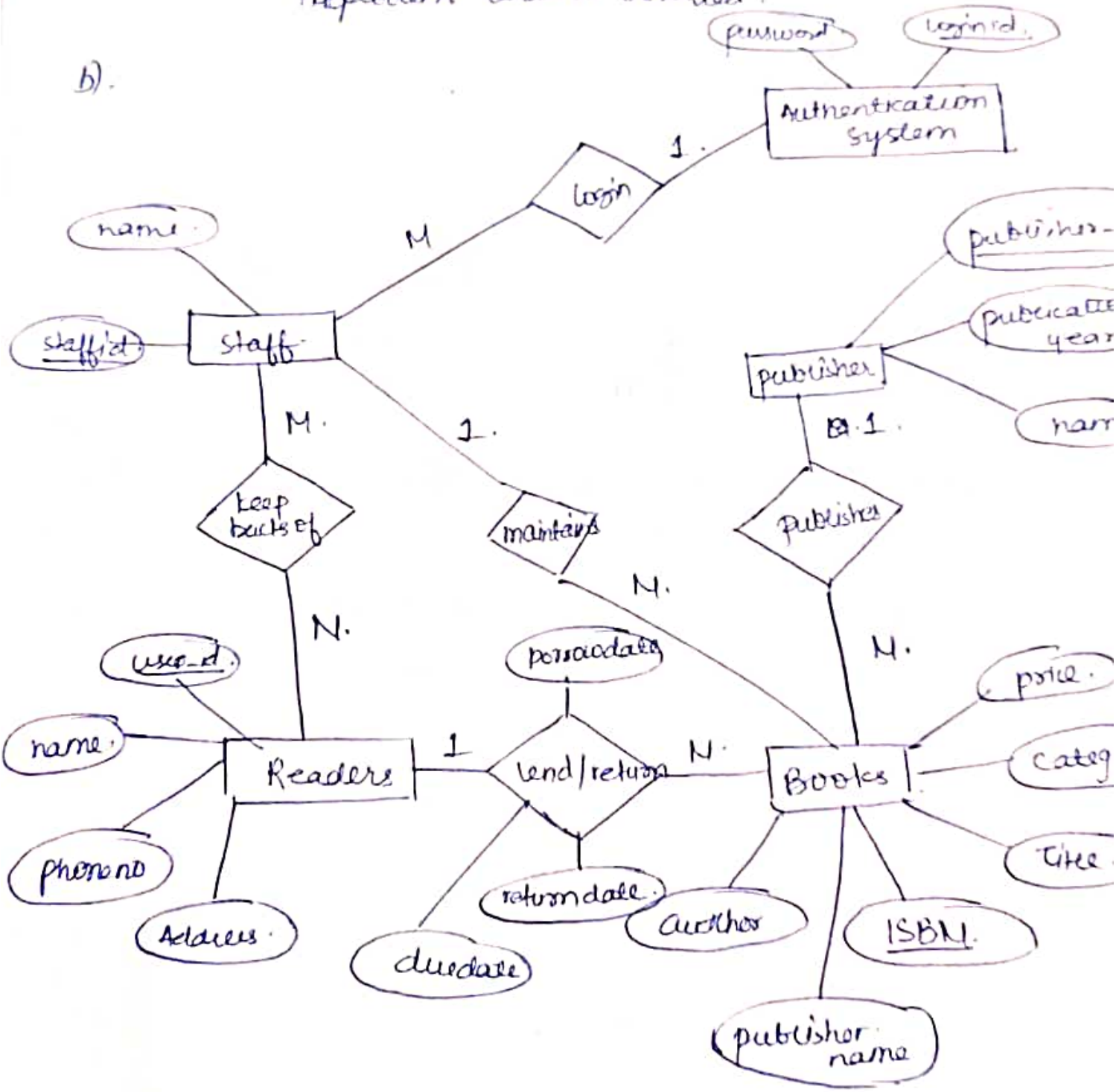
2016-17.

1.(a). Error → An error is defined as a mistake in the code that prevents it from being executed a compiled. A error is a mistake that causes a fault.

(b) Fault → Anabnormal condition that can cause an item to fail. A fault is the cause of a failure.

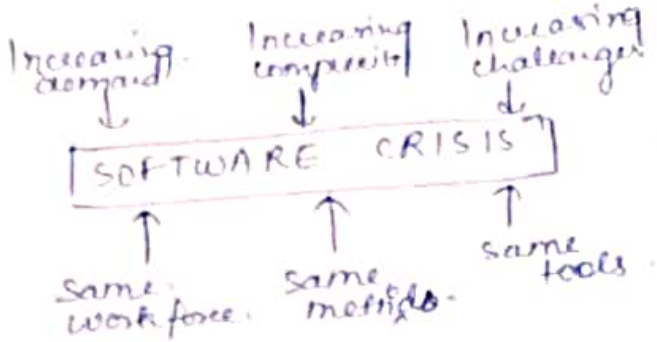
(c) Failure → A difference from expected result. A failure is the problem that is observed.

b).



