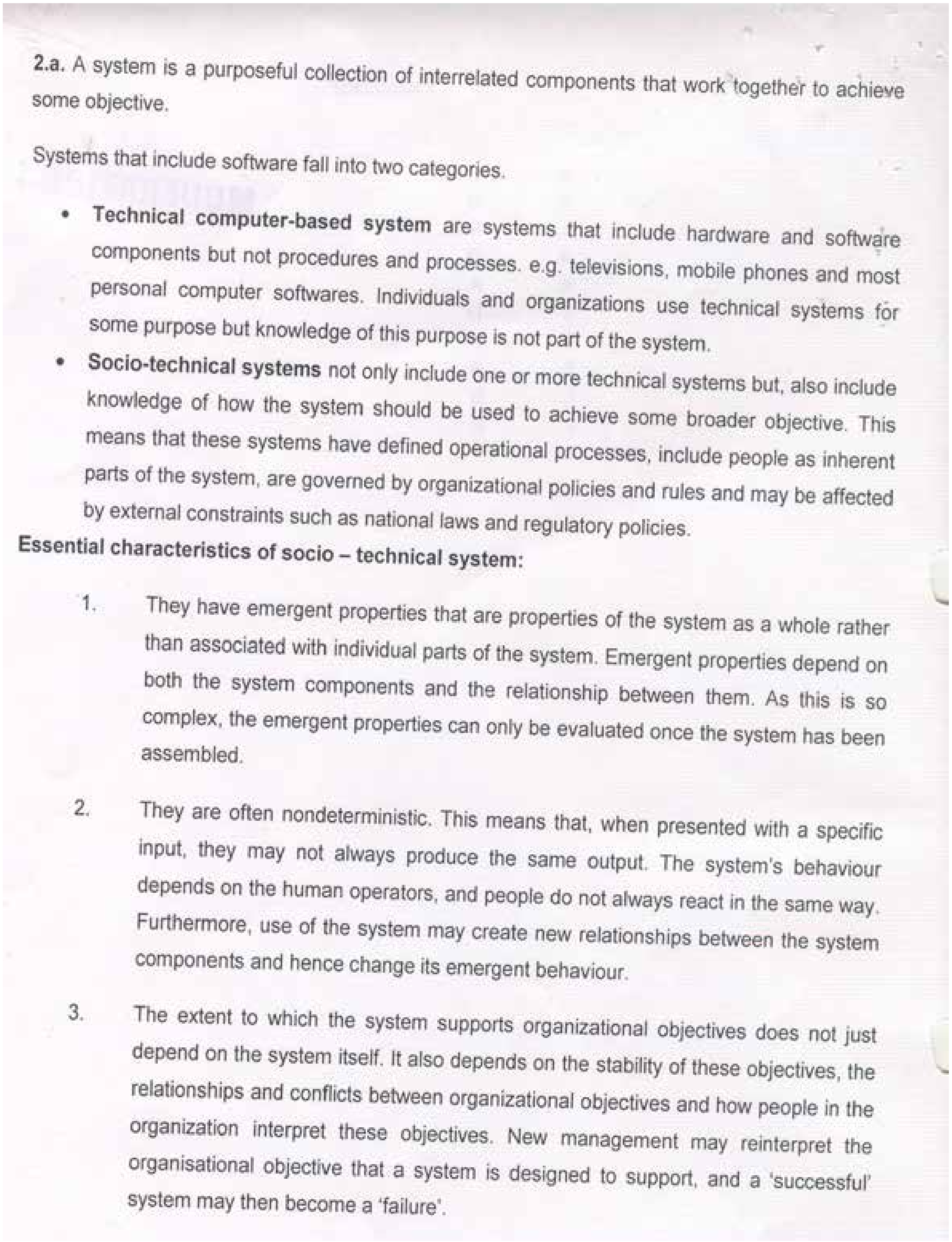
Q.2 a. Explain the concept of socio-technical systems. What are the characteristics of socio-technical systems?

Answer:



2.a. A system is a purposeful collection of interrelated components that work together to achieve some objective.

Systems that include software fall into two categories.

- **Technical computer-based system** are systems that include hardware and software components but not procedures and processes. e.g. televisions, mobile phones and most personal computer softwares. Individuals and organizations use technical systems for some purpose but knowledge of this purpose is not part of the system.
- **Socio-technical systems** not only include one or more technical systems but, also include knowledge of how the system should be used to achieve some broader objective. This means that these systems have defined operational processes, include people as inherent parts of the system, are governed by organizational policies and rules and may be affected by external constraints such as national laws and regulatory policies.

Essential characteristics of socio – technical system:

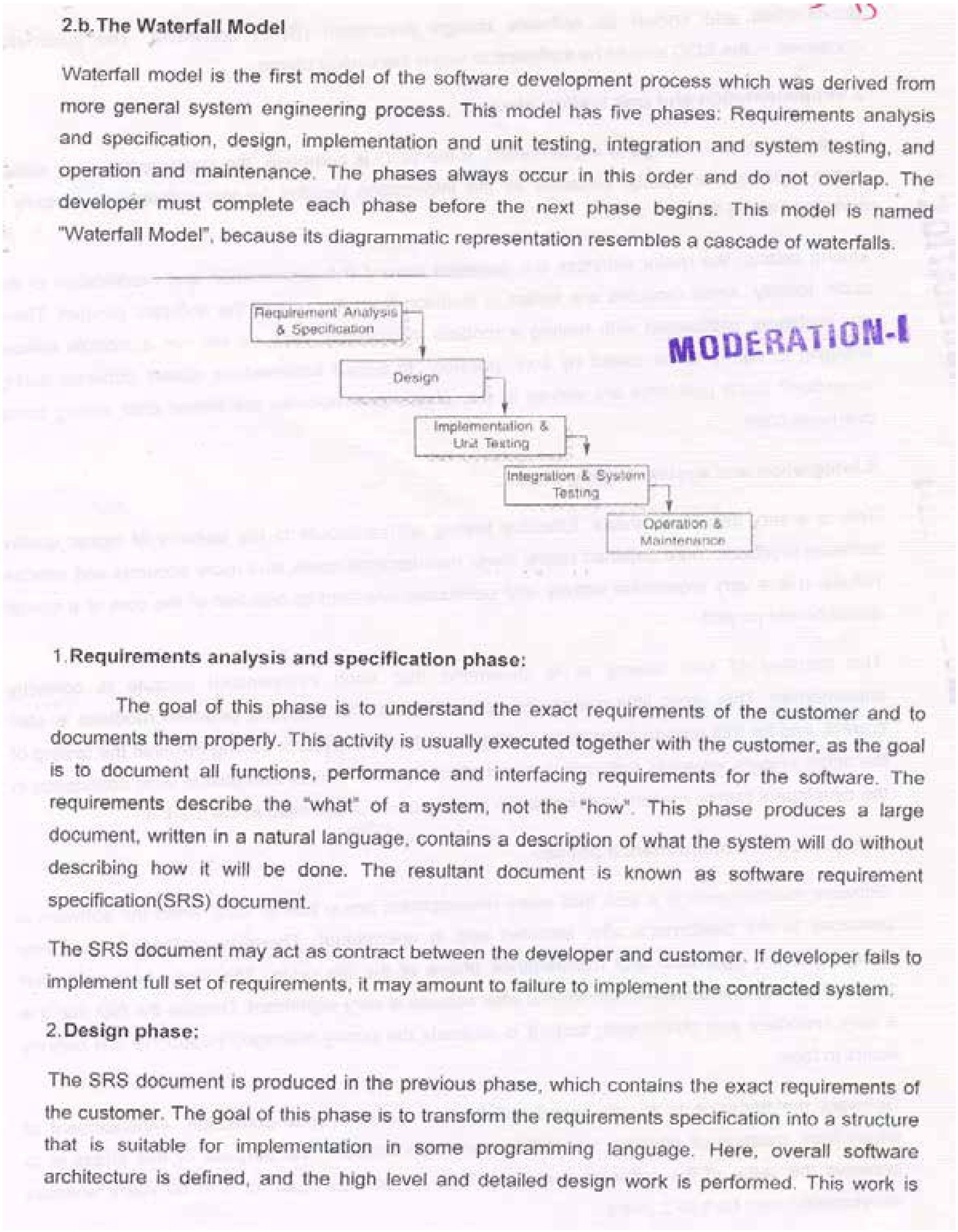
1. They have emergent properties that are properties of the system as a whole rather than associated with individual parts of the system. Emergent properties depend on both the system components and the relationship between them. As this is so complex, the emergent properties can only be evaluated once the system has been assembled.

2. They are often nondeterministic. This means that, when presented with a specific input, they may not always produce the same output. The system's behaviour depends on the human operators, and people do not always react in the same way. Furthermore, use of the system may create new relationships between the system components and hence change its emergent behaviour.

3. The extent to which the system supports organizational objectives does not just depend on the system itself. It also depends on the stability of these objectives, the relationships and conflicts between organizational objectives and how people in the organization interpret these objectives. New management may reinterpret the organisational objective that a system is designed to support, and a 'successful' system may then become a 'failure'.

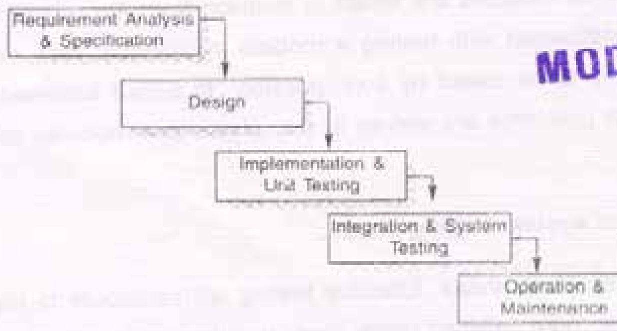
b. Explain the Waterfall model. Illustrate your answer with the help of Block diagram.

Answer:



2.b. The Waterfall Model

Waterfall model is the first model of the software development process which was derived from more general system engineering process. This model has five phases: Requirements analysis and specification, design, implementation and unit testing, integration and system testing, and operation and maintenance. The phases always occur in this order and do not overlap. The developer must complete each phase before the next phase begins. This model is named "Waterfall Model", because its diagrammatic representation resembles a cascade of waterfalls.


Requirement Analysis & Specification
Design
Implementation & Unit Testing
Integration & System Testing
Operation & Maintenance

MODERATION-I

1. **Requirements analysis and specification phase:**

The goal of this phase is to understand the exact requirements of the customer and to documents them properly. This activity is usually executed together with the customer, as the goal is to document all functions, performance and interfacing requirements for the software. The requirements describe the "what" of a system, not the "how". This phase produces a large document, written in a natural language, contains a description of what the system will do without describing how it will be done. The resultant document is known as software requirement specification(SRS) document.

The SRS document may act as contract between the developer and customer. If developer fails to implement full set of requirements, it may amount to failure to implement the contracted system.

2. **Design phase:**

The SRS document is produced in the previous phase, which contains the exact requirements of the customer. The goal of this phase is to transform the requirements specification into a structure that is suitable for implementation in some programming language. Here, overall software architecture is defined, and the high level and detailed design work is performed. This work is documented and known as software design description (SDD) document. The information contained in the SDD should be sufficient to begin the coding phase.

3.Implementation and unit testing phase :

During this phase, design is implemented. If the SDD is complete, the implementation or coding phase proceeds smoothly, because all the information needed by the software developers is contained in the SDD.

During testing, the major activities are centered around the examination and modification of the code. Initially, small modules are tested in isolation from the rest of the software product. There are problems associated with testing a module in isolation. How do we run a module without anything to call it, to be called by it or, possibly, to output intermediate values obtained during execution? Such problems are solved in this phase and modules are tested after writing some overhead code.

4.Integration and system testing phase:

This is a very important phase. Effective testing will contribute to the delivery of higher quality software products, more satisfied users, lower maintenance costs, and more accurate and reliable results. It is a very expensive activity and consumes one-third to one half of the cost of a typical development project.

The purpose of unit testing is to determine that each independent module is correctly implemented. This gives little chance to determine that the interface between modules is also correct, and for this reason integration testing is performed. System testing involves the testing of the entire system, whereas software is a part of the system. This is essential to build confidence in the developers before system is delivered to the customer or released in the market.

