Q.2 a. What do you mean by system analysis and design? Discuss. (8)

Answer:

Systems development can generally be thought of as having two major components: Systems analysis and Systems design. System design is the process of planning a new business system or one to replace or complement an existing system. But before this planning can be done, we must thoroughly understand the old system and determine how computers can best be used to make its operation more effective. System analysis, then, is the process of gathering and interpreting facts, diagnosing problems, and using the information to recommend improvements to the system. This is the job of the systems analyst.

Systems analysts do more than solve current problems. They are frequently called upon to help handle the planned expansion of a business. In the case of the clothing store, the systems study is future oriented, since no system currently exists. Analysts assess as carefully as possible what the future needs of the business will be and what changes should be considered to meet these needs. In this instance and in most others, analysts may recommend alternatives for improving the situation. Usually more than one strategy is possible. Working with managers and employees in the organization, systems analysts recommend which alternative to adopt, based on such concerns as the suitability of the solution to the particular organization and setting, as well as the employee support the solution is likely to have. Sometimes the time required to develop one alternative, compared with others, is the most critical issue. Costs and benefits are also important determinants. In the end, management, which will pay for and use the result, actually decides which alternative to accept. Once this decision is made, a plan is developed to implement the recommendation. The plan includes all systems design features, such as new data capture needs, file specifications, operating procedures, equipment and personnel needs. The systems design is like the blueprint for a building: it specifies all the features that are to be in the finished product.

2 marks for defining SAD and 6 marks for discussions.

b. What is meant by a System? What are the characteristics of a System? What are the elements of a System? Explain in detail. (8)

Answer:

The term system is derived form the Greek word systema, which means an organized relationship among functioning units or components. A system exists because it is designed to achieve one or more objectives. We come into daily contact with the transportation system, the telephone system, the accounting system, the production system, and, for over two decades, the computer system. Similarly, we talk of the business system and of the organization as a system consisting of interrelated departments (subsystems) such as production, sales, personnel, and an information system. None of these subsystems is of much use as a single, independent unit. When they are properly coordinated, however, the firm can function effectively and profitably. There are more than a hundred definitions of the word system, but most seem to have a common thread that suggests that a system is an orderly grouping of interdependent components linked together according to a plan to achieve a specific objective. The word component may refer to physical parts (engines, wings of aircraft, car), managerial steps (planning, organizing and controlling), or a system in a multi level structure. The component may be simple or complex, basic or advanced. They may be single computer with a keyboard, memory, and printer or a series of intelligent terminals linked to a mainframe. In either case, each component is part of the total system and has to do its share of work for the system to achieve the intended goal. This orientation requires an orderly grouping of the components for the design of a successful system.

The study of systems concepts, then, has three basic implications:
1. A system must be designed to achieve a predetermined objective.
2. Interrelationships and interdependence must exist among the components.
3. The objectives of the organization as a whole have a higher priority than the objectives of its subsystems.

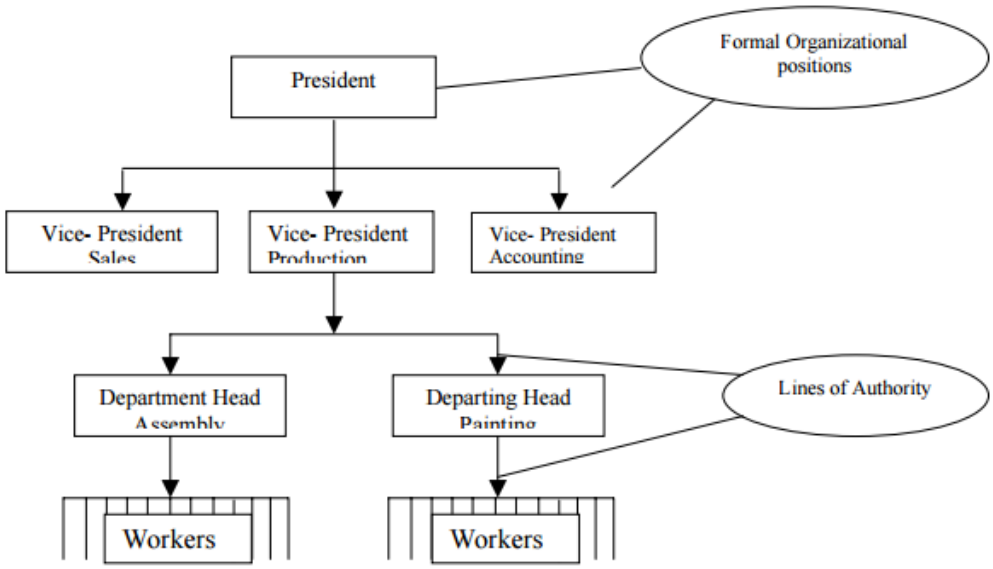
Characteristics of a System

Our definition of a system suggests some characteristics that are present in all systems: organization (order), interaction, interdependence, integration and a central objective.

i. **Organization**: Organization implies structure and order. It is the arrangement of components that helps to achieve objectives. In the design of a business system, for example, the hierarchical relationships starting with the president on top and leading downward to the blue – collar workers represents the organization structure. Such an arrangement portrays a system – subsystem relationship, defines the authority structure, specifies the formal flow of communication and formalizes the chain of command. Like – wise, a computer system is designed around an input device, a central processing unit, an output device and one or more storage units. When linked together they work as a whole system for producing information.