Ques 2(a). Software Crisis →

Software crisis is a term used in computer science for the difficulty of writing useful and efficient computer programs in the required time with an increase in software complexity, many software problems arised because the existing methods were insufficient



Causes of software crisis.

→ The cost of owning and maintaining a software was as ex as developing a software.
→ At that time, projects were running overtime.
→ At that time, software was inefficient & often didn't mee requirements.
→ It was challenging to alter debug & enhance software.
→ Non-optimal resource utilization.

factors contributing to software crisis.

→ Poor project management.
→ Lack of adequate training in software engineering.
→ Less skilled project members.
→ Low productivity improvements.

Solution to software crisis –

There is no single solution to the crisis. One possible solution is software engineering because software engineering is a systematic, disciplined and quantifiable approach.

Goals of Software Engineering –

→ Reduction in software overbudget.
→ The quality of software must be high.
→ Less time needed to develop a software project.
→ Timely delivery of a software.
→ Software must meet user requirements.

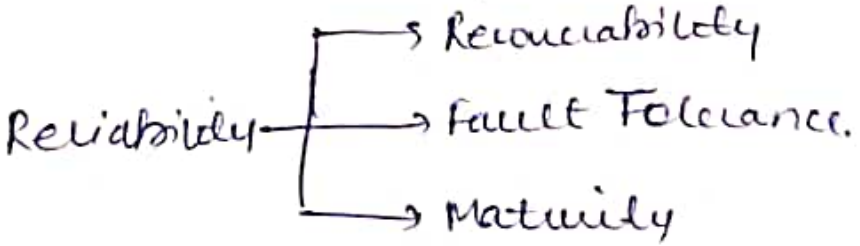
2(b). Important Attributes of Software Products

(1) Functionality →.

Functionality refers to the set of features and capabilities that a software program or system provides to its users. It determines the usefulness of the software for its intended purpose. The more functionality a software has, ~~Software programs are intangible programs that run on hardware to perform specific tasks. Hardware products~~ the more powerful and versatile it is, but also more complex it can be.

(2) Reliability →.

Reliability is a characteristic of software that refers to its ability to perform its intended functions correctly and consistently over time. It ensures that software will work correctly & will not fail unexpectedly.



(3) Efficiency →

It refers to the ability of the software to use resources such as memory, processing power, network bandwidth in an optimal way. High efficiency means the software program can perform its intended functions quickly & with minimal use of resources.

Efficiency → In time.
Efficiency → In Resources

(4) Usability

It refers to extent to which the software can be used with ease, the amount of effort or time required to learn how to use the software.

Usability → Understandability
Usability → Learnability
Usability → Operability

(5) Maintainability.

It refers to the ease with which modification can be made in a software system to extend its functionality, improve its performance etc.

Maintainability → Testability
Maintainability → Stability
Maintainability → Changability
Maintainability → Operability.

(6) Portability

A set of attributes that bear the ability of software to be transfused from one environment to another with minimum changes.

Portability → Adaptability
Portability → Installability
Portability → Replacability

Hardware attributes include physical characteristics such as size, weight, material, processing power, connectivity, and durability, constituting the tangible components that support software operations.

Ques 2(a). Prototyping model.

This model suggests building a working prototype of the system, before the development of the actual software. A prototype is a toy or crude implementation of a system.

The prototyping model is advantageous for specific situations -

- For the development of GUI part of an application, it becomes easier to illustrate the ~~int~~ input formats & interactive dialogs to the customer.
- It is useful when exact technical requirements are unclear to the development team.
- It can be deployed when the development of highly optimized & efficient software is required.

Weakness of the Prototyping Model.

The prototype mode incurs the additional cost of development of a prototype. It increases the cost of development in projects that are routine development work & do not suffer from significant risks.

Since prototype is constructed only at the beginning of the project, this model is ineffective for risks identified at later stages of development.

Iterative waterfall Model

The iterative waterfall method provides feedback paths from every phase to the preceding phase. This feedback allows for the correction of errors committed during some phase, as and when they are detected at a later stage.

There is no feedback to the feasibility stage, as once a team having accepted to take a project, doesn't give up the project easily due to legal or moral reasons.