5.Operation and maintenance phase:

Software maintenance is a task that every development group has to face, when the software is delivered to the customer's site, installed and is operational. Therefore, release of software inaugurates the operation and maintenance phase of the life cycle. The time spent and effort required to keep the software operational after release is very significant. Despite the fact that it is a very important and challenging task; it is routinely the poorly managed headache that nobody wants to face.

Software maintenance is a very broad activity that includes error correction, enhancement of capabilities, deletion of obsolete capabilities, and optimization. The purpose of this phase is to preserve the value of the software over time. This phase may span for 5 to 50 years whereas development may be 1 to 3 years.

This model is easy to understand and reinforces the notion of "define before design" and "design before code". This model expects complete and accurate requirements early in the process, thus delaying the discovery of serious errors. It also does not incorporate any kind of risk assessment.

MODERATION-I

Due to these weaknesses, the application of waterfall model should be limited to situations where the requirements and their implementation are well understood. For example, if an organization has experience in developing accounting systems then building a new accounting system based on existing designs could be easily managed with the waterfall model.

Q.3 a. Explain various stages of Requirement Engineering process. Illustrate your answer with the help of Block Diagram.

Answer:

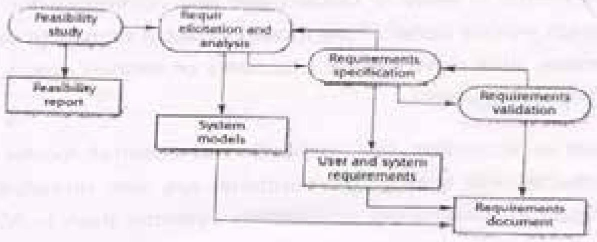
3.a. Requirement Engineering : Software specification or requirements engineering is the process of understanding and defining what services are required from the system and identifying the constraints on the system's operation and development. Requirements engineering is a critical stage of the software process as errors at this stage inevitably leads to later problems in the system design and implementation.

Requirement Engineering Process: This process leads to the production of a requirements document that is the specification for the system. Requirements are presented at two levels of detail in this document. End-users and customers require a high level statement of the requirements; system developers require a more detailed system specification.

Phases in Requirement Engineering Process:

1. **Feasibility Study**: An estimate is made of whether the identified user needs can be satisfied using current software and hardware technologies. Feasibility study considers whether the proposed system is cost effective from a business point of view and whether it can be developed within existing budgetary constraints. A feasibility study should be cheap and quick. The result of feasibility study helps in making decision of whether to go ahead with a more detailed analysis.
2. **Requirements elicitation and analysis:** This is the process of deriving the system requirements through observation of existing systems, discussions with potential users, task analysis and so on. This involves the development of one or more system models and prototypes. These help the analyst in understanding the system to be specified.
3. **Requirement specification:** The activity of translating the information gathered during the analysis activity into a document that defines a set of requirements. Various types of requirements that are included in this document are:
    1. **User Requirements:** are abstract statements of the system requirements for the customer and end-user of the system.
    2. **System Requirements:** are more detailed description of the functionality to be provided.

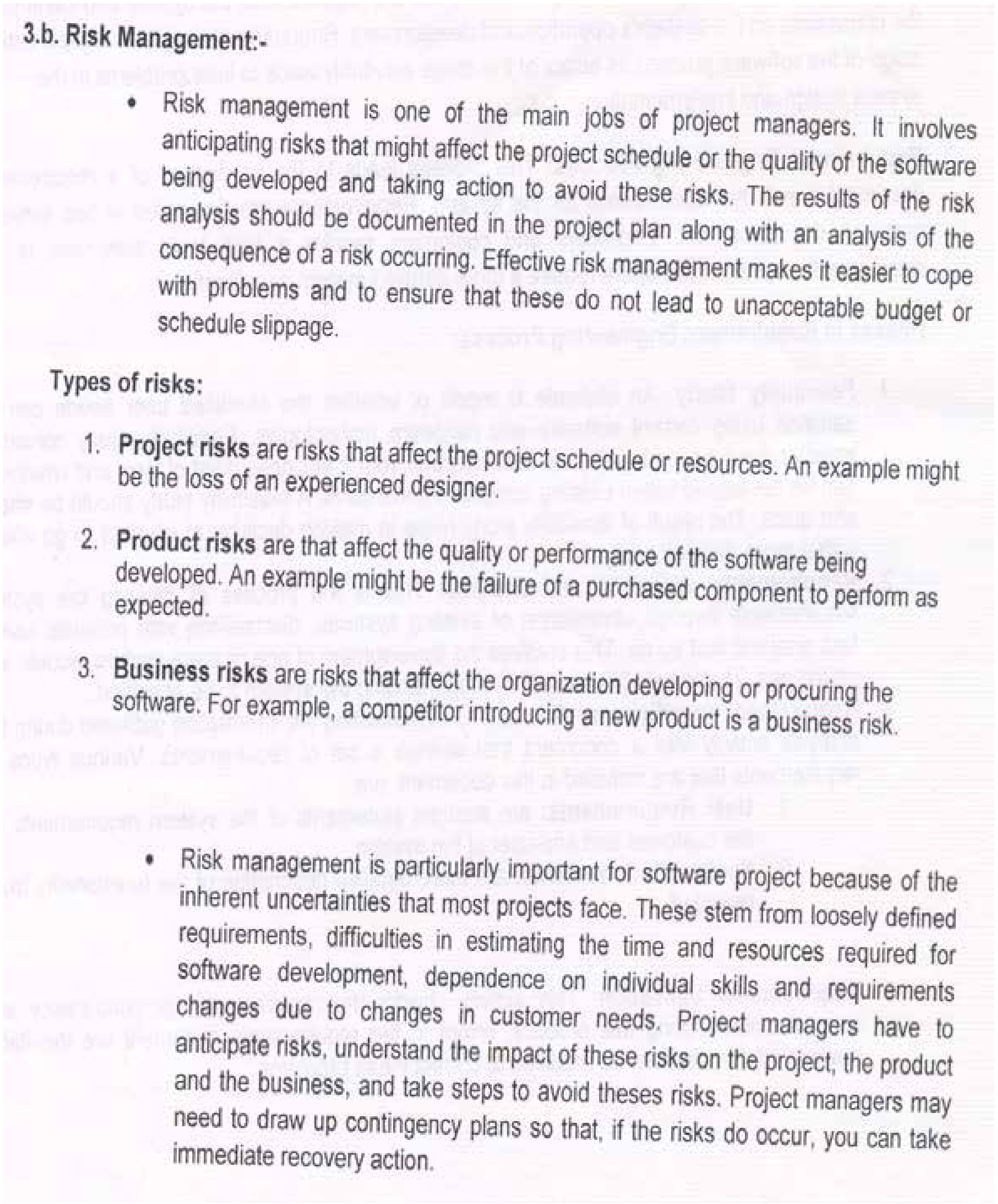
4. **Requirements Validation**: This activity checks the requirements for consistency and completeness. During this process, errors in the requirements document are inevitably discovered. It must then be modified to correct these problems.



The activities in the requirements process are not supposed to be carried out in strict sequence. Requirement analysis continues during definition and specification and new requirements keep on coming throughout the process. Therefore, the activities of analysis, definition and specification are interleaved.

b. What is Software Requirement Specification (SRS)? List five desirable characteristics of a good SRS document.

Answer:



3.b. Risk Management:-

- Risk management is one of the main jobs of project managers. It involves anticipating risks that might affect the project schedule or the quality of the software being developed and taking action to avoid these risks. The results of the risk analysis should be documented in the project plan along with an analysis of the consequence of a risk occurring. Effective risk management makes it easier to cope with problems and to ensure that these do not lead to unacceptable budget or schedule slippage.

Types of risks:

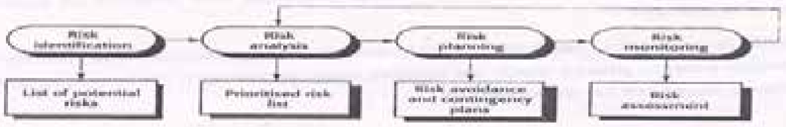
1. **Project risks** are risks that affect the project schedule or resources. An example might be the loss of an experienced designer.
2. **Product risks** are that affect the quality or performance of the software being developed. An example might be the failure of a purchased component to perform as expected.
3. **Business risks** are risks that affect the organization developing or procuring the software. For example, a competitor introducing a new product is a business risk.

- Risk management is particularly important for software project because of the inherent uncertainties that most projects face. These stem from loosely defined requirements, difficulties in estimating the time and resources required for software development, dependence on individual skills and requirements changes due to changes in customer needs. Project managers have to anticipate risks, understand the impact of these risks on the project, the product and the business, and take steps to avoid theses risks. Project managers may need to draw up contingency plans so that, if the risks do occur, you can take immediate recovery action.

Q.4 a. Explain the various stages in the general process of an object-oriented design.

Answer:

The process of risk management is illustrated in Fig.



It involves several stages:-

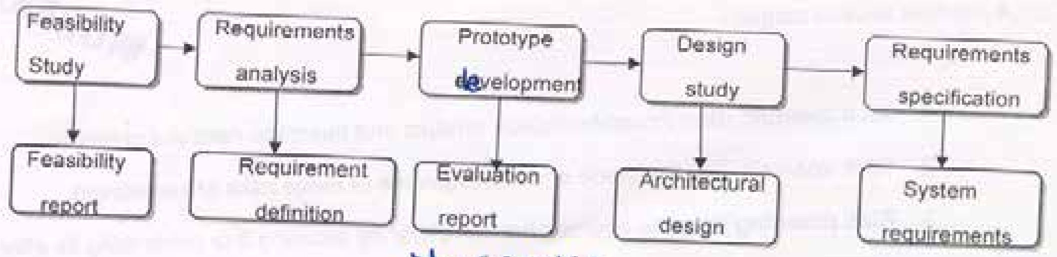
MODERATIO

1. **Risk identification** Possible project, product and business risks are identified.
2. **Risk analysis** The likelihood and consequence of these risks are assessed.
3. **Risk planning** Plans to address the risk either by avoiding it or minimizing its effects on the project is drawn up.
4. **Risk monitoring** The risk is constantly assessed and planes for risk mitigation are revised as more information about the risk becomes available.

- The risk management process, like all other project planning, is an interactive process which continues throughout the project. Once an initial set of plans are drawn up, the situation is monitored. As more information about the risks becomes available, the risks have to be reanalyzed and new priorites established. The risk avoidance and contingency plans may be modified as new risk information emerges.

'Coding 70% complete' that can't be checked are useless for project management. One can't check whether this state has been achieved because the amount of code that still has to be developed is uncertain.

A *deliverable* is a project result that is delivered to the customer. It is usually delivered at the end of some major project phase such as specification or design. Deliverable are usually milestones, but milestones need not be deliverables.