ii. **Interaction:** Interaction refers to the manner in which each component functions with other components of the system. In an organization, for example, purchasing must interact with production, advertising with sales and payroll with personnel. In a computer system, the central processing unit must interact with the input device to solve a problem. In turn, the main memory holds programs and data that the arithmetic unit uses for computation. The interrelationship between these components enables the computer to perform.

iii. **Interdependence:** Interdependence means that parts of the organization or computer system depend on one another. They are coordinated and linked together according to a plan. One subsystem depends on the input of another subsystem for proper functioning: that is, the output of one subsystem is the required input for another subsystem. This interdependence is crucial in systems work. An integrated information system is designed to serve the needs of authorized users (department heads, managers, etc.) for quick access and retrieval via remote terminals. The interdependence between the personnel subsystem and the organization's users is obvious. In summary, no subsystem can function in isolation because it is dependent on the data (inputs) it receives from other subsystems to perform its required tasks. Figure 1: Organization Structure



iv. **Integration:** Integration refers to the holism of systems. Synthesis follows analysis to achieve the central objective of the organization. Integration is concerned with how a system is tied together. It is more than sharing a physical part or location. It means that parts of the system work together within the system even though each part performs a unique function. Successful integration will typically produce a synergistic effect and greater total impact than if each component works separately.

v. **Central objective**: The last characteristic of a system is its central objective. Objectives may be real or stated. Although a stated objective may be the real objective, it is not uncommon for an organization to state one objective and operate to achieve another. The important point is that users must know the central objective of a computer application early in the Formal Organizational positions President Vice- President Sales Vice- President Production Vice- President Accounting Department Head Assembly Departing Head Painting Lines of Authority Workers Workers analysis for a successful design and conversion. Political as well as organizational considerations often cloud the real objective. This means that the analyst must work around such obstacles to identify the real objective of the proposed change.

Elements of a System

In most cases, systems analysts operate in a dynamic environment where change is a way of life. The environment may be a business firm, a business application, or a computer system. To reconstruct a system, the following key elements must be considered:

1. Outputs and inputs.
2. Processor(s).
3. Control.
4. Feedback.
5. Environment.
6. Boundaries and interface.

2marks for defining system,
4 marks for characteristics of a system,
2 marks for element of system.

Q.3 a. Explain the concept of Object Oriented Analysis and Design. What are its purposes? What is Object Modeling? (8)

Answer:

Object-oriented analysis and design (OOAD) is a popular technical approach to analyzing, designing an application, system, or business by applying the object-oriented paradigm and visual modeling throughout the development life cycles to foster better stakeholder communication and product quality. According to the popular guide Unified Process, OOAD in modern software engineering is best conducted in an iterative and incremental way. Iteration by iteration, the outputs of OOAD activities, analysis models for OOA and design models for OOD respectively, will be refined and evolve continuously driven by key factors like risks and business value.

The purpose of any analysis activity in the software life-cycle is to create a model of the system's functional requirements that is independent of implementation constraints.

The main difference between object-oriented analysis and other forms of analysis is that by the object-oriented approach we organize requirements around objects, which integrate both behaviors (processes) and states (data) modeled after real world objects that the system interacts with. In other or traditional analysis methodologies, the two aspects: processes and data are considered separately. For example, data may be modeled by ER diagrams, and behaviors by flow charts or structure charts.

The primary tasks in object-oriented analysis (OOA) are:

- Find the objects
- Organize the objects
- Describe how the objects interact
- Define the behavior of the objects
- Define the internals of the objects

Common models used in OOA are use cases and object models. Use cases describe scenarios for standard domain functions that the system must accomplish. Object models describe the names, class relations (e.g. Circle is a subclass of Shape), operations, and properties of the main objects. User-interface mockups or prototypes can also be created to help understanding.

During object-oriented design (OOD), a developer applies implementation constraints to the conceptual model produced in object-oriented analysis. Such constraints could include the hardware and software platforms, the performance requirements, persistent storage and transaction, usability of the system, and limitations imposed by budgets and time. Concepts in the analysis model which is technology independent, are mapped onto implementing classes and interfaces resulting in a model of the solution domain, i.e., a detailed description of how the system is to be built on concrete technologies.

Object-oriented modeling (OOM) is a common approach to modeling applications, systems, and business domains by using the object-oriented paradigm throughout the entire development life cycles. OOM is a main technique heavily used by both OOA and OOD activities in modern software engineering.

Object-oriented modeling typically divides into two aspects of work: the modeling of dynamic behaviors like business processes and use cases, and the modeling of static structures like classes and components. OOA and OOD are the two distinct abstract levels (i.e. the analysis level and the design level) during OOM. The Unified Modeling Language (UML) and SysML are the two popular international standard languages used for object-oriented modeling.