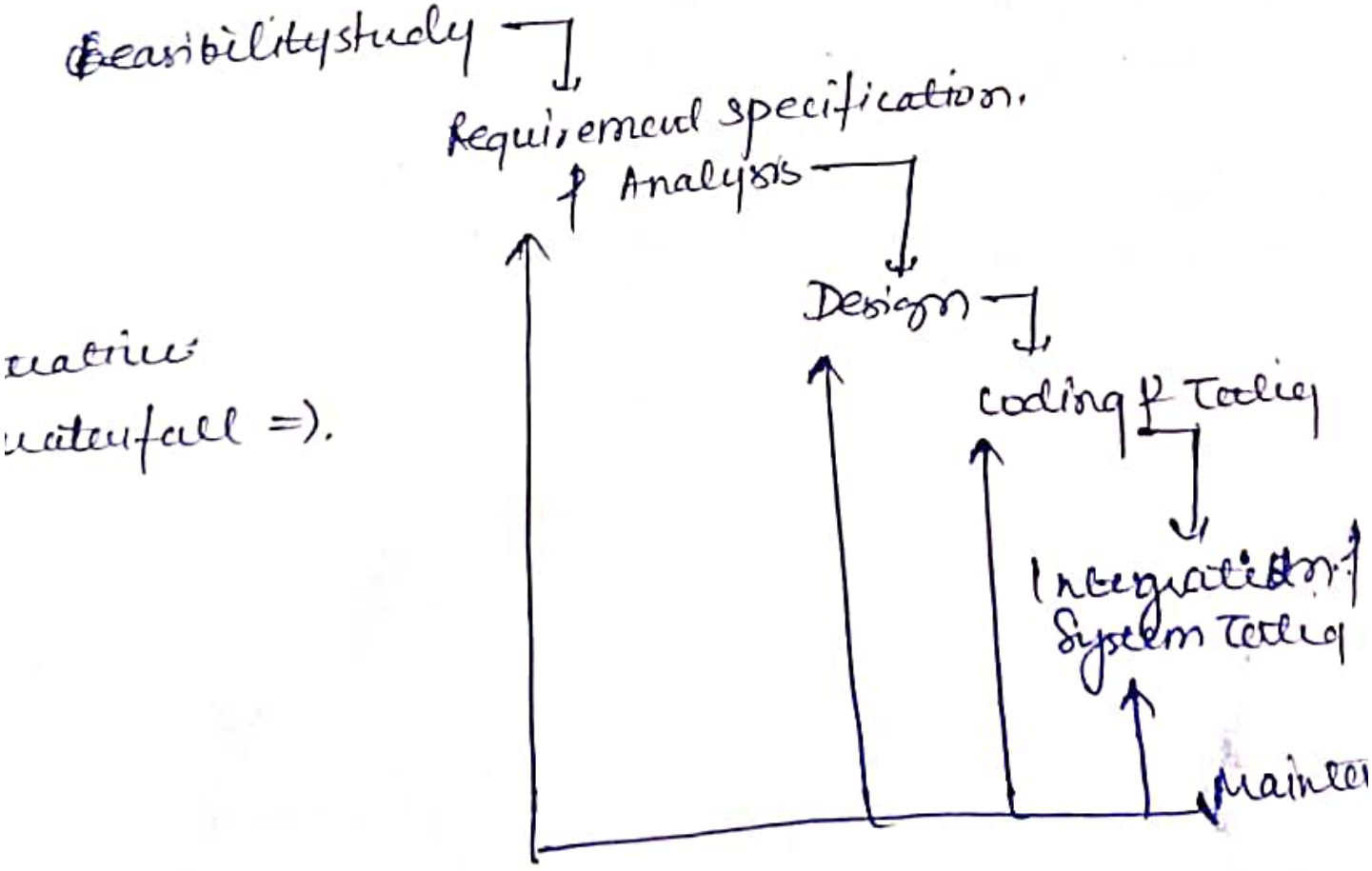
Shortcomings

→ Incremental delivery not supported.

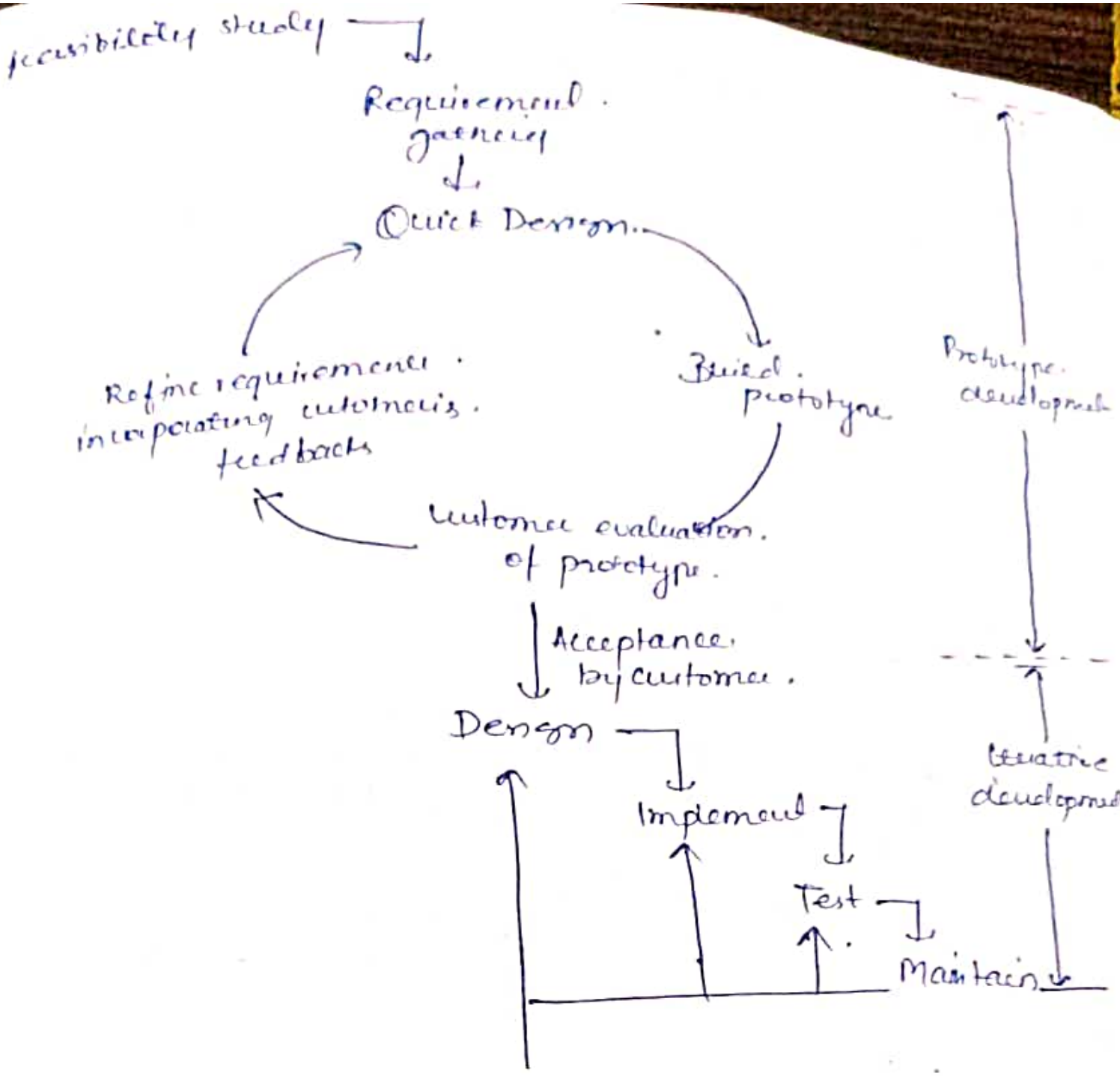
In this model, there is no provision for intermediate delivery to the customer. The full software is developed & tested before it is delivered to the customer which might take a long time & by that time the customer needs may have changed.