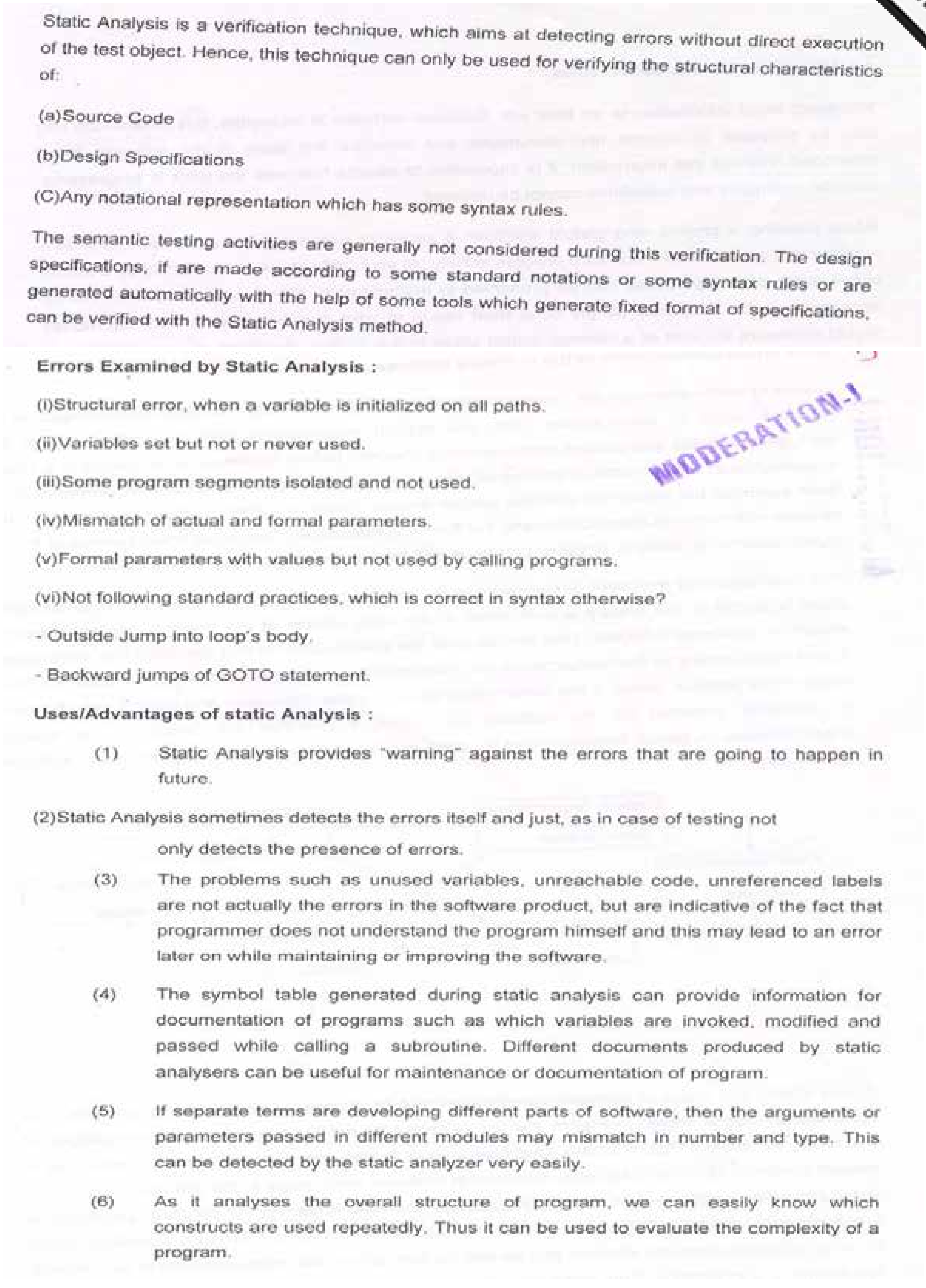
MILESTONES

Milestones may be internal project results that are used by the project manager to check project progress but which are not delivered to the customer.

To establish milestones, the software process must be broken down into basic activities with associated outputs. For example, Figure shows possible activities involved in requirements specification when prototyping is used to help validate requirements. The milestones in this case are the completion of the outputs for each activity. The project deliverables, which are delivered on the customer, are the requirements definition and the requirements specification.

b. Explain various key factors that are considered when planning application system reuse.

Answer:



Static Analysis is a verification technique, which aims at detecting errors without direct execution of the test object. Hence, this technique can only be used for verifying the structural characteristics of:

(a)Source Code

(b)Design Specifications

(C)Any notational representation which has some syntax rules.

The semantic testing activities are generally not considered during this verification. The design specifications, if are made according to some standard notations or some syntax rules or are generated automatically with the help of some tools which generate fixed format of specifications, can be verified with the Static Analysis method.

Errors Examined by Static Analysis :

(i)Structural error, when a variable is initialized on all paths.

(ii)Variables set but not or never used.

(iii)Some program segments isolated and not used.

(iv)Mismatch of actual and formal parameters.

(v)Formal parameters with values but not used by calling programs.

(vi)Not following standard practices, which is correct in syntax otherwise?

- Outside Jump into loop's body.

- Backward jumps of GOTO statement.

MODERATION-I

Uses/Advantages of static Analysis :

(1) Static Analysis provides "warning" against the errors that are going to happen in future.

(2)Static Analysis sometimes detects the errors itself and just, as in case of testing not only detects the presence of errors.

(3) The problems such as unused variables, unreachable code, unreferenced labels are not actually the errors in the software product, but are indicative of the fact that programmer does not understand the program himself and this may lead to an error later on while maintaining or improving the software.

(4) The symbol table generated during static analysis can provide information for documentation of programs such as which variables are invoked, modified and passed while calling a subroutine. Different documents produced by static analysers can be useful for maintenance or documentation of program.

(5) If separate terms are developing different parts of software, then the arguments or parameters passed in different modules may mismatch in number and type. This can be detected by the static analyzer very easily.

(6) As it analyses the overall structure of program, we can easily know which constructs are used repeatedly. Thus it can be used to evaluate the complexity of a program.

Q.5 a. Explain the process of Formal specifications in the software process. List various activities that are performed while developing formal specifications of sub system interface.

Answer:

5. a.Formal Specification in the software process:

Critical systems development usually involves a plan-based software process that is based on the waterfall model of development. Both the system requirements and the system design are expressed in detail and carefully analysed and checked before implementation begins. If a formal specification of the software is developed, this usually comes after the system requirements have been specified but before the detailed system design. There is a tight feedback loop between the detailed requirements specification and the formal specification. One of the main benefits of formal specification is its ability to uncover problems and ambiguities in the system requirements.

The involvement of the client decreases and the involvement of the contractor increases as more detail is added to the system specification. In the early stages of the process, the specification should be 'customer-oriented'. One should write the specification so that the client can understand it, and should make as few assumptions as possible about the software design. However, the final stage of the process, which is the construction of a complete, consistent and precise specification, is principally intended for the software contractor. It specifies the details of the system implementation. A formal language can be used at this stage to avoid ambiguity in the software specification.

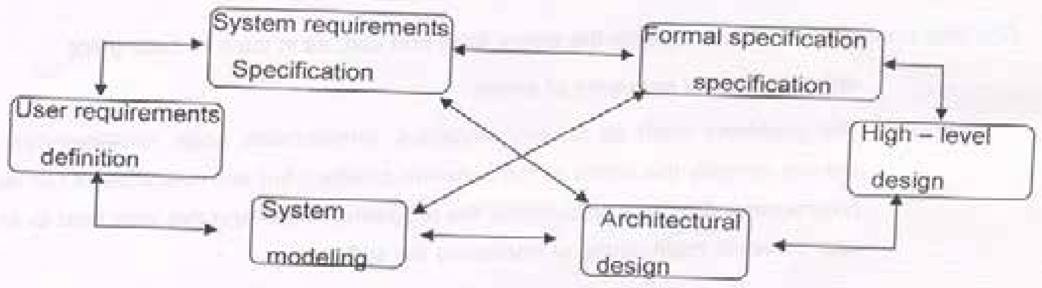
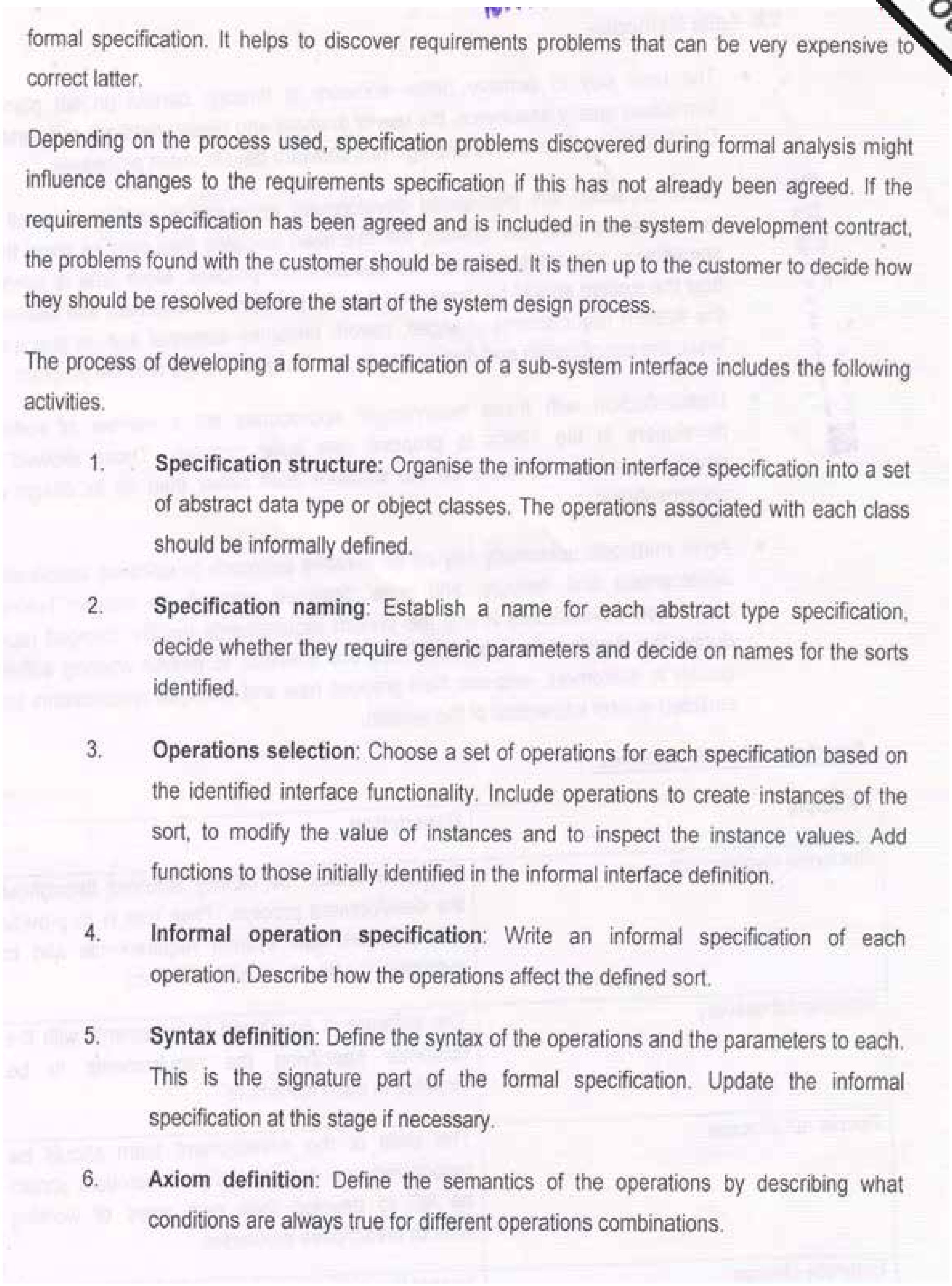

System requirements Specification
Formal specification specification
User requirements definition
High – level design
System modeling
Architectural design

Figure shows the stage of software specification and its interface with the design process. The specification stages shown in Figure are not independent nor are they necessarily developed in the sequence shown. Figure shows specification and design activities that may be carried out in parallel streams. There is a two-way relationship between each stage in the process. Information is fed from the specification to the design process and vice versa. As the detailed specifications are developed, understanding of that specification increases. Creating a formal specification forces to make a detailed systems analysis that usually reveals errors and inconsistencies in the informal requirements specification. This error detection is the most important argument for developing a



formal specification. It helps to discover requirements problems that can be very expensive to correct latter.

Depending on the process used, specification problems discovered during formal analysis might influence changes to the requirements specification if this has not already been agreed. If the requirements specification has been agreed and is included in the system development contract, the problems found with the customer should be raised. It is then up to the customer to decide how they should be resolved before the start of the system design process.

The process of developing a formal specification of a sub-system interface includes the following activities.

1. **Specification structure:** Organise the information interface specification into a set of abstract data type or object classes. The operations associated with each class should be informally defined.

2. **Specification naming**: Establish a name for each abstract type specification, decide whether they require generic parameters and decide on names for the sorts identified.

3. **Operations selection**: Choose a set of operations for each specification based on the identified interface functionality. Include operations to create instances of the sort, to modify the value of instances and to inspect the instance values. Add functions to those initially identified in the informal interface definition.