The benefits of OOM are:

i. **Efficient and effective communication:** Users typically have difficulties in understanding comprehensive documents and programming language codes well. Visual model diagrams can be more understandable and can allow users and stakeholders to give developers feedback on the appropriate requirements and structure of the system. A key goal of the object-oriented approach is to decrease the "semantic gap" between the system and the real world, and to have the system be constructed using terminology that is almost the same as the stakeholders use in everyday business. Object-oriented modeling is an essential tool to facilitate this.

ii. **Useful and stable abstraction:** Modeling helps coding. A goal of most modern software methodologies is to first address "what" questions and then address "how" questions, i.e. first determine the functionality the system is to provide without consideration of implementation constraints, and then consider how to make specific solutions to these abstract requirements, and refine them into detailed designs and codes by constraints such as technology and budget. Object-oriented modeling enables this by producing abstract and accessible descriptions of both system requirements and designs, i.e. models that define their essential structures and behaviors like processes and objects, which are important and valuable development assets with higher abstraction levels above concrete and complex source code.

4 marks for OOAD
4 marks for object modeling

b. What are the essential elements of a design pattern? (8)

Answer:

In general, a pattern has four essential elements:

i. **The pattern name** is a handle we can use to describe a design problem, its solutions, and consequences in a word or two. Naming a pattern immediately increases our design vocabulary. It lets us design at a higher level of abstraction. Having a vocabulary for patterns lets us talk about them with our colleagues, in our documentation, and even to ourselves. It makes it easier to think about designs and to communicate them and their trade-offs to others. Finding good names has been one of the hardest parts of developing our catalog.

ii. **The problem** describes when to apply the pattern. It explains the problem and its context. It might describe specific design problems such as how to represent algorithms as objects. It might describe class or object structures that are symptomatic of an inflexible design. Sometimes the problem will include a list of conditions that must be met before it makes sense to apply the pattern.

iii. **The solution** describes the elements that make up the design, their relationships, responsibilities, and collaborations. The solution doesn't describe a particular concrete design or implementation, because a pattern is like a template that can be applied in many different situations. Instead, the pattern provides an abstract description of a design problem and how a general arrangement of elements (classes and objects in our case) solves it.

iv. **The consequences** are the results and trade-offs of applying the pattern. Though consequences are often unvoiced when we describe design decisions, they are critical for evaluating design alternatives and for understanding the costs and benefits of applying the pattern. The consequences for software often concern space and time trade-offs. They may address language and implementation issues as well. Since reuse is often a factor in object-oriented design, the consequences of a pattern include its impact on a system's flexibility, extensibility, or portability. Listing these consequences explicitly helps you understand and evaluate them.

A design pattern names, abstracts, and identifies the key aspects of a common design structure that make it useful for creating a reusable object-oriented design. The design pattern identifies the participating classes and instances, their roles and collaborations, and the distribution of responsibilities. Each design pattern Design Patterns: Elements of Reusable Object-Oriented Software focuses on a particular object-oriented design problem or issue. It describes when it applies, whether it can be applied in view of other design constraints, and the consequences and trade-offs of its use.

2 marks for each point

Q.4 a. Define Data Modeling. Explain the process of Logical Data Modeling and Physical Data Modeling. (8)

Answer:

Data modeling is a process used to define and analyze data requirements needed to support the business processes within the scope of corresponding information systems in organizations. Therefore, the process of data modeling involves professional data modellers working closely with business stakeholders, as well as potential users of the information system.

According to Steve Hoberman, data modeling is the process of learning about the data, and the data model is the end result of the data modeling process.[2]

There are three different types of data models produced while progressing from requirements to the actual database to be used for the information system.[3] The data requirements are initially recorded as a conceptual data model which is essentially a set of technology independent specifications about the data and is used to discuss initial requirements with the business stakeholders. The conceptual model is then translated into a logical data model, which documents structures of the data that can be implemented in databases. Implementation of one conceptual data model may require multiple logical data models. The last step in data modeling is transforming the logical data model to aphysical data model that organizes the data into tables, and accounts for access, performance and storage details. Data modeling defines not just data elements, but also their structures and the relationships between them.[4]

Data modeling techniques and methodologies are used to model data in a standard, consistent, predictable manner in order to manage it as a resource. The use of data modeling standards is strongly recommended for all projects requiring a standard means of defining and analyzing data within an organization, e.g., using data modeling:

- to assist business analysts, programmers, testers, manual writers, IT package selectors, engineers, managers, related organizations and clients to understand and use an agreed semi-formal model the concepts of the organization and how they relate to one another
- to manage data as a resource
- for the integration of information systems
- for designing databases/data warehouses (aka data repositories)

Data modeling may be performed during various types of projects and in multiple phases of projects. Data models are progressive; there is no such thing as the final data model for a business or application. Instead a data model should be considered a living document that will change in response to a changing business. The data models should ideally be stored in a repository so that they can be retrieved, expanded, and edited over time. Whitten et al. (2004) determined two types of data modeling:[5]

- Strategic data modeling: This is part of the creation of an information systems strategy, which defines an overall vision and architecture for information systems is defined.Information engineering is a methodology that embraces this approach.
- Data modeling during systems analysis: In systems analysis logical data models are created as part of the development of new databases.

Data modeling is also used as a technique for detailing business requirements for specific databases. It is sometimes called database modeling because a data model is eventually implemented in a database.

Logical data models represent the abstract structure of a domain of information. They are often diagrammatic in nature and are most typically used in business processes that seek to capture things of importance to an organization and how they relate to one another. Once validated and approved, the logical data model can become the basis of a physical data model and inform the design of a database.

Logical data models should be based on the structures identified in a preceding conceptual data model, since this describes the semantics of the information context, which the logical model should also reflect. Even so, since the logical data model anticipates implementation on a specific computing system, the content of the logical data model is adjusted to achieve certain efficiencies.