→ Limited customer interaction

This model supports very little customer interactions. Unlike the prototyping model. The developed software usually turns out to be a misfit to the customer's actual requirements.



Iterative waterfall =).


feasibility study
Requirement gathering
Quick Design
Build prototype
Customer evaluation of prototype
Refine requirements incorporating customers feedback
Acceptance by customer
Design
Implement
Test
Maintain
Prototype development
Iterative development

b). Important characteristics of an SRS. document

(1) Correctness →.

This review is used to ensure the correctness of requirements stated in the SRS. SRS is said to be correct if it covers all the requirements that are actually expected from the system

(2) Completeness →.

SRS. completeness is indicated by the resolution of as the to be determined parts to as much extent as possible as well as covering all functional & non-functional requirements proper

(3) Consistency →.

There should not be any conflicts between any set of requirements like differences in terminologies e.t.c..

(4) Unambiguousness →.

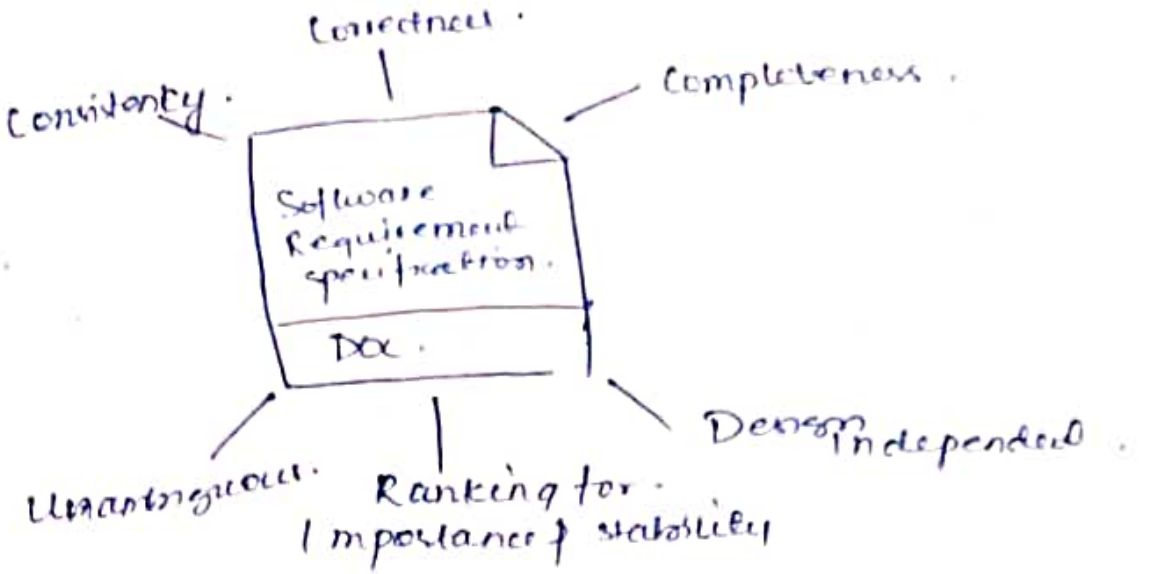
~~An SRS must have~~ All requirements stated in an SRS. must have only 1 interpretation. One way to ensure this is to include modelling techniques like ER diagrams e.t.c.