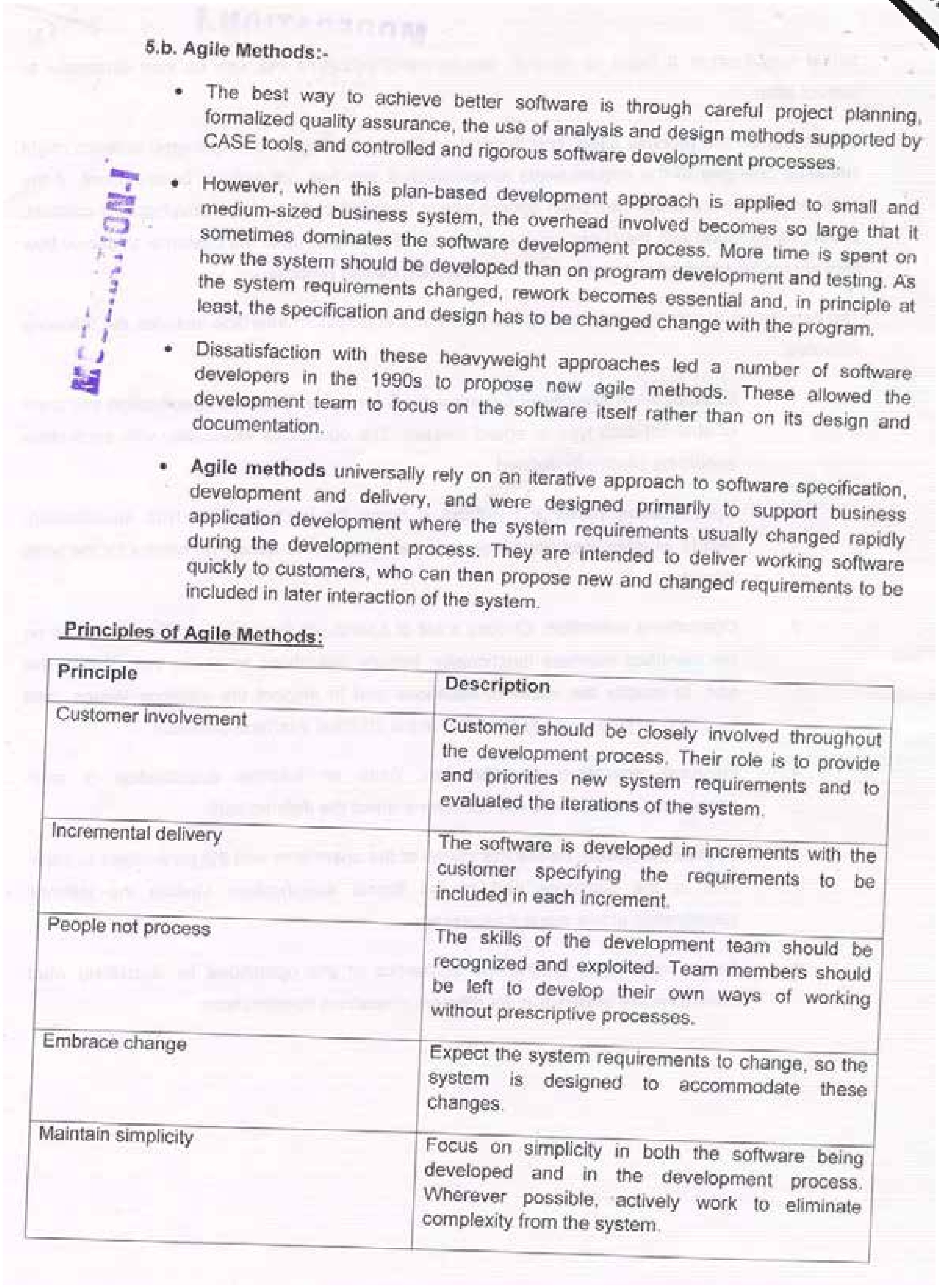
4. **Informal operation specification**: Write an informal specification of each operation. Describe how the operations affect the defined sort.

5. **Syntax definition**: Define the syntax of the operations and the parameters to each. This is the signature part of the formal specification. Update the informal specification at this stage if necessary.

6. **Axiom definition**: Define the semantics of the operations by describing what conditions are always true for different operations combinations.

b. Define and explain Agile Methods. What are Principles of Agile Methods? Illustrate with the help of taking example of widely used Agile methods.

Answer:



5.b. Agile Methods:-

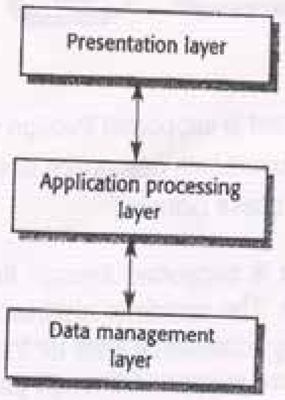
- The best way to achieve better software is through careful project planning, formalized quality assurance, the use of analysis and design methods supported by CASE tools, and controlled and rigorous software development processes.
- However, when this plan-based development approach is applied to small and medium-sized business system, the overhead involved becomes so large that it sometimes dominates the software development process. More time is spent on how the system should be developed than on program development and testing. As the system requirements changed, rework becomes essential and, in principle at least, the specification and design has to be changed change with the program.
- Dissatisfaction with these heavyweight approaches led a number of software developers in the 1990s to propose new agile methods. These allowed the development team to focus on the software itself rather than on its design and documentation.
- **Agile methods** universally rely on an iterative approach to software specification, development and delivery, and were designed primarily to support business application development where the system requirements usually changed rapidly during the development process. They are intended to deliver working software quickly to customers, who can then propose new and changed requirements to be included in later interaction of the system.

<u>Principles of Agile Methods:</u>

Principle	Description
Customer involvement	Customer should be closely involved throughout the development process. Their role is to provide and priorities new system requirements and to evaluated the iterations of the system.
Incremental delivery	The software is developed in increments with the customer specifying the requirements to be included in each increment.
People not process	The skills of the development team should be recognized and exploited. Team members should be left to develop their own ways of working without prescriptive processes.
Embrace change	Expect the system requirements to change, so the system is designed to accommodate these changes.
Maintain simplicity	Focus on simplicity in both the software being developed and in the development process. Wherever possible, actively work to eliminate complexity from the system.

- Several server processes can run on a single server processor so there is not necessarily a 1:1 mapping between processes and processors in the system. Client and servers, are logical processes rather than the physical computers on which they execute.

- The design of client-server system should reflect the logical structure of the application that is being developed. One way to look at an application is illustrated in Fig. which shows an application structured into three layers. The presentation layer is concerned with presenting information to the user and with all user interaction. The application processing layer is concerned with implementing the logic of the application, and the data management layer is concerned with all database operations. In centralized systems, these need be clearly separated. However, when a distributed system is designed, a clear distinction should be made between them, as it then becomes possible to distribute each layer to a different computer.



Types of Client-Server Architecture:

1. Two-tier
2. Three-tier

The simplest client-server architecture is called a two-tier client-server architecture, where an application is organized as a server (or multiple identical servers) and a set of clients.

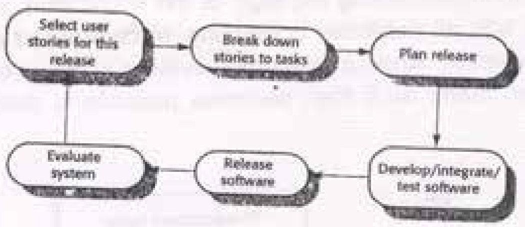
Types of Two-Tier Architecture:-

1. Thin-client model: In a thin-client model, all of the application processing and data management is carried out on the server. The client is simply responsible for running the presentation software.

2. Fat-client model: In this model, the server is only responsible for data management. The software on the client implements the application logic and the interactions with the system user.

Extreme programming:-

- Extreme programming (XP) is the best known and most widely used agile methods.
- Extreme programming involves a number of practices, summarized in Fig. that fit into the principles of agile methods:-


Select user stories for this release
Break down stories to tasks
Plan release
Develop/integrate/ test software
Release software
Evaluate system

- Incremental development is supported through small, frequent release of the system and by an approach to requirements description based on customer stories or scenarios that can be the basis for process planning.

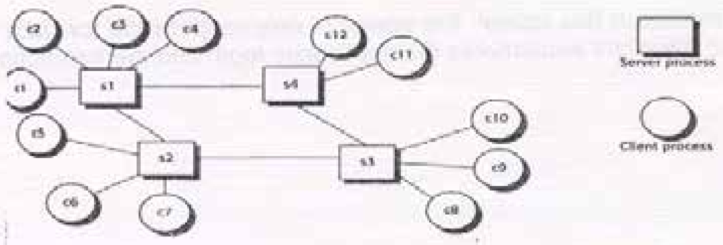
2. Customer involvement is supported through the full-time engagement of the customer in the development team. The customer representative takes part in the development and is responsible for defining acceptance tests for the system.
3. People, not process, are supported through pair programming, collective ownership of the system code, and a sustainable development process that does not involve excessively long working hours.
4. Change is supported through regular system releases, test-first development and continuous integration.
5. Maintaining simplicity is supported through constant refactoring to improve code quality and using simple design that do not anticipate future changes to the system.

Q.6 a. Describe Client-Server architecture. What are various types of Client-Server Architecture? Explain.

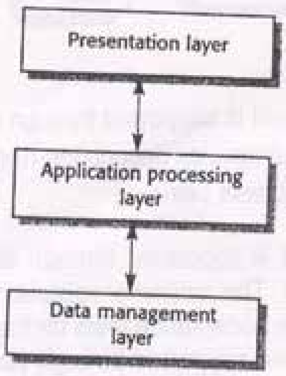
Answer:

6.a. Client-Server Architectures:-

- In a client-server architecture, an application is modeled as a set of services that are provided by servers and a set of clients that use these services.
- Clients need to be aware of the servers that are available but usually do not know of the existence of other clients. Clients and servers are separate processes, as shown in Fig., which is a logical model of distributed client-server architecture.



- Several server processes can run on a single server processor so there is not necessarily a 1:1 mapping between processes and processors in the system. Client and servers, are logical processes rather than the physical computers on which they execute.
- The design of client-server system should reflect the logical structure of the application that is being developed. One way to look at an application is illustrated in Fig. which shows an application structured into three layers. The presentation layer is concerned with presenting information to the user and with all user interaction. The application processing layer is concerned with implementing the logic of the application, and the data management layer is concerned with all database operations. In centralized systems, these need be clearly separated. However, when a distributed system is designed, a clear distinction should be made between them, as it then becomes possible to distribute each layer to a different computer.



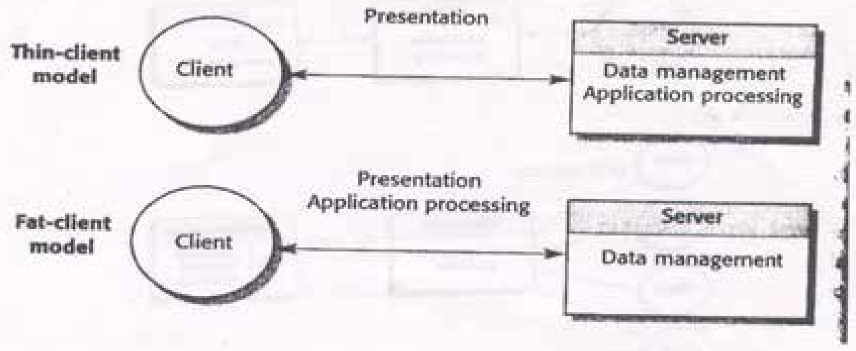
Types of Client-Server Architecture:

1. Two-tier
2. Three-tier

The simplest client-server architecture is called a two-tier client-server architecture, where an application is organized as a server (or multiple identical servers) and a set of clients.

Types of Two-Tier Architecture:-

1. Thin-client model: In a thin-client model, all of the application processing and data management is carried out on the server. The client is simply responsible for running the presentation software.
2. Fat-client model: In this model, the server is only responsible for data management. The software on the client implements the application logic and the interactions with the system user.