The term 'Logical Data Model' is sometimes used as a synonym of 'domain model' or as an alternative to the domain model. While the two concepts are closely related, and have overlapping goals, a domain

model is more focused on capturing the concepts in the problem domain rather than the structure of the data associated with that domain.

A physical data model (or database design) is a representation of a data design which takes into account the facilities and constraints of a given database management system. In the lifecycle of a project it typically derives from a logical data model, though it may be reverse-engineered from a given database implementation. A complete physical data model will include all the database artifacts required to create relationships between tables or to achieve performance goals, such asindexes, constraint definitions, linking tables, partitioned tables or clusters. Analysts can usually use a physical data model to calculate storage estimates; it may include specific storage allocation details for a given database system.

2 marks for defining Data Modeling
3 marks for Logical Data Modeling
3 marks for Physical Data Modeling

b. Give an overview of Requirement Analysis phase. What are the issues associated with it? (8)

Answer:

Requirements analysis in systems engineering and software engineering, encompasses those tasks that go into determining the needs or conditions to meet for a new or altered product or project, taking account of the possibly conflicting requirements of the various stakeholders, analyzing, documenting, validating and managing software or system requirements.

Requirements analysis is critical to the success of a systems or software project. The requirements should be documented, actionable, measurable, testable, traceable, related to identified business needs or opportunities, and defined to a level of detail sufficient for system design.

Conceptually, requirements analysis includes three types of activities:

- Eliciting requirements:(e.g. the project charter or definition), business process documentation, and stakeholder interviews. This is sometimes also called requirements gathering.
- Analyzing requirements: determining whether the stated requirements are clear, complete, consistent and unambiguous, and resolving any apparent conflicts.
- Recording requirements: Requirements may be documented in various forms, usually including a summary list and may include natural-language documents, use cases, user stories, or process specifications.

Requirements analysis can be a long and tiring process during which many delicate psychological skills are involved. New systems change the environment and relationships between people, so it is important to identify all the stakeholders, take into account all their needs and ensure they understand the implications of the new systems. Analysts can employ several techniques to elicit the requirements from the customer. These may include the development of scenarios (represented as user stories in agile methods), the identification of use cases, the use of workplace observation or ethnography, holding interviews, or focus groups (more aptly named in this context as requirements workshops, or requirements review sessions) and creating requirements lists. Prototyping may be used to develop an example system that can be demonstrated to stakeholders. Where necessary, the analyst will employ a combination of these methods to establish the exact requirements of the stakeholders, so that a system that meets the business needs is produced.[citation needed] Requirements quality can be improved through these and other methods

- Visualization. Using tools that promote better understanding of the desired end-product such as visualization and simulation.
- Consistent use of templates. Producing a consistent set of models and templates to document the requirements.

- Documenting dependencies. Documenting dependencies and interrelationships among requirements, as well as any assumptions and congregations.

Requirements analysis issues

i. Stakeholder issues

Steve McConnell, in his book Rapid Development, details a number of ways users can inhibit requirements gathering:

- Users do not understand what they want or users don't have a clear idea of their requirements
- Users will not commit to a set of written requirements
- Users insist on new requirements after the cost and schedule have been fixed
- Communication with users is slow
- Users often do not participate in reviews or are incapable of doing so
- Users are technically unsophisticated
- Users do not understand the development process
- Users do not know about present technology

This may lead to the situation where user requirements keep changing even when system or product development has been started.

ii. **Engineer/developer issues**

Possible problems caused by engineers and developers during requirements analysis are:

- Engineer/developer starts coding/implementation immediately before they really understand the whole requirement from analyst, which usually causes lots of defect fixing or reworking in test/verification phase.
- Technical personnel and end-users may have different vocabularies. Consequently, they may wrongly believe they are in perfect agreement until the finished product is supplied.
- Engineers and developers may try to make the requirements fit an existing system or model, rather than develop a system specific to the needs of the client.
- Analysis may often be carried out by engineers or programmers, rather than personnel with the domain knowledge to understand a client's needs properly.

iii. Attempted solutions

One attempted solution to communications problems has been to employ specialists in business or system analysis.

Techniques introduced in the 1990s like prototyping, Unified Modeling Language (UML), use cases, and Agile software development are also intended as solutions to problems encountered with previous methods.

Also, a new class of application simulation or application definition tools have entered the market. These tools are designed to bridge the communication gap between business users and the IT organization — and also to allow applications to be 'test marketed' before any code is produced. The best of these tools offer:

- electronic whiteboards to sketch application flows and test alternatives
- ability to capture business logic and data needs
- ability to generate high fidelity prototypes that closely imitate the final application
- interactivity
- capability to add contextual requirements and other comments
- ability for remote and distributed users to run and interact with the simulation

4 marks for explaining Requirement Analysis Phase
2+2 marks for explaining the issues

Q.5 Write short notes on the following: (8+8)

(i) Model Driven Approach

(ii) Rapid Application Development

Answer:

a. MDA approach

One of the main aims of the MDA is to separate design from architecture. As the concepts and technologies used to realize designs and the concepts and technologies used to realize architectures have changed at their own pace, decoupling them allows system developers to choose from the best and most fitting in both domains. The design addresses the functional (use case) requirements while architecture provides the infrastructure through which non-functional requirements like scalability, reliability and performance are realized. MDA envisages that the platform independent model (PIM), which represents a conceptual design realizing the functional requirements, will survive changes in realization technologies and software architectures.