(5) Ranking for Importance & Stability →.

There should be a criteria to ~~specify~~ ~~the~~ classify the requirements as ~~important~~ less or more important, or ~~desirable~~ as desirable or essential.

(6) Design Independence →.

The SRS must not include any implementation details.



(Que4(a). Problem Analysis is the process of identifying, defining and understanding a problem. It involves decomposing a system into smaller parts to identify possible inputs, processes & outputs. The goal of problem analysis is to gain a better understanding of the problem before developing a solution.

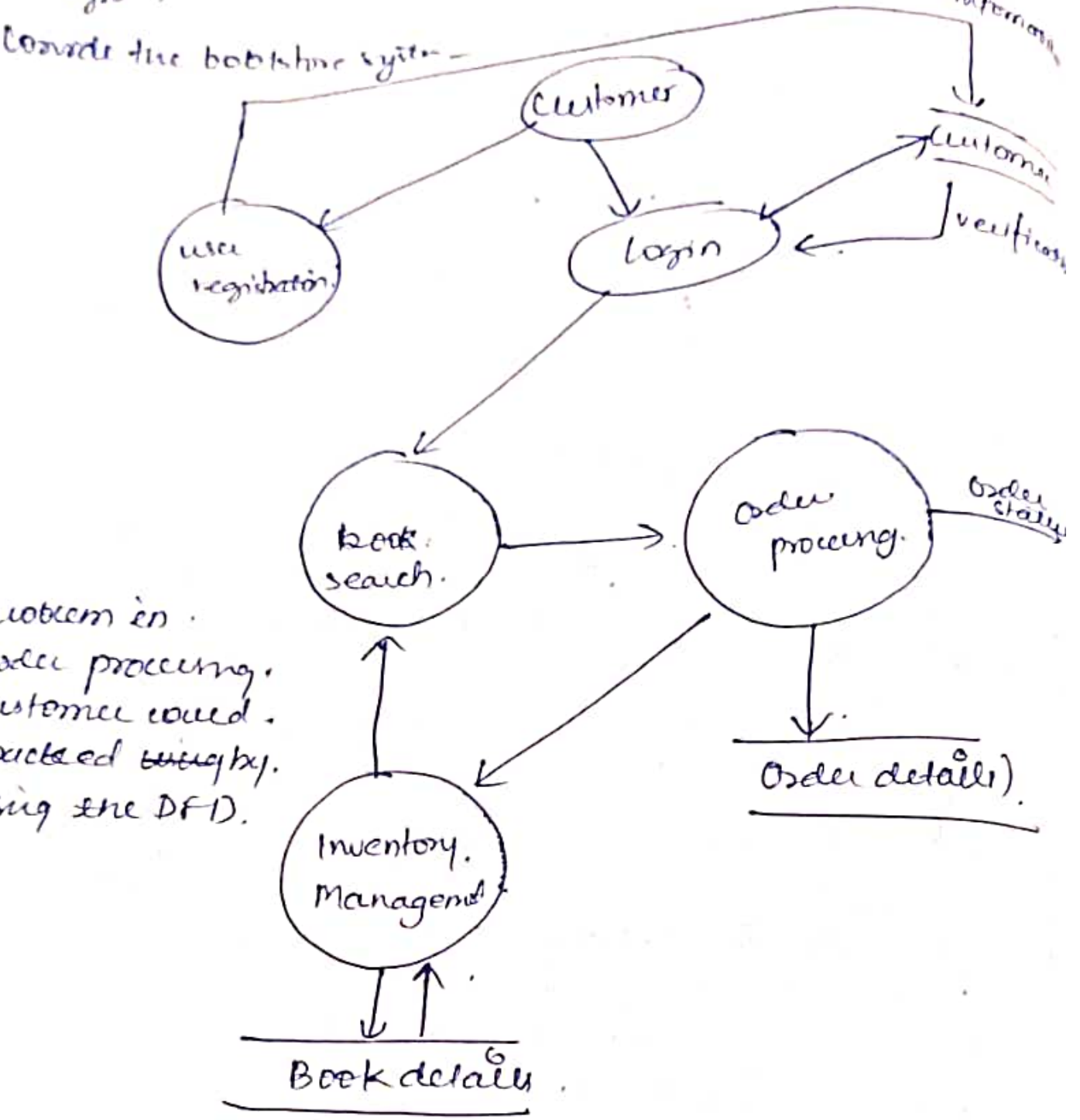
A problem can be defined as the difference b/w thing as perceived and thing as desired.

ADFD is a visual representation of how information flows through a system or a process. DFDs can help with problem analysis in software development by—

(1) understanding system operations.
(2) Identifying potential opeations.
(3) Improving efficiency.
(4) Developing better processes.
(5) Tracking data movement by external users.

DFDs serves as a powerful tools to depict the data flow with...
angle.

towards the bookstore syste...



Any problem in the order processing of a customer could be tracked ~~through~~ by analysing the DFD.