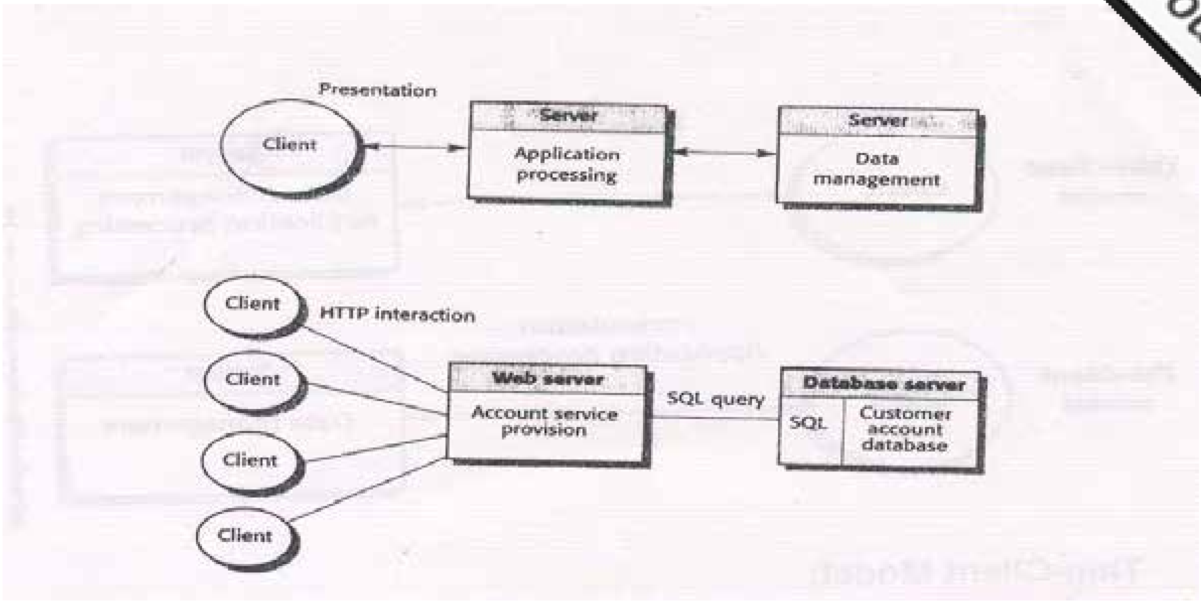


Thin-Client Model:

- A thin-client, two-tier architecture is the simplest approach to use when centralized legacy systems, are evolved to client-server architecture. The user interface for these systems is migrating to PCs, and the application itself acts as a server and handles all application processing and data management.
- A thin-client model may also be implemented when the client are simple network devices rather than PCs or workstations. The network device runs an Internet browser and the user interface implemented through that system.
- A major disadvantage of the thin-client model is that it places a heavy processing load on both the server and the network. The server is responsible for all computation, and this may involve the generation of significant network traffic between the client and the server. There is a lot of processing power available in modern computing devices, which is largely unused in the thin-client approach.

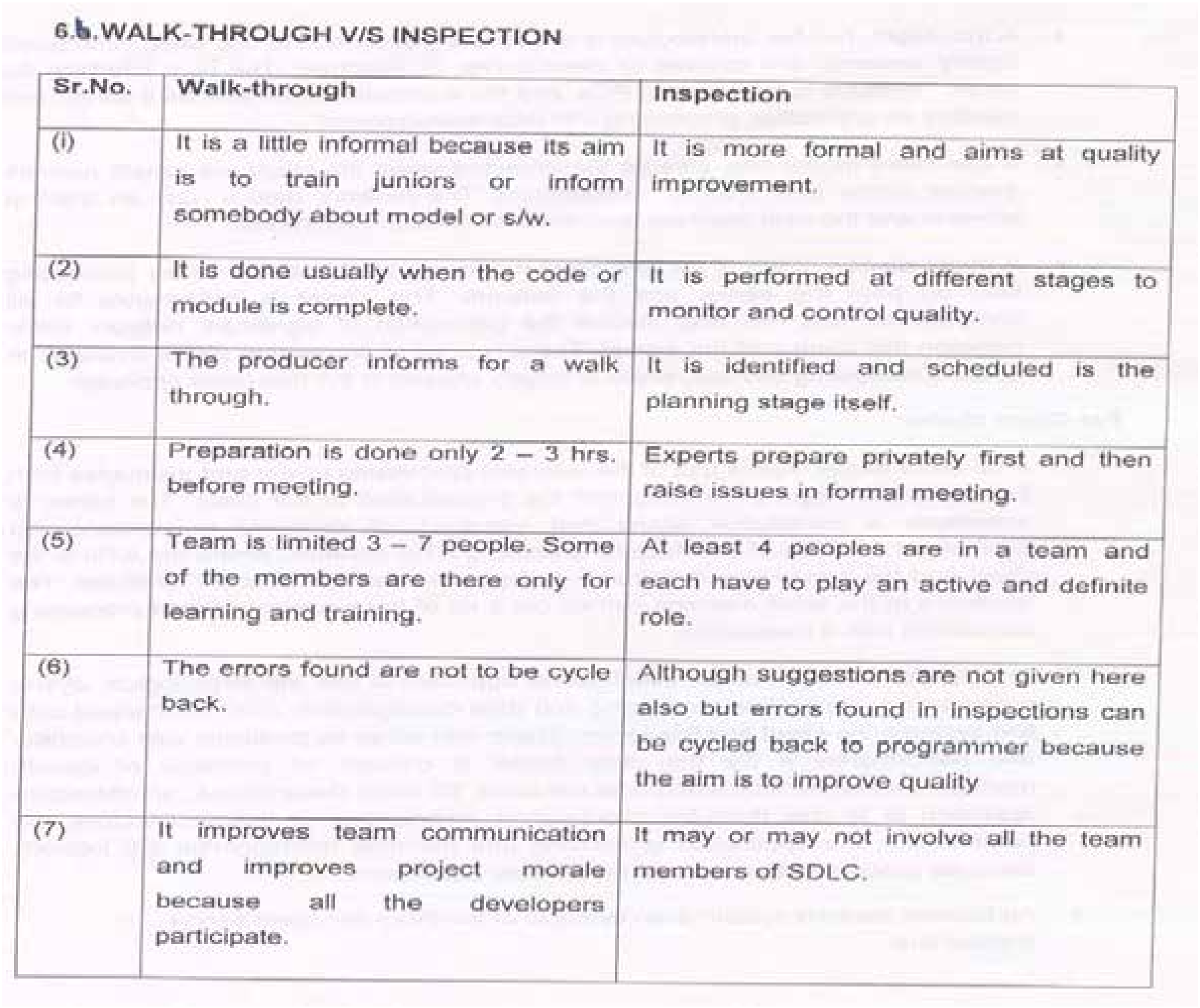
Fat-Client Model:

- A fat-client model makes use of the available processing power and distributes both the application logic processing and the presentation to the client. The server is essentially a transaction server that manages all database transactions. An example of this type of architecture is banking ATM systems, where the ATM is the client and the server is a mainframe running the customer account database. The hardware in the teller machine carries out a lot of the customer-related processing associated with a transaction.
- The problem with a two-tier client-server approach is that the three logical layers-presentation, application processing and data management- must be mapped onto two systems-the client and the server. There may either be problems with scalability and performance if the thin-client model is chosen, or problems of system management of the fat-client model are used. To avoid these issues, an alternative approach is to use three-tier client-server architecture. In this architecture, the presentation, the application processing and the data management are logically separate processes that execute on different processors.
- An Internet banking system is an example of the three-tier client-server architecture.



b. Write advantages and disadvantages of a shared repository.

Answer:

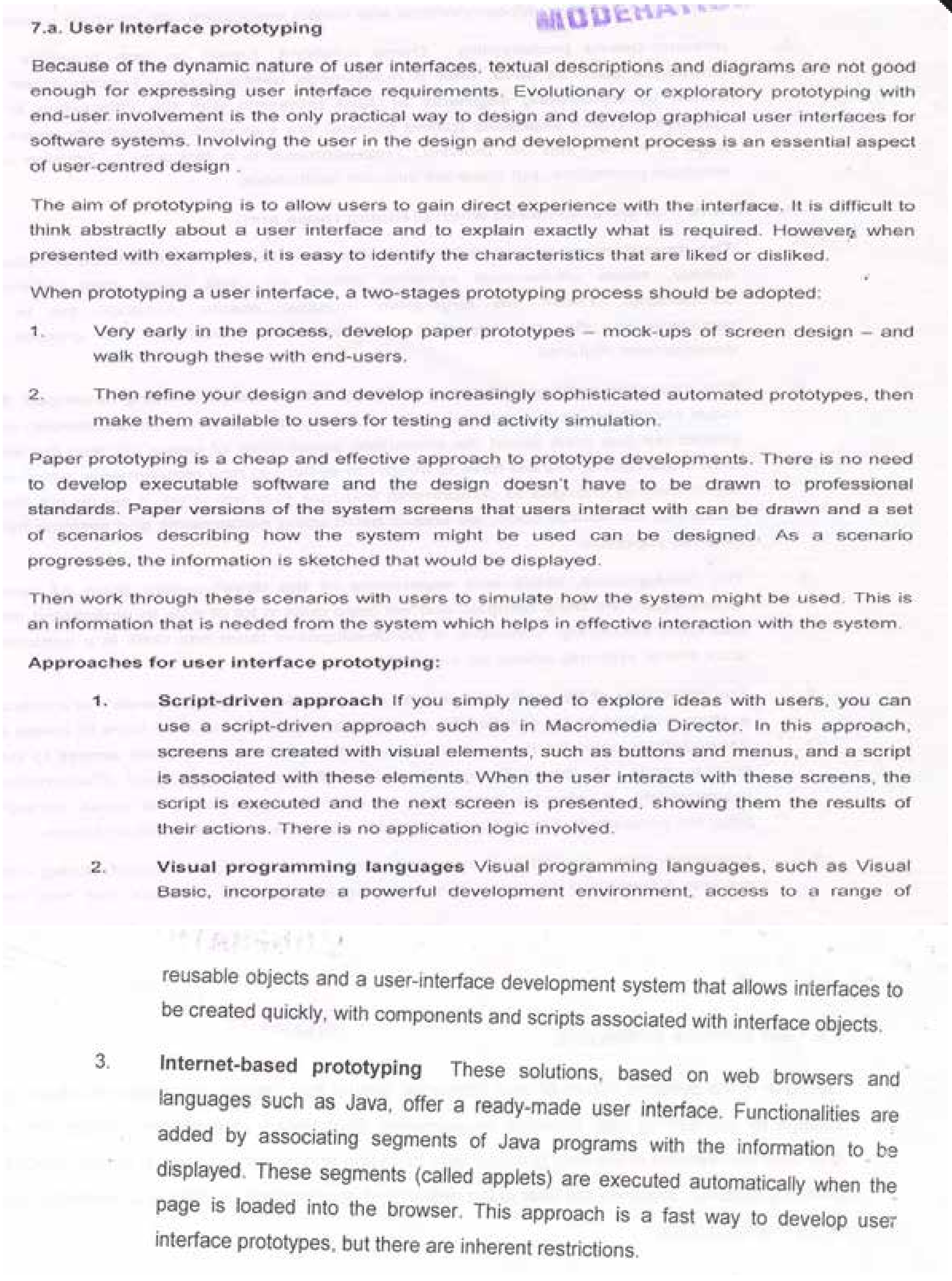


6.b.WALK-THROUGH V/S INSPECTION

Sr.No.	Walk-through	Inspection
(i)	It is a little informal because its aim is to train juniors or inform somebody about model or s/w.	It is more formal and aims at quality improvement.
(2)	It is done usually when the code or module is complete.	It is performed at different stages to monitor and control quality.
(3)	The producer informs for a walk through.	It is identified and scheduled is the planning stage itself.
(4)	Preparation is done only 2 – 3 hrs. before meeting.	Experts prepare privately first and then raise issues in formal meeting.
(5)	Team is limited 3 – 7 people. Some of the members are there only for learning and training.	At least 4 peoples are in a team and each have to play an active and definite role.
(6)	The errors found are not to be cycle back.	Although suggestions are not given here also but errors found in inspections can be cycled back to programmer because the aim is to improve quality
(7)	It improves team communication and improves project morale because all the developers participate.	It may or may not involve all the team members of SDLC.

Q.7 a. What are different approaches used for user interface prototyping. Explain.

Answer:



7.a. User Interface prototyping

Because of the dynamic nature of user interfaces, textual descriptions and diagrams are not good enough for expressing user interface requirements. Evolutionary or exploratory prototyping with end-user involvement is the only practical way to design and develop graphical user interfaces for software systems. Involving the user in the design and development process is an essential aspect of user-centred design .