MDA tools

Basically, an MDA tool is a tool used to develop, interpret, compare, align, measure, verify, transform, etc. models or metamodels. In the following section "model" is interpreted as meaning any kind of model (e.g. a UML model) or metamodel (e.g. the CWM metamodel). In any MDA approach we have essentially two kinds of models: initial models are created manually by human agents while derived models are created automatically by programs. For example an analyst may create a UML initial model from its observation of some loose business situation while a Java model may be automatically derived from this UML model by a Model transformation operation.

An MDA tool may be one or more of the following types:

Creation Tool

A tool used to elicit initial models and/or edit derived models.

Analysis Tool

A tool used to check models for completeness, inconsistencies, or error and warning conditions. Also used to calculate metrics for the model.

Transformation Tool

A tool used to transform models into other models or into code and documentation.

Composition Tool

A tool used to compose (i.e. to merge according to a given composition semantics) several source models, preferably conforming to the same metamodel.

Test Tool

A tool used to "test" models as described in Model-based testing.

Simulation Tool

A tool used to simulate the execution of a system represented by a given model. This is related to the subject of model execution.

Metadata Management Tool

A tool intended to handle the general relations between different models, including the metadata on each model (e.g. author, date of creation or modification, method of creation (which tool? which transformation? etc.)) and the mutual relations between these models (i.e. one metamodel is a version of another one, one model has been derived from another one by a transformation, etc.)

Reverse Engineering Tool

A tool intended to transform particular legacy or information artifact portfolios into full-fledged models.

Some tools perform more than one of the functions listed above. For example, some creation tools may also have transformation and test capabilities. There are other tools that are solely for creation, solely for graphical presentation, solely for transformation, etc.

One of the characteristics of MDA tools is that they mainly take models (e.g. MOF models or metamodels) as input and generate models as output[citation needed]. In some cases however the

parameters may be taken outside the MDA space like in model to text or text to model transformation tools.

b. **Rapid application development** (**RAD**) is both a general term used to refer to alternatives to the conventional waterfall model of software development as well as the name for James Martin's approach to rapid development. In general, RAD approaches to software development put less emphasis on planning tasks and more emphasis on development. In contrast to the waterfall model, which emphasizes rigorous specification and planning, RAD approaches emphasize the necessity of adjusting requirements in reaction to knowledge gained as the project progresses. This causes RAD to use prototypes in addition to or even sometimes in place of design specifications. RAD approaches also emphasize a flexible process that can adapt as the project evolves rather than rigorously defining specifications and plans correctly from the start. In addition to James Martin's RAD methodology, other approaches to rapid development include Agile methods and the spiral model. RAD is especially well suited (although not limited to) developing software that is driven by user interface requirements. Graphical user interface builders are often called rapid application development tools.

The advantages of RAD include:

- Better Quality. By having users interact with evolving prototypes the business functionality from a RAD project can often be much higher than that achieved via a waterfall model. The software can be more usable and has a better chance to focus on business problems that are critical to end users rather than technical problems of interest to developers.
- Risk Control. Although much of the literature on RAD focuses on speed and user involvement a critical feature of RAD done correctly is risk mitigation. It's worth remembering that Boehm initially characterized the spiral model as a risk based approach. A RAD approach can focus in early on the key risk factors and adjust to them based on empirical evidence collected in the early part of the process. E.g., the complexity of prototyping some of the most complex parts of the system.
- More projects completed on time and within budget. By focusing on the development of incremental units the chances for catastrophic failures that have dogged large waterfall projects is reduced. In the Waterfall model it was common to come to a realization after six months or more of analysis and development that required a radical rethinking of the entire system. With RAD this kind of information can be discovered and acted upon earlier in the process.

The disadvantages of RAD include:

- The risk of a new approach. For most IT shops RAD was a new approach that required experienced professionals to rethink the way they worked. Humans are virtually always averse to change and any project undertaken with new tools or methods will be more likely to fail the first time simply due to the requirement for the team to learn.
- Requires time of scarce resources. One thing virtually all approaches to RAD have in common is that there is much more interaction throughout the entire life-cycle between users and developers. In the waterfall model, users would define requirements and then mostly go away as developers created the system. In RAD users are involved from the beginning and through virtually the entire project. This requires that the business is willing to invest the time of application domain experts. The paradox is that the better the expert, the more they are familiar with their domain, the more they are required to actually run the business and it may be difficult to convince their supervisors to invest their time. Without such commitments RAD projects will not succeed.
- Less control. One of the advantages of RAD is that it provides a flexible adaptable process. The ideal is to be able to adapt quickly to both problems and opportunities. There is an inevitable trade-off between flexibility and control, more of one means less of the other. If a project (e.g. life-critical software) values control more than agility RAD is not appropriate.
- Poor design. The focus on prototypes can be taken too far in some cases resulting in a "hack and test" methodology where developers are constantly making minor changes to individual

components and ignoring system architecture issues that could result in a better overall design. This can especially be an issue for methodologies such as Martin's that focus so heavily on the User Interface of the system.

