operational and organizational aspects to ensure software quality

→ It is based on a premise that if a proper process is followed for production, then good quality products are bound to follow automatically.

→ Benefits to obtain ISO certification:

1) Confidence of customer in organization increases.
2) High quality software can be produced.
3) Makes the development process efficient.
4) Points out weakness of organization and recommeds solution.

9.10 SEI Capability maturity model

→ CMM is a reference model for inducting the software process maturity into different levels.

→ It is used to predict most likely outcome to be expected from next project that the organization overtakes.

→ CMM can be used in two ways:

1) Capability evaluation

→ It provides a way to assess the software process capability of an organization.

→ The results provide the indication of likely contractor performance.

→ It can be used to select a contractor for the next project.

2) Software process assessment

→ It is used to improve the software process capability.

→ It is for internal use only.

9.10.1 Maturity Levels of CMM

1) Initial

→ It is characterized by ad-hoc activities

→ Software process is not developed, so engineers follow their own process resulting in chaotic efforts.

→ Success of projects depends on individual efforts.

→ If an engineer leaves, it is difficult for others to follow the process.

→ It may leads to low quality.

2) Repeatable

→ Basic management practices like cost estimation, scheduling are established.

→ For this techniques like function point analysis, COCOMO, etc are used.

→ For similar applications, earlier successes on projects are repeated.

3) Defined
→ Management and development processes are defined and documented
→ Common understanding of activities within an organization is achieved.
→ The process and product qualities are not measured.

4) Managed
→ It focuses on software metrics.
→ The metrics measure the characteristics of the project like size, reliability, time complexity
→ The metrics provide effectiveness of process being used like average number of defects found, average no. of failures detected, etc.
→ Software quality is measured.
→ The results are used to evaluate performance but not to improve the process.

5) Optimizing
→ The process and product metrics are collected and analyzed for process improvement.
→ Improvement is achieved by analyzing feedback from measurements.
→ The best software engineering practices and innovations can be identified and may be used as tools for other projects.

→ CMM provides list of key areas on which to focus to take an organization from one maturity level to next providing gradual quality improvement.

9.10.2 ISO vs CMM
→ ISO is international standard so can be quoted in official document but CMM is purely for internal use.
→ CMM is developed for software industry only.
→ CMM helps to achive quality management while ISO aims at level 3 of CMM model.
→ CMM provides a way for achieving gradual quality improvement.

9.11 SQA Plan

→ It consists of procedure, technique and tools used to ensure that a product meets the requirement specifications.

→ Phases:

1) SQA Plan for software requirements
2) SQA Plan for architectural design
3) SQA Plan for detailed design
4) SQA Plan for transfer

Chapter- 10

Configuration Management

10.1 Configuration Management Planning
10.2 Change Management
10.3 Version and Release Management
10.4 System Building
10.5 CASE tools

[4-marks]

Questions

1) Write short notes on:
a) Change management [3] b) Release management [3] c) Version control [3]
2) "Survival of fittest" is valid to slw industry. Explain in context of issues modern slw configuration management must address. [8]
3) Explain versioning process with all the associated components. [5]
4) Describe types of software maintenance. [10]
5) Explain CASE, CASE environment and CASE tools. [11]

10.1 Configuration Management

→ CM is the process of managing changing software system.
→ It consists of policies, processes and tools for management.
→ Configuration management plan describes the standards and procedures that should be used for configuration management.
→ CM planning should include :-
1) Documents to be managed
2) Responsible personnel for CM procedure
3) Policies for change and version management.
4) Tools to be used.
5) Database to record configuration information.

10.2 Change Management

→ Changes to software system is required as requirements of the software may change over time.

→ To apply such changes in a controlled and efficient way, change management is necessary.

→ It involves analysis of cost and benefits of proposed changes, approval for worthy changes and tracking of changed components.

→ The steps involved are :-

1) A change request is made describing the required system changes.

2) The change request is analyzed regarding estimated cost, benefits and necessity.

3) If the change request is invalid, duplicated or already considered, it is rejected.

4) If the changes are valid, change assessment and cost estimation is done and submitted to change control team.

5) Change control team review and approve if necessary changes are proposed.

6) The change is made to software, quality is tested and finally new system version is delivered to the customer.

10.3 Version Management

→ Version management includes planned and proper procedures to keep tracks of all versions of the system so as to ensure that versions may be retrieved when required.

→ A version is a system instance that differs in functionality with other instances.

→ Any versions should be defined unambigiously so that it can be identified.

→ It includes various techniques to identify versions:

1) Version Numbering

→ Each version is given a unique version number with linear scheme. in general.

→ Hierarchial naming scheme leads to fewer errors in version identification.

2) Attribute based identification

→ Each version is identified with a unique set of attributes.

→ Attributes may be language, development status, platform, creation date, etc.

→ It reflects the relation with other versions in efficient way.

3) Change oriented identification
→ Used for system rather than components.
→ Each proposed change has a change set describing the changes ~~made~~ to be implemented
→ It allows system consistency rules to be specified.

10.4 Release Management

→ A system release is a version of the system which is distributed to the end-users.
→ Release management involves deciding when to release system version to users, managing release creation and distribution media.
→ A system release includes executable code, configuration files, data files, installation program and documentation.
→ New releases should not rely on previous one.
→ In some case, user may not want to use new release.

10.5 System Building

→ It is the process of creating a complete executable system by compiling and linking the software components that executes on a target configuration.
→ Version management and system building tools should be integrated.

Garvin's Quality Dimensions

1) Performance (Operating characteristics)
2) Features (Characteristics that enhance appeal of the product)
3) Reliability (Probability that product will not fail in specified time duration)
4) Conformance (Product meets specified standards)
5) Durability (Length of product's life)
6) Serviceability (Speed at which product can be put into service)
7) Aesthetic (Response of a user)
8) Perceived Quality (Quality to service based measure)

Symbolic Execution

→ It involves analysis of program to determine which inputs cause each part of a program to execute.
→ Software testing techniques that aids the generation of test datas.

Cyclomatic Complexity

→ It is a software metric that indicates the complexity of a program.
→ Measures number of linearly independent paths through a program source code.
→ It can be used in basic path testing.
→ Mathematically, it is defined with reference to control flow graph of a program.

i.e. $M = E - N + 2P$

where M = complexity
E = no. of edges on graph
N = no. of nodes on graph
P = no. of connected components

Equivalence Partitioning

→ It is a software testing technique that divides the input test data of the application under test into each partition of equivalent data.
→ It helps to derive test cases easily.
→ It minimizes the time required to perform software testing.
→ It is based on black box testing method (functional test)

Walkthroughs

→ Software peer review
→ Programmer leads through a software product and participants ask questions and make comments about possible errors and violation of standards.
→ Objectives:
1) To gain feedback about quality or content of the document
2) To familiarize the audience with the content.

Software Maintenance

→ Modification of a software product after delivery to correct faults or to improve the performance.
→ Most of the maintenance effort is used for non-corrective actions.
→ The purpose is to preserve the value of software product over time.

Types of Maintenance

1) Corrective → It includes modifications in order to correct problems discovered by users.
2) Adaptive → It includes modifications to keep software up to date according to changing world of technology.
3) Perfective → It includes modifications to keep software usable over long period of time.
4) Preventive → It includes modifications to prevent future problems of software.

Software Re-engineering

- It is the process of keeping or updating software as per the current market without change in its functionality.
- It involves changing of design of software and re-writing programs.
- It includes reverse engineering, program reconstruction and forward engineering.
- Eg: Initially Unix was developed in Assembly language. When C came into existence, Unix was re-engineered in C because working in C is easier than that in assembly language.