The aim of prototyping is to allow users to gain direct experience with the interface. It is difficult to think abstractly about a user interface and to explain exactly what is required. However, when presented with examples, it is easy to identify the characteristics that are liked or disliked.

When prototyping a user interface, a two-stages prototyping process should be adopted:

1. Very early in the process, develop paper prototypes – mock-ups of screen design – and walk through these with end-users.
2. Then refine your design and develop increasingly sophisticated automated prototypes, then make them available to users for testing and activity simulation.

Paper prototyping is a cheap and effective approach to prototype developments. There is no need to develop executable software and the design doesn't have to be drawn to professional standards. Paper versions of the system screens that users interact with can be drawn and a set of scenarios describing how the system might be used can be designed. As a scenario progresses, the information is sketched that would be displayed.

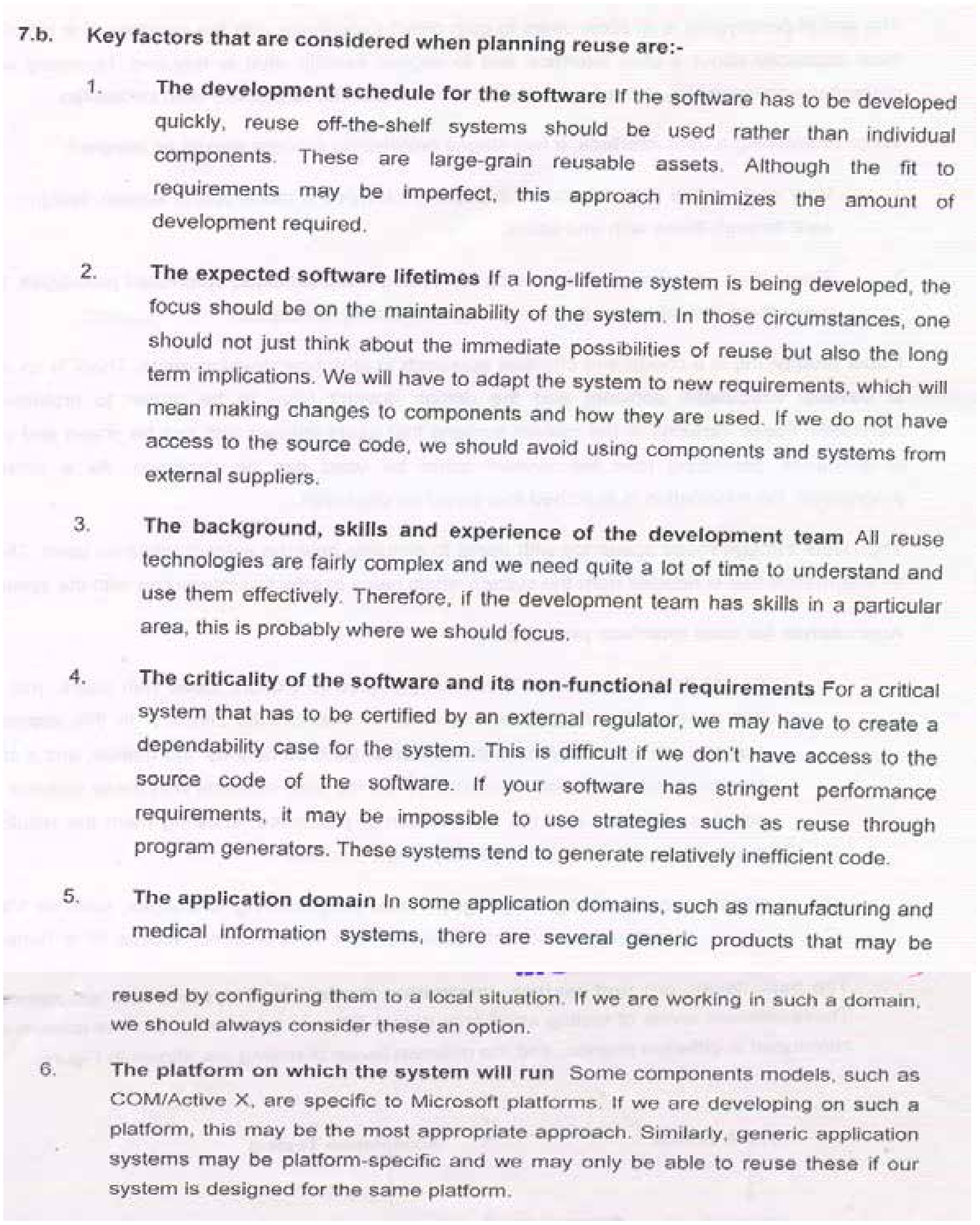
Then work through these scenarios with users to simulate how the system might be used. This is an information that is needed from the system which helps in effective interaction with the system.

Approaches for user interface prototyping:

1. **Script-driven approach** If you simply need to explore ideas with users, you can use a script-driven approach such as in Macromedia Director. In this approach, screens are created with visual elements, such as buttons and menus, and a script is associated with these elements. When the user interacts with these screens, the script is executed and the next screen is presented, showing them the results of their actions. There is no application logic involved.
2. **Visual programming languages** Visual programming languages, such as Visual Basic, incorporate a powerful development environment, access to a range of reusable objects and a user-interface development system that allows interfaces to be created quickly, with components and scripts associated with interface objects.
3. **Internet-based prototyping** These solutions, based on web browsers and languages such as Java, offer a ready-made user interface. Functionalities are added by associating segments of Java programs with the information to be displayed. These segments (called applets) are executed automatically when the page is loaded into the browser. This approach is a fast way to develop user interface prototypes, but there are inherent restrictions.

b. Components are usually developed using object oriented approach. Explain how components differ from objects.

Answer:



7.b. Key factors that are considered when planning reuse are:-

1. **The development schedule for the software** If the software has to be developed quickly, reuse off-the-shelf systems should be used rather than individual components. These are large-grain reusable assets. Although the fit to requirements may be imperfect, this approach minimizes the amount of development required.

2. **The expected software lifetimes** If a long-lifetime system is being developed, the focus should be on the maintainability of the system. In those circumstances, one should not just think about the immediate possibilities of reuse but also the long term implications. We will have to adapt the system to new requirements, which will mean making changes to components and how they are used. If we do not have access to the source code, we should avoid using components and systems from external suppliers.

3. **The background, skills and experience of the development team** All reuse technologies are fairly complex and we need quite a lot of time to understand and use them effectively. Therefore, if the development team has skills in a particular area, this is probably where we should focus.

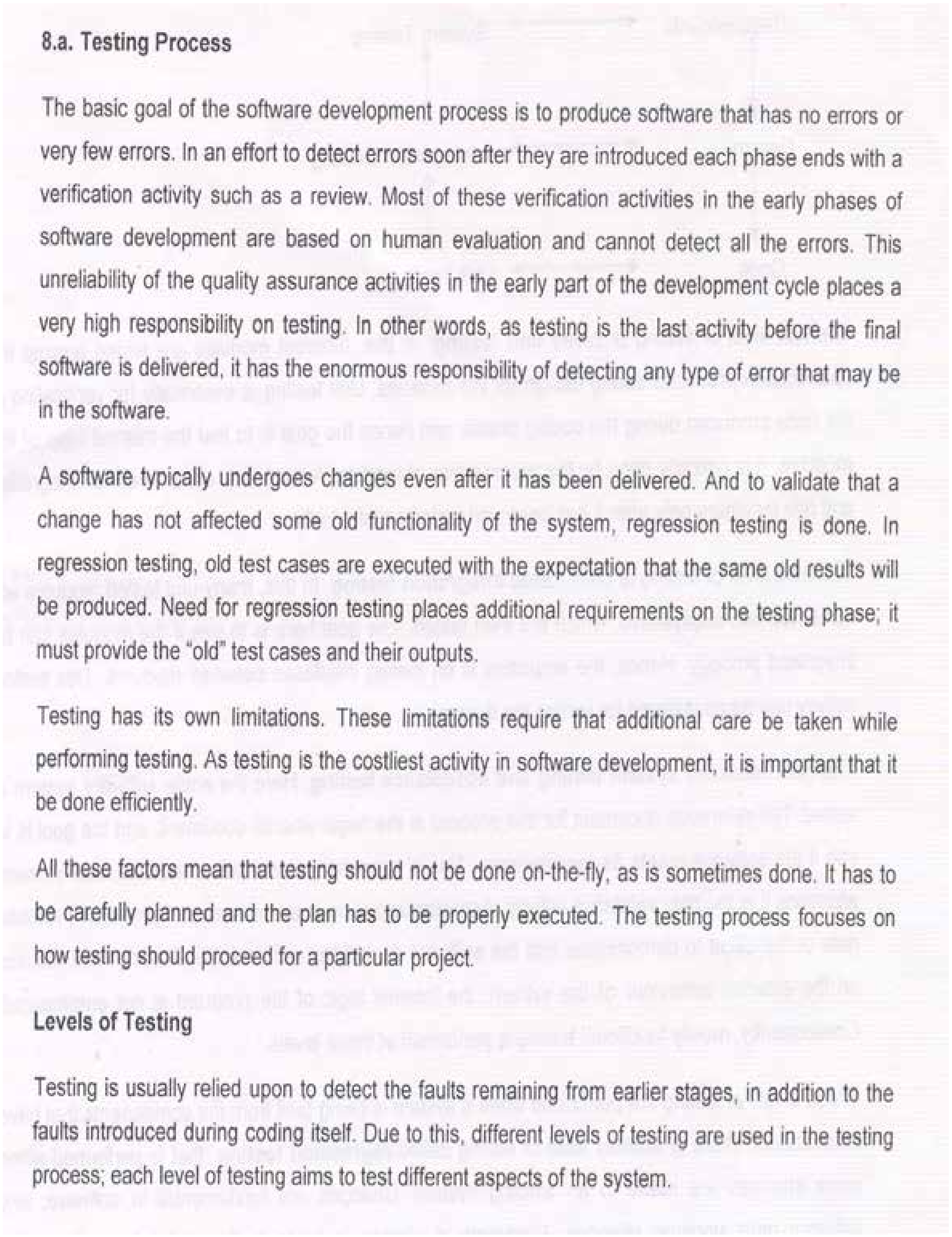
4. **The criticality of the software and its non-functional requirements** For a critical system that has to be certified by an external regulator, we may have to create a dependability case for the system. This is difficult if we don't have access to the source code of the software. If your software has stringent performance requirements, it may be impossible to use strategies such as reuse through program generators. These systems tend to generate relatively inefficient code.

5. **The application domain** In some application domains, such as manufacturing and medical information systems, there are several generic products that may be reused by configuring them to a local situation. If we are working in such a domain, we should always consider these an option.

6. **The platform on which the system will run** Some components models, such as COM/Active X, are specific to Microsoft platforms. If we are developing on such a platform, this may be the most appropriate approach. Similarly, generic application systems may be platform-specific and we may only be able to reuse these if our system is designed for the same platform.

Q.8 a. What are different levels of testing? List various goals of different levels, for each level specify which of the testing approaches is most suitable.

Answer:



8.a. Testing Process

The basic goal of the software development process is to produce software that has no errors or very few errors. In an effort to detect errors soon after they are introduced each phase ends with a verification activity such as a review. Most of these verification activities in the early phases of software development are based on human evaluation and cannot detect all the errors. This unreliability of the quality assurance activities in the early part of the development cycle places a very high responsibility on testing. In other words, as testing is the last activity before the final software is delivered, it has the enormous responsibility of detecting any type of error that may be in the software.

A software typically undergoes changes even after it has been delivered. And to validate that a change has not affected some old functionality of the system, regression testing is done. In regression testing, old test cases are executed with the expectation that the same old results will be produced. Need for regression testing places additional requirements on the testing phase; it must provide the "old" test cases and their outputs.

Testing has its own limitations. These limitations require that additional care be taken while performing testing. As testing is the costliest activity in software development, it is important that it be done efficiently.

All these factors mean that testing should not be done on-the-fly, as is sometimes done. It has to be carefully planned and the plan has to be properly executed. The testing process focuses on how testing should proceed for a particular project.

Levels of Testing

Testing is usually relied upon to detect the faults remaining from earlier stages, in addition to the faults introduced during coding itself. Due to this, different levels of testing are used in the testing process; each level of testing aims to test different aspects of the system.