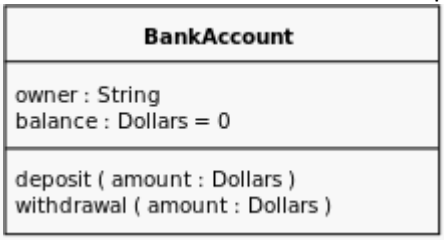
- Very large systems. RAD typically focuses on small to medium-sized project teams. The other issues cited above (less design and control) present special challenges when using a RAD approach for very large scale systems

4 marks for MDA approach
4 marks for RDA approach

Q.6 a. What is the role of Class Diagram in Object Oriented Modelling? (8)

Answer:

The class diagram is the main building block of object oriented modelling. It is used both for general conceptual modelling of the systematics of the application, and for detailed modelling translating the models into programming code. Class diagrams can also be used for data modeling. The classes in a class diagram represent both the main objects, interactions in the application and the classes to be programmed.



A class with three sections.

In the diagram, classes are represented with boxes which contain three parts:

- The top part contains the name of the class. It is printed in bold and centered, and the first letter is capitalized.
- The middle part contains the attributes of the class. They are left-aligned and the first letter is lowercase.
- The bottom part contains the methods the class can execute. They are also left-aligned and the first letter is lowercase.

In the design of a system, a number of classes are identified and grouped together in a class diagram which helps to determine the static relations between those objects. With detailed modelling, the classes of the conceptual design are often split into a number of subclasses.

In order to further describe the behaviour of systems, these class diagrams can be complemented by a state diagram or UML state machine.

UML provides mechanisms to represent class members, such as attributes and methods, and additional information about them.

To specify the visibility of a class member (i.e., any attribute or method), these notations must be placed before the member's name:

+	Public
-	Private
#	Protected
/	Derived (can be combined with one of the others)
~	Package

The UML specifies two types of scope for members: instance and classifier and these last are represented by underlined names.

- Classifier members are commonly recognized as "static" in many programming languages. The scope is the class itself.
  - Attribute values are equal for all instances
  - Method invocation does not affect the instance's state
- Instance members are scoped to a specific instance.
  - Attribute values may vary between instances
  - Method invocation may affect the instance's state (i.e., change instance's attributes)

To indicate a classifier scope for a member, its name must be underlined. Otherwise, instance scope is assumed by default.

4 marks for explaining without example
Full marks for explaining with example

b. Explain the role of Implementation phase in System Development. (8)

Answer:

Implementation Phase

The implementation phase is less creative than system design. It is primarily concerned with user training site preparation, and file conversion. When the candidate system is linked to terminals or remote sites, the telecommunication network and tests of the network along with the system are also included under implementation. During the final testing, user acceptance is tested, followed by user training. Depending on the nature of the system, extensive user training may be required. Conversion usually takes place at about the same time the user is being trained or later. In the extreme, the programmer is falsely viewed as someone who ought to be isolated from other aspects of system development. Programming is itself design work, however. The initial parameters of the candidate system should be modified as a result of programming efforts. Programming provides a " reality test" for the assumptions made by the analyst. It is therefore a mistake to exclude programmers from the initial system design.

System testing checks the readiness and accuracy of the system to access, update and retrieve data from new files. Once the programs become available, test data are read into the computer and processed against the file(s) provided for testing. If successful, the program(s) is then run with " live" data. Otherwise, a diagnostic procedure is used to locate and correct errors in the program. In most conversions, parallel run is conducted where the new system runs simultaneously with the "old" system. This method, though costly, provides added assurance against errors in the candidate system and also gives the user staff an opportunity to gain experience through operation. In some cases, however, parallel processing in not practical. For example, it is not plausible to run parallel two online point-of-sale (POS) systems for a retail chain. In any case, after the candidate system proves itself, the old system is phased out.

Marks to be awarded based on explanation (Answer is too subjective)

Q.7 a. What are the four types of UML diagrams? (8)

Answer:

UML Diagram is divided into four types:

i. Activity Diagrams - a generic flow chart used much in business modelling and sometimes in use case modelling to indicate the overall flow of the use case. This diagram type replaces the need for dataflow diagrams but is not a main diagram type for the purposes of analysis and design.

ii. State Machine Diagrams - in information systems these tend to be used to descibe the lifecycle of an important data entity. In real-time systems they tend to be used to describe state dependent behaviour

iii. Component Diagrams - show the types of components, their interfaces and dependencies in the software architecture that is the solution to the application being developed.

iv. Deployment Diagrams - show actual computing nodes, their communication relationships and the processes or components that run on them.

UML can be used to model a business, prior to automating it with computers. The same basic UML syntax is used; however, a number of new symbols are added, in order to make the diagrams more relevant to the business process world. A commonly-used set of these symbols is available in current versions of Rational Rose. The most commonly used UML extensions for web applications were developed by Jim Conallen. You can access his own website to learn more about them by following the link. These symbols are also available in current versions of Rational Rose. UML is designed to be extended in this way. Extensions to the syntax are created by adding 'stereotypes' to a model element. The stereotype creates a new model element from an existing one with an extended, user-defined meaning. User defined symbols, which replace the original UML symbol for the model element, can then be assigned to the stereotype. UML itself uses this mechanism, so it is important to know what stereotypes are predefined in UML in order not to clash with them when creating new ones.