7). Use Cases.

A use case can be viewed as a series related together by common scenarios used together by common actors. The use case represents the different ways in which a system can be used by the user. A simple way to find use cases for a system is to ask the question - "what all can the different categories of users - use the system for?"

Then, for library management information system, the use cases could be -

- issue-book
- Return-book
- Query-book
- create-member etc.

During software development, use cases are used during various phases -

- Requirements Gathering - They help in understanding user needs & system functionalities
- Design phase → use cases Aid in designing system features and interfaces. They help developers understand how the users will interact with the system
- Testing → They serve as a basis for testing scenarios. Test cases are often derived from use cases to validate the system functionalities.

Throughout the software development, use cases help ensure the system aligns with user requirements, guiding the development process, design choices & testing procedures to create a product that meet user's need & expectation.

5(a) Module – A module is a self contained unit of software that encapsulates a set of related functions, procedures, or data structures. They are designed to perform specific tasks or functionalities within a larger software system. They help in organizing and managing data complex systems by breaking them into smaller, manageable components.

Modularization criteria used for function-oriented design to produce modular designs include –

(1) Functional Cohesion ⇒. Module should contain elements that are closely related and perform similar functions. It ensures that functions within a module are logically connected and contributed to a specific purpose or task.

(2) Low Coupling ⇒. Modules should have minimal interdependence or coupling with other modules. It ensures that change in one module have minimal impact on other.

High Encapsulation ⇒. Modules should hide their internal details and only expose necessary interfaces or functionalities. This allows changes to be made within the module without affecting other parts of the system.

Single Responsibility System ⇒ Each module should have a single, well-defined responsibility. It ensures modules are focused & easily understandable.

6(a)(i) A class diagram describes the static structure of a system. It shows how a system is structured rather than how it behaves.

They visualise class structure, relationships/ attributes and methods. They model classes, their associations, inheritance, aiding in system understanding, planning and implementation.

(ii) Interaction Diagram.

Interaction diagrams are models that describe how groups of object interact among themselves through message passing to realize some behaviour.

They illustrate system behaviour, reference cases, aid in design validation facilitating system understanding & refinement.

(iii) Activity Diagram.

They are used to visually represent the flow of activities or workflows within a system. They depict the action, decision points, and transitions that occur during the execution of a process or use case.

(10).

Ans. Test Case - A test case is a triplet [I, S, R] where.

I = Input data to the program

S = State of program at which data is to input.

R = Expected result to be produced by program.

Test criteria ⇒ Test criteria are a way to determine if a test has passed or failed. Test criteria can also help determine the boundaries of what's acceptable for an application.

eg. A test case is [Input: 'abc', State: edit, Result: abc] is displayed], which means the input abc needs to be applied in the edit mode, & expected result is that string abc would be displayed.

① Black-Box Testing

In black box-approach test cases are designed using only the functional specifications of software ie. test cases are designed solely based on an analysis of the I/O behaviour ie. functional behaviour.

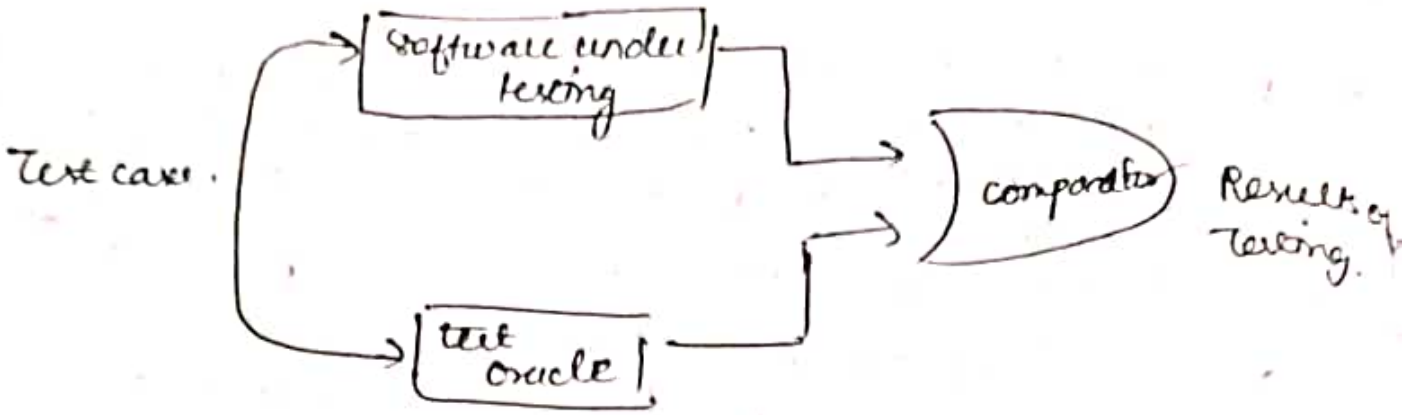
It doesn't require any knowledge of the internal structure of the program, for this reason it is also known as functional testing.

② White-Box Testing

Designing white-box test cases require a ~~thrott~~ thorough knowledge of the internal structure of a program, and therefore it is also called structural testing.