The basic levels are **unit testing, integration testing, and system and acceptance testing**. These different levels of testing attempt to detect different types of faults. The relation of the faults introduced in different phases, and the different levels of testing are shown in Figure.

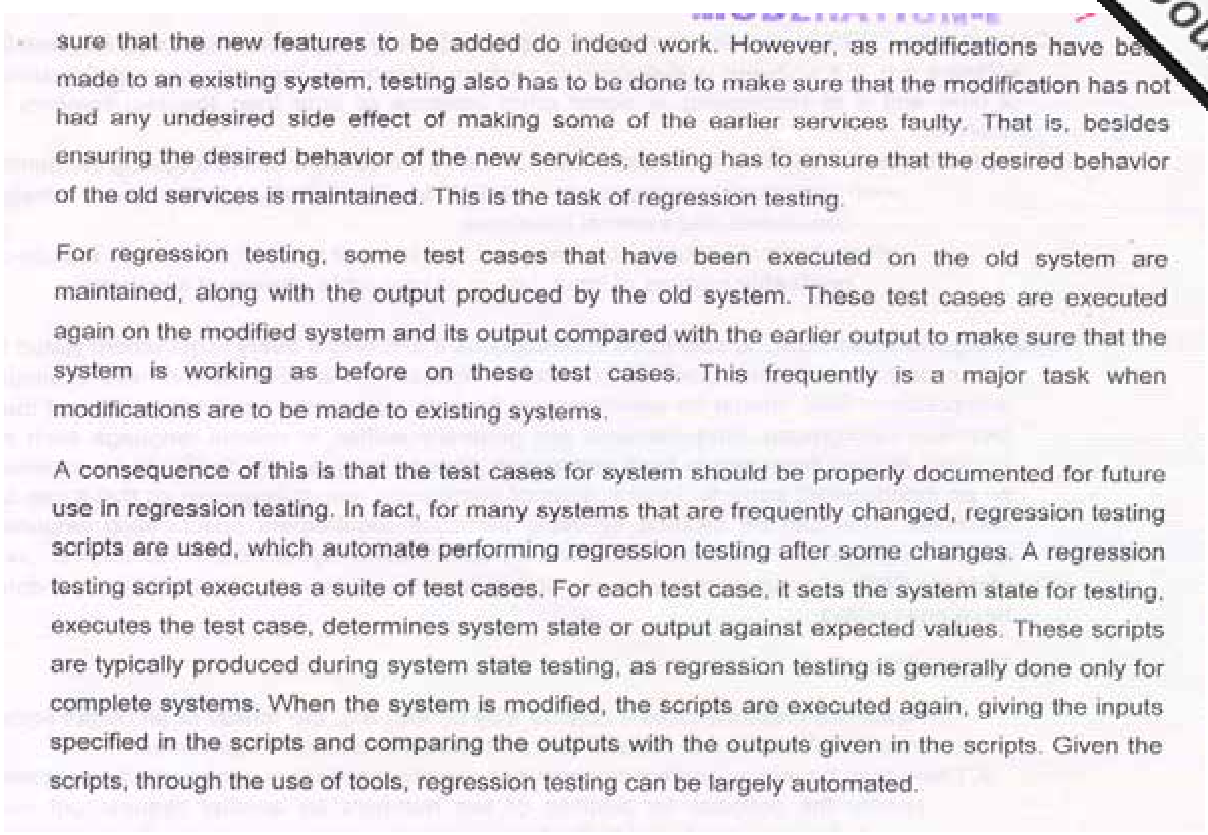
$$\begin{tikzcd}
\text{Client Needs} \arrow[r, leftrightarrow] \arrow[d] & \text{Acceptance Testing} \\
\text{Requirements} \arrow[r, leftrightarrow] \arrow[d] & \text{System Testing} \arrow[u] \\
\text{Design} \arrow[r, leftrightarrow] \arrow[d] & \text{Integration Testing} \arrow[u] \\
\text{Code} \arrow[r, leftrightarrow] & \text{Unit Testing} \arrow[u]
\end{tikzcd}$$

The first level of testing is called **unit testing**. In this, different modules are tested against the specification produced during design for the modules. Unit testing is essentially for verification of the code produced during the coding phase, and hence the goal is to test the internal logic of the modules. It is typically done by the programmer of module. A module is considered for integration and use by others only after it has been unit tested satisfactorily.

The next level of testing is often called **integration testing**. In this, many unit tested modules are combined into subsystems, which are then tested. The goal here is to see if the modules can be integrated properly. Hence, the emphasis is on testing interfaces between modules. This testing activity can be considered for testing the design.

The next levels are **system testing and acceptance testing**. Here the entire software system is tested. The reference document for this process is the requirements document, and the goal is to see if the software meets its requirements. This is essentially a validation exercise, and in many situations it is the only validation activity. Acceptance testing is sometimes performed with realistic data of the client to demonstrate that the software is working satisfactorily. Testing here focuses on the external behaviour of the system; the internal logic of the program is not emphasized. Consequently, mostly functional testing is performed at these levels.

These levels of testing are performed when a system is being built from the components that have been coded. There is another level of testing called **regression testing**, that is performed when some changes are made to an existing system. Changes are fundamental to software; any software must undergo changes. Frequently a change is made to "upgrade" the software by adding new features and functionality. Clearly, the modified software needs to be tested to make



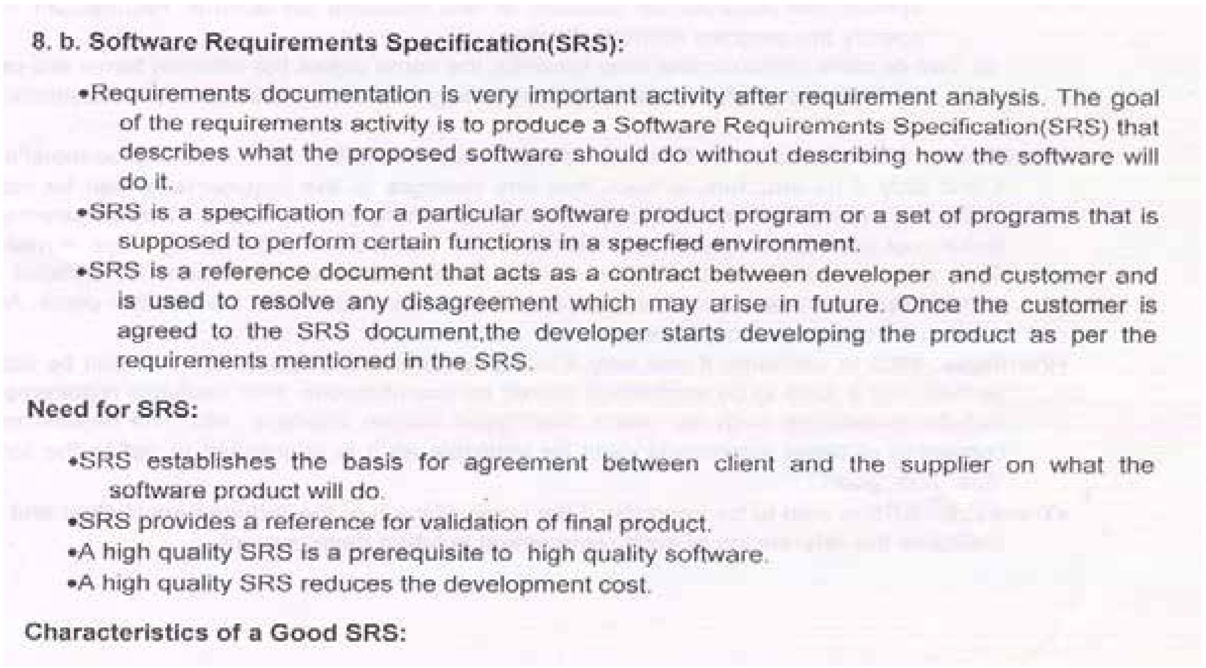
sure that the new features to be added do indeed work. However, as modifications have be made to an existing system, testing also has to be done to make sure that the modification has not had any undesired side effect of making some of the earlier services faulty. That is, besides ensuring the desired behavior of the new services, testing has to ensure that the desired behavior of the old services is maintained. This is the task of regression testing.

For regression testing, some test cases that have been executed on the old system are maintained, along with the output produced by the old system. These test cases are executed again on the modified system and its output compared with the earlier output to make sure that the system is working as before on these test cases. This frequently is a major task when modifications are to be made to existing systems.

A consequence of this is that the test cases for system should be properly documented for future use in regression testing. In fact, for many systems that are frequently changed, regression testing scripts are used, which automate performing regression testing after some changes. A regression testing script executes a suite of test cases. For each test case, it sets the system state for testing, executes the test case, determines system state or output against expected values. These scripts are typically produced during system state testing, as regression testing is generally done only for complete systems. When the system is modified, the scripts are executed again, giving the inputs specified in the scripts and comparing the outputs with the outputs given in the scripts. Given the scripts, through the use of tools, regression testing can be largely automated.

b. Explain the term "Software Inspection". List major advantages of inspection over testing.

Answer:



8. b. Software Requirements Specification(SRS):

- •Requirements documentation is very important activity after requirement analysis. The goal of the requirements activity is to produce a Software Requirements Specification(SRS) that describes what the proposed software should do without describing how the software will do it.
- •SRS is a specification for a particular software product program or a set of programs that is supposed to perform certain functions in a specfied environment.
- •SRS is a reference document that acts as a contract between developer and customer and is used to resolve any disagreement which may arise in future. Once the customer is agreed to the SRS document,the developer starts developing the product as per the requirements mentioned in the SRS.

Need for SRS:

- •SRS establishes the basis for agreement between client and the supplier on what the software product will do.
- •SRS provides a reference for validation of final product.
- •A high quality SRS is a prerequisite to high quality software.
- •A high quality SRS reduces the development cost.

Characteristics of a Good SRS:

- **Correctness:** SRS is said to be correct if and only if every requirement stated in it is met by software e.g. if a software is supposed to perform a particular task at a particular instance of time and it is responding at some other instance of time then the requirements is incorrect.
- **Completeness:** SRS is said to be complete if and only if it constitutes the following elements:
  - All important requirements relating to functionality performance, design constraints and external interfaces.
  - Responses/Output to both valid and invalid input values i.e. it should include all **realizable** classes of input data in all realizable classes of situations.
- **Unambiguousness:** SRS is said to be unambiguous if and only if every requirement stated in it has only one interpretation. Each written sentence in the SRS should have a unique interpretation.SRS should be unambiguous for both author and user irrespective of their technical background. Requirements are generally written in natural language such as English. Natural language is itself ambiguous. Natural language SRS should be reviewed by an independent party for identification of ambiguous use of language so that it can be corrected. This can be avoided by using particular requirement specification language whose language processor can automatically detect lexical,syntactic and semantic errors.
- **Consistent:** SRS is consistent if and only if individual requirements documented in it don't have any conflict.

  Types of Conflicts:

  i. The specified characteristics if objects may conflict e.g. the format of an output report may be tabular in one requirement and textual in another requirement.
  ii. There may be logical conflict between two specified actions e.g. one requirement may specify the program for addition of two numbers an another requirement may specify the program for multiplication.
  iii. Two or more requirements may describe the same object but different terms are used for that object. Use of standard terminology and definitions ensures consistency.

- **Modifiable:** SRS should be well structured and easily modifiable. SRS is said to be modifiable if and only if its structure is such that any changes to the requirements can be made easily,completely and consistently without disturbing the structure. The requirements should not be redundant as it can lead to errors. Sometimes redundancy helps in making SRS more readable but problem arises when redundant document is updated as requirements may be altered at one place but remains as it is at some other place. As a result SRS becomes inconsistent.
- **Verifiable:** SRS is verifiable if and only if every requirement documented in it can be easily verified. For a SRS to be verifiable,it should be unambiguous. Non verifiable requirements include statements such as "works fine","good human interface" etc. The requirements consisting of these ststements can't be verifiable as it is impossible to define the terms "fine" and "good".
- **Traceable:** SRS is said to be traceable if the origin of each of the requirement is clear and if it facilitates the referencing of each requirement in future development.