4 marks for explanation
4 marks for diagrams

b. Write in brief:
(i) System Maintenance
(ii) System Evaluation (8)

Answer:

SYSTEM MAINTENANCE

Once the system is installed, the MIS job has just begun. Computer systems are constantly changing. Hardware upgrades occur continually, and commercial software tools may change every year. Users change jobs. Errors may exist in the system. The business changes, and management and users demand new information and expansions. All of these actions mean the system needs to be modified. The job of overseeing and making these modifications is called software maintenance. The pressures for change are so great that in most organizations today as much as 80 per cent of the MIS staff is devoted to modifying existing programs. These changes can be time consuming and difficult. Most major systems were created by teams of programmers and analysts over a long period. In order to make a change to a program, the programmer has to understand how the current program works. Because the program was written by many different people with varying styles, it can be hard to understand. Finally, when a programmer makes a minor change in one location, it can affect another area of the program, which can cause additional errors or necessitate more changes. One difficulty with software maintenance is that every time part of an application is modified, there is a risk of adding defects (bugs). Also, over time the application becomes less structured and more complex, making it harder to understand. These are some of the main reasons why the year 2000 alterations were so expensive and time consuming. At some point, a company may decide to replace or improve the heavily modified system. There are several techniques for improving an existing system, ranging from rewriting individual sections to restructuring the entire application.. The difference lies in scope-how much of the application needs to be modified. Older applications that were subject to modifications over several years tend to contain code that is no longer used, poorly documented changes, and inconsistent naming conventions. These applications are prime candidates for restructuring, during which the entire code is analyzed and reorganized to make it more efficient. More important, the code is organized, standardized, and documented to make it easier to make changes in the future.

SYSTEM EVALUATION

An important phase in any project is evaluating the resulting system. As part of this evaluation, it is also important to assess the effectiveness of the particular development process. There are several questions to ask. Were the initial cost estimates accurate? Was the project completed on time? Did users have sufficient input? Are maintenance costs higher than expected? Evaluation is a difficult issue. How can you as a manager tell the difference between a good system and a poor one? In some way, the

system should decrease costs, increase revenue, or provide a competitive advantage. Although these effects are important, they are often subtle and difficult to measure. The system should also be easy to use and flexible enough to adapt to changes in the business. If employees or customers continue to complain about a system, it should be re-examined. A system also needs to be reliable. It should be available when needed and should produce accurate output. Error detection can be provided in the system to recognize and avoid common problems. Similarly, some systems can be built to tolerate errors, so that when errors arise, the system recognizes the problem and works around it. For example, some computers exist today that automatically switch to backup components when one section fails, thereby exhibiting fault tolerance.

4 marks for system maintenance
4 marks for system evaluation

Q.8 a. What are the phases or processes in User Interface Design? (8)

Answer:

User interface design requires a good understanding of user needs. There are several phases and processes in the user interface design, some of which are more demanded upon than others, depending on the project.

Functionality requirements gathering – assembling a list of the functionality required by the system to accomplish the goals of the project and the potential needs of the users.

User and task analysis – a form of field research, it's the analysis of the potential users of the system by studying how they perform the tasks that the design must support, and conducting interviews to elucidate their goals. Typical questions involve:

What would the user want the system to do?

How would the system fit in with the user's normal workflow or daily activities?

How technically savvy is the user and what similar systems does the user already use?

What interface look & feel styles appeal to the user?

Information architecture – development of the process and/or information flow of the system (i.e. for phone tree systems, this would be an option tree flowchart and for web sites this would be a site flow that shows the hierarchy of the pages).

Prototyping – development of wire-frames, either in the form of paper prototypes or simple interactive screens. These prototypes are stripped of all look & feel elements and most content in order to concentrate on the interface.

Usability inspection – letting an evaluator inspect a user interface. This is generally considered to be cheaper to implement than usability testing, and can be used early on in the development process since it can be used to evaluate prototypes or specifications for the system, which usually can't be tested on users. Some common usability inspection methods include cognitive walkthrough, which focuses the simplicity to accomplish tasks with the system for new users, heuristic evaluation, in which a set of heuristics are used to identify usability problems in the UI design, and pluralistic walkthrough, in which a selected group of people step through a task scenario and discuss usability issues.

Usability testing – testing of the prototypes on an actual user—often using a technique called think aloud protocol where you ask the user to talk about their thoughts during the experience. User interface design testing allows the designer to understand the reception of the design from the viewer's standpoint, and thus facilitates creating successful applications.

Graphical user interface design – actual look and feel design of the final graphical user interface (GUI). It may be based on the findings developed during the user research, and refined to fix any usability problems found through the results of testing.

2 marks for each point with a maximum of 8 marks

b. What are the principles of User Interface Design? (8)

Answer:

The principles of user interface design are intended to improve the quality of user interface design. According to Larry Constantine and Lucy Lockwood in their usage-centered design, these principles are:

- The structure principle: Design should organize the user interface purposefully, in meaningful and useful ways based on clear, consistent models that are apparent and recognizable to users, putting related things together and separating unrelated things, differentiating dissimilar things and making similar things resemble one another. The structure principle is concerned with overall user interface architecture.
- The simplicity principle: The design should make simple, common tasks easy, communicating clearly and simply in the user's own language, and providing good shortcuts that are meaningfully related to longer procedures.
- The visibility principle: The design should make all needed options and materials for a given task visible without distracting the user with extraneous or redundant information. Good designs don't overwhelm users with alternatives or confuse with unneeded information.
- The feedback principle: The design should keep users informed of actions or interpretations, changes of state or condition, and errors or exceptions that are relevant and of interest to the user through clear, concise, and unambiguous language familiar to users.
- The tolerance principle: The design should be flexible and tolerant, reducing the cost of mistakes and misuse by allowing undoing and redoing, while also preventing errors wherever possible by tolerating varied inputs and sequences and by interpreting all reasonable actions.
- The reuse principle: The design should reuse internal and external components and behaviors, maintaining consistency with purpose rather than merely arbitrary consistency, thus reducing the need for users to rethink and remember.