White-box test cases are designed based on the analysis of the code.

Q4. (a). Test Oracle is a mechanism, different from the program itself, that can be used to test the accuracy of the program's output of test cases.



Test Oracles are required for testing. Ideally, we want an automated oracle which will always give the correct answer. However, often oracles are human beings, As it is often difficult to determine ~~what the~~ whether the behaviour corresponds to expected behaviour, 'our 'human decision' may. make mistakes.

b) Software Maintenance.

It refers to the process of modifying and updating a software system after it has been delivered to the customer. This can include fixing bugs, adding new features, improving performance, or updating the software to work with new hardware or software systems.

The goal of software maintenance is to keep the software system working correctly, efficiently, and securely and to ensure that it continues to meet user requirements.

(iii) Data Dictionary -

A data dictionary lists the purpose of all data items and ~~and~~ the definitions of all composite data items in terms of their component data items.

e.g. a data dictionary entry may represent that the data group pay consists of the components regular pay + overtime pay.

grossPay = regular Pay + overtime Pay

A data dictionary plays a very important role for the following reasons—

- It provides a standard terminology for all relevant data for use by the developers working on project
- It helps the developers to determine the definitions of different data structure in terms of their component elements while implementing the design.
- It helps to perform ~~important~~ impact analysis i.e. it is possible to determine the effect of some data on various processing activities & vice versa.

(iv) The open-closed principle ~~that~~ in software engineering states that code should be open for ~~modification~~ extension but closed for modification. This means that new entities can be added without modifying the existing code.

It is ~~avoid~~ used to avoid situations where modification in one part of the codebase causes unintended effects on other parts of the code.

The goal is to be able to extend the system for cheap, so we won't have to make changes to our core/ modules. everytime we need to extend it. It approach helps to keep the complexity low.

2018-19.

a) A program refers to a set of instructions written in programming language to perform a specific task or a set of tasks on a computer. It's a smaller component of software that typically serves a single purpose.
e.g. a program might be designed to calculate mathematical equation or perform a specific function like word processing or image editing etc.
Programs can be standalone or part of a larger software system.

Software product, it's a complete package or a collection of programs, documentation, user interfaces, and possibly other components that are bundled together to provide a comprehensive solution or service. It's a broader term that encompasses not only the program's code but also all associated elements required for its usage, distribution and maintenance.
Ex. Operating systems (like Windows, Mac OS, Linux), office suites (MS word, Google Workspace), video games etc.

(b). Verification — Verification is the process of determining whether the output of one phase of software developed conforms to that of its previous phase. The objective of verification is to check if the work products produced after a phase conform to that which was input to that phase.

Validation → Validation is the process of determining whether a fully developed software conforms to its requirement specification. Validation is applied to the fully developed & integrated software. to check if it satisfies the customer's requirements.

(d). Testing -

Testing involves the systematic execution of software to identify defects or bugs. It is performed to validate that the software meets the specified requirements and behaves as expected. ~~Testing can be~~

Testing aims to find issues in the software and assess its functionality against predefined criteria. It is used to identify bugs, inconsistencies, or deviations from the expected behavior

Debugging →.

It is process of identifying, isolating, and fixing defects or issues found during testing or general usage of the software. It involves investigating the root cause of unexpected behaviour or errors in the code and making corrections to ensure the software work as intended.

e). Software Reuse →.

Software reuse refers to the practice of using existing software assets, components, modules or libraries in the development of new software systems or applications. Instead of building ~~the~~ so everything from scratch., software reuse involves leveraging existing software artifacts to save time effort / resources.

Software Engineering →.

It is the process of designing, developing, testing and maintaining ~~software~~ high-quality software systems. It involves applying engineering principles and practices to create reliable, efficient and scalable software solutions that meet specific user requirements.

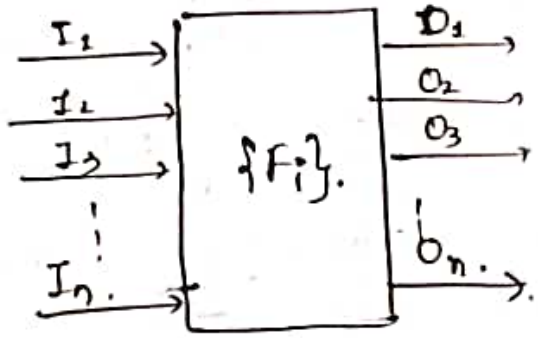
(f) SRS Document –

The important parts of the SRS document are –.

① functional requirements of the system.
② Non- functional requirement of the system.
③ goal of implementation

① functional Requirements →.

This part discusses the functionalities needed from the system. The system is taken into account to perform a group of high-level function f_i. The Each function f_i of system can be considered as a transformation of a set of input I_i to the corresponding set of output O_i.



Input

Output.

② Non-functional Requirements.

Non-functional necessities accomodate the characteristics of the system which may not be expressed as function-like maintainability, movability, usability of the system etc. Non- functional requirements may include –.