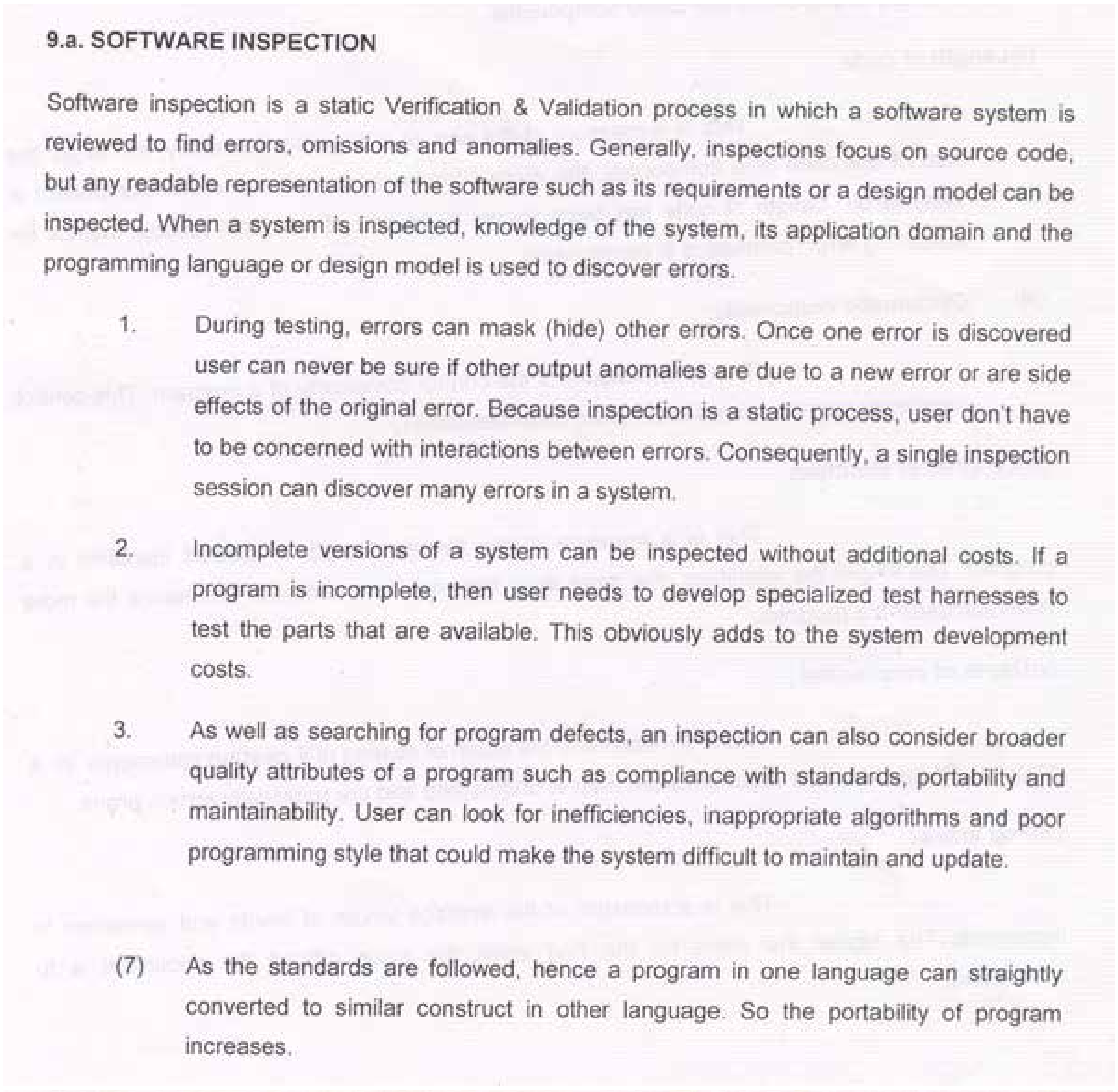
Types of Traceability:

i. **Backward** Traceability: It includes every requirement explicitly referencing its source in previous document.
ii. **Forward** Traceability: It includes each requirement in SRS having a unique name or reference number.

- **Maintainability,Portability and Adaptability:**
  i. Maintainability is the **collection** of qualities which deals with how much easily the system can be changed.
  ii. Portability: ensures using a system under new operating system or a new hardware platform.
  iii. Adaptability: flexibility of the system to meet with predictable new requirements e.g. a payroll program should be adaptable to yearly changes in tax structure and Dearness allowance.

Q.9 a. Define and explain Release Management. Explain various factors that influence system release strategy.

Answer:



9.a. SOFTWARE INSPECTION

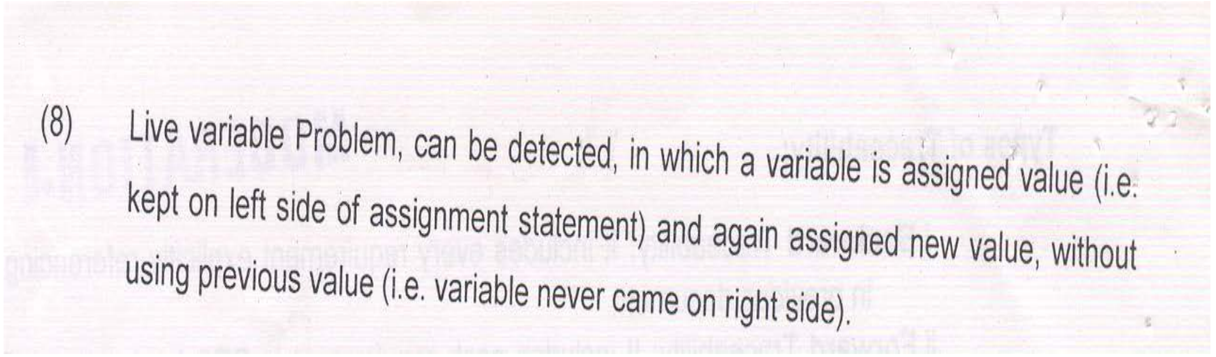
Software inspection is a static Verification & Validation process in which a software system is reviewed to find errors, omissions and anomalies. Generally, inspections focus on source code, but any readable representation of the software such as its requirements or a design model can be inspected. When a system is inspected, knowledge of the system, its application domain and the programming language or design model is used to discover errors.

1. During testing, errors can mask (hide) other errors. Once one error is discovered user can never be sure if other output anomalies are due to a new error or are side effects of the original error. Because inspection is a static process, user don't have to be concerned with interactions between errors. Consequently, a single inspection session can discover many errors in a system.

2. Incomplete versions of a system can be inspected without additional costs. If a program is incomplete, then user needs to develop specialized test harnesses to test the parts that are available. This obviously adds to the system development costs.

3. As well as searching for program defects, an inspection can also consider broader quality attributes of a program such as compliance with standards, portability and maintainability. User can look for inefficiencies, inappropriate algorithms and poor programming style that could make the system difficult to maintain and update.

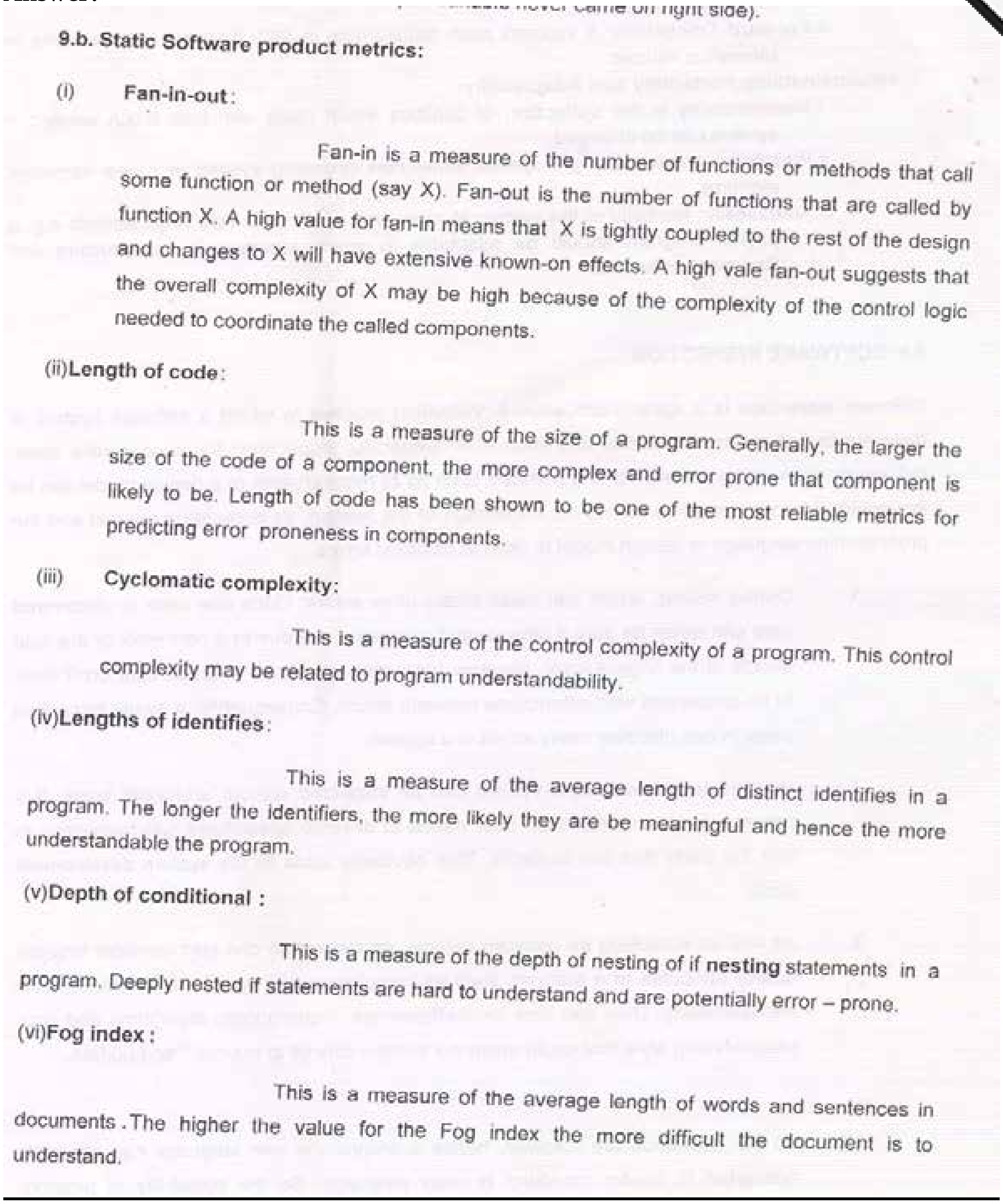
(7) As the standards are followed, hence a program in one language can straightly converted to similar construct in other language. So the portability of program increases.



(8) Live variable Problem, can be detected, in which a variable is assigned value (i.e. kept on left side of assignment statement) and again assigned new value, without using previous value (i.e. variable never came on right side).

b. Define and explain various static software product metrics.

Answer:



never came on right side).

9.b. Static Software product metrics:

(i) **Fan-in-out**:

Fan-in is a measure of the number of functions or methods that call some function or method (say X). Fan-out is the number of functions that are called by function X. A high value for fan-in means that X is tightly coupled to the rest of the design and changes to X will have extensive known-on effects. A high vale fan-out suggests that the overall complexity of X may be high because of the complexity of the control logic needed to coordinate the called components.

(ii)**Length of code**:

This is a measure of the size of a program. Generally, the larger the size of the code of a component, the more complex and error prone that component is likely to be. Length of code has been shown to be one of the most reliable metrics for predicting error proneness in components.

(iii) **Cyclomatic complexity:**

This is a measure of the control complexity of a program. This control complexity may be related to program understandability.

(iv)**Lengths of identifies**:

This is a measure of the average length of distinct identifies in a program. The longer the identifiers, the more likely they are be meaningful and hence the more understandable the program.

(v)**Depth of conditional :**

This is a measure of the depth of nesting of if **nesting** statements in a program. Deeply nested if statements are hard to understand and are potentially error – prone.

(vi)**Fog index :**

This is a measure of the average length of words and sentences in documents .The higher the value for the Fog index the more difficult the document is to understand.

<u>**TEXT BOOK**</u>

Software Engineering, Ian Sommerville, 7th edition, Pearson Education, 2004