According to Jef Raskin in his book The Humane Interface, there are two laws of user interface design, based on the fictional laws of robotics created by Isaac Asimov:

- First Law: A computer shall not harm your work or, through inactivity, allow your work to come to harm.
- Second Law: A computer shall not waste your time or require you to do more work than is strictly necessary.

Jef Raskin also mentions that "users should set the pace of an interaction," meaning that a user should not be kept waiting unnecessarily.

One mark for each point

Q.9 a. What are the four common system conversion strategies? (8)

Answer:

The conversion plan may include one of the following commonly used installation strategies:

- *Abrupt cut-over*—On a specific date (usually a date that coincides with an official business period such as month, quarter, or fiscal year), the old system is terminated and the new system is placed into operation. This is a high-risk approach because there may still be major problems that won't be uncovered until the system has been in operation for at least one business period. On the other hand, there are no transition costs. Abrupt cut-over may be necessary if, for instance, a government mandate or business policy becomes effective on a specific date and the system couldn't be implemented before that date.
- *Parallel conversion*—Under this approach, both the old and the new systems are operated for some time period. This ensures that all major problems in the new system have been solved before the old system is discarded. The final cut-over may be either abrupt (usually at the end of one business period) or gradual, as portions of the new system are deemed adequate. This strategy minimizes the risk of major flaws in the new system causing irreparable harm to the business; however, it also means the cost of running two systems over some period must be incurred. Because running two editions of the same system on the computer could place an unreasonable demand on computing resources, this may be possible only if the old system is largely manual.
- *Location conversion*—When the same system will be used in numerous geographical locations, it is usually converted at one location first (using either abrupt or parallel conversion). As soon as that site has approved the system, it can be farmed to the other sites. Other sites can be cut over abruptly because major errors have been fixed. Furthermore, other sites benefit from the learning experiences of the first test site. The first production test site is often called a *beta test site*.
- *Staged conversion*—Like location conversion, staged conversion is a variation on the abrupt and parallel conversions. A staged conversion is based on the version concept introduced earlier. Each successive version of the new system is converted as it is developed. Each version may be converted using the abrupt, parallel, or location strategy.

2 marks for explaining each conversion strategy

b. Explain the three levels of System Acceptance Test. (8)

Answer:

The conversion plan also typically includes a systems acceptance test plan. The systems acceptance test is the final opportunity for end users, management, and information systems operations management to accept or reject the system. A **systems acceptance test** is a final system test performed by end users using real data over an extended time period. It is an extensive test that addresses three levels of acceptance testing—verification testing, validation testing, and audit testing:

systems acceptance test a test performed on the final system wherein users conduct verification, validation, and audit tests.

- *Verification testing* runs the system in a simulated environment using simulated data. This simulated test is sometimes called *alpha testing*. The simulated test is primarily looking for errors and omissions

regarding end-user and design specifications that were specified in the earlier phases but not fulfilled during construction.

- *Validation testing* runs the system in a live environment using real data. This is sometimes called *beta testing*. During this validation, a number of items are tested:
  a. *Systems performance.* Is the throughput and response time for processing adequate to meet a normal processing workload? If not, some programs may have to be rewritten to improve efficiency or processing hardware may have to be replaced or upgraded to handle the additional workload.
  b. *Peak workload processing performance.* Can the system handle the workload during peak processing periods? If not, improved hardware and/or software may be needed to increase efficiency or processing may need to be rescheduled—that is, consider doing some of the less critical processing during nonpeak periods.
  c. *Human engineering test.* Is the system as easy to learn and use as anticipated? If not, is it adequate? Can enhancements to human engineering be deferred until after the system has been placed into operation?
  d. *Methods and procedures test.* During conversion, the methods and procedures for the new system will be put to their first real test. Methods and procedures may have to be modified if they prove to be awkward and inefficient from the end users' standpoint.
  e. *Backup and recovery testing.* All backup and recovery procedures should be tested. This should include simulating a data loss disaster and testing the time required to recover from that disaster. Also, a before-and-after comparison of the data should be performed to ensure that data was properly recovered. It is crucial to test these procedures. Don't wait until the first disaster to find an error in the recovery procedures.

audit test a test performed to ensure a new system is ready to be placed into operation.

- **Audit testing** certifies that the system is free of errors and is ready to be placed into operation. Not all organizations require an audit. But many firms have an independent audit or quality assurance staff that must certify a system's acceptability and documentation before that system is placed into final operation. There are independent companies that perform systems and software certification for end users' organizations.

3 marks for explaining each level of testing with a maximum of 8 marks

TEXT BOOK

I. Systems Analysis and Design Methods, Jeffrey L Whitten, Lonnie D Bentley, Seventh Edition, TMH, 2007