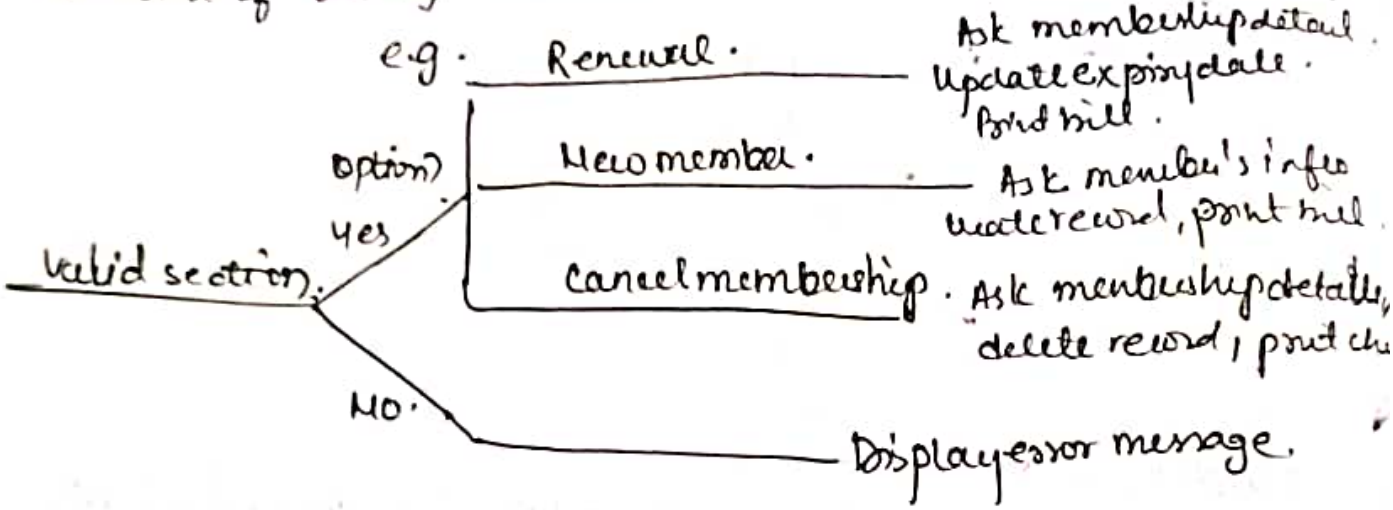
① Reliability issues –
② Accuracy of results.
③ Human- computer interface issues.
④. Constraints on system Implementation

(3) Goals of Implementation -

This part documents the some genral suggestion relating to development. These suggestion guide trade-off among style goals.

(9) Decision Tree.

A decision tree gives a graphic view of the processing logic involved in the decision making and the corresponding actions taken.
The edges of a decision tree represent conditions & the leaf nodes represent the actions to be performed depending upon the outcome of testing conditions.



(2) Decision Table.

A decision table shows the decision making logic and the corresponding actions taken in a tabular or matrix form.
The upper rows of the table specify the variable or conditions to be evaluated and lower rows specify the actions to be taken when an evaluation test is satisfie

eg.

Conditions			
Valid section.	NO	YES	YES.
New member.	—	YES	NO.
Renewal.	—	NO	YES.
Cancellation.	—	NO	NO
Actions	—	—	—

2

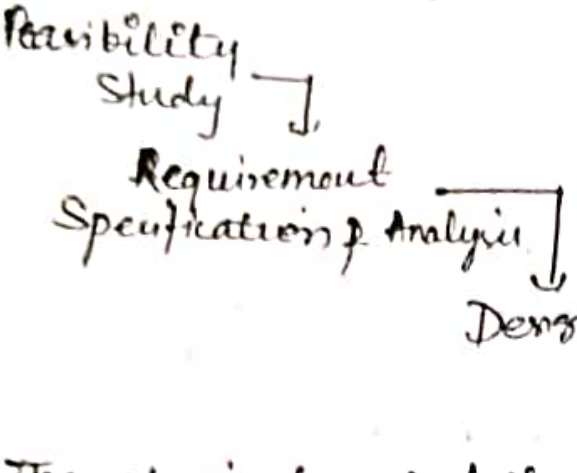
Decision trees are easier to read and understand when no. of conditions are small. On the other hand, a decision table allure the analyst to look at every possible combination of conditions which he might otherwise omit.

Decision trees can more intuitively represent multilevel decision making heirarchically, whereas decision table can only represent a single decision to select the appropriate action for execution.

When very large number of decisions are involved, the decision table representation is puffed as decision trees become complex with large no. of decisions.

P4TM10D

(5) The classical waterfall method arranges all the phases sequentially so that each new phase depends on the outcome of the previous phase.



The classical waterfall model provides no feedback paths to the preceding phases. In practical development environments, the developers commit a large no. of mistakes. These errors are generally detected at a later stage of some work needs to be redone. Therefore in any non-trivial software development project, it becomes nearly impossible to follow this model.

Also this model recommends that phases be carried out sequentially. It leads to let a large no. of teams members idle for an extended period of time.

In practical scenario rather than having a precise point of time at which the transition occurs, the different phases need to overlap for cost & time efficiency reasons.