13MCA33

USN

Third Semester MCA Degree Examination, Dec.2016/Jan.2017
Software Engineering

Time: 3 hrs. Max. Marks:100

Note: ***Answer any FIVE full questions.***

1 a. What are the attributes of good software? **(08 Marks)**
b. What are the key challenges faced by software engineer? Explain. **(06 Marks)**
c. Explain the professional responsibilities of a software engineer. **(06 Marks)**

2 a. Explain the principles of agile methods. **(06 Marks)**
b. Discuss rational unified process with a neat diagram. **(06 Marks)**
c. Explain waterfall model with its merits and demerits. **(08 Marks)**

3 a. Discuss requirement engineering process with a neat diagram. **(07 Marks)**
b. Explain requirements elicitation and analysis process. **(07 Marks)**
c. What are the requirements validation techniques? Explain briefly. **(06 Marks)**

4 a. Explain system models with suitable examples. **(10 Marks)**
b. What is architectural design? Explain the repository model and client-server model with an example for each. **(10 Marks)**

5 a. Explain basic elements of a component model with a neat diagram. **(12 Marks)**
b. List out the advantages and disadvantages of using a distributed approach to systems development. **(08 Marks)**

6 a. Differentiate between black box testing and white box testing. **(08 Marks)**
b. Name the various estimation techniques in software systems. **(06 Marks)**
c. Discuss project scheduling and staffing. **(06 Marks)**

7 a. Explain risk management process with a neat diagram. **(08 Marks)**
b. Explain functional and non-functional requirements. **(04 Marks)**
c. What are the practices followed in extreme programming? **(08 Marks)**

8 Briefly explain the following:
a. CBSE process.
b. Data flow diagram of an ATM.
c. Software as a service.
d. Function oriented design. **(20 Marks)**

* * * * *

Answer Key –VTU QP 2016

Subject Code- 13MCA33 Subject Name: Software Engineering

Q1(a) : What are the attributes of good software?

Ans: **Attributes of good Software**: Good software should deliver the required functionality and performance to the user and should be maintainable, dependable and usable.

Product characteristic	**Description**
Maintainability	Software should be written in such a way so that it can evolve to meet the changing needs of customers. This is a critical attribute because software change is an inevitable requirement of a changing business environment.
Dependability and security	Software dependability includes a range of characteristics including reliability, security and safety. Dependable software should not cause physical or economic damage in the event of system failure. Malicious users should not be able to access or damage the system.
Efficiency	Software should not make wasteful use of system resources such as memory and processor cycles. Efficiency therefore includes responsiveness, processing time, memory utilisation, etc.
Acceptability	Software must be acceptable to the type of users for which it is designed. This means that it must be understandable, usable and compatible with other systems that they use.

Q1(b)What are the key challenges faced by software engineer? Explain.

Ans: Coping with increasing diversity, demands for reduced delivery times and developing trustworthy software are the key challenges faced by software engineer.

    - General issues that affect most software:
    - Heterogeneity
    - Increasingly, systems are required to operate as distributed systems across networks that include different types of computer and mobile devices.

- ✧ Business and social change
    - Business and society are changing incredibly quickly as emerging economies develop and new technologies become available. They need to be able to change their existing software and to rapidly develop new software.
- ✧ Security and trust
    - As software is intertwined with all aspects of our lives, it is essential that we can trust that software.

Q1(c) : Explain the Professional responsibilities of a software engineer.

Ans: Software engineering involves wider responsibilities than simply the application of technical skills. Software engineers must behave in an honest and ethically responsible way if they are to be respected as professionals. The professional societies in the US have cooperated to produce a code of ethical practice. The Code contains eight Principles related to the behaviour of and decisions made by professional software engineers, including practitioners, educators, managers, supervisors and policy makers, as well as trainees and students of the profession.

- ✧ PUBLIC - Software engineers shall act consistently with the public interest.
- ✧ 2. CLIENT AND EMPLOYER - Software engineers shall act in a manner that is in the best interests of their client and employer consistent with the public interest.
- ✧ 3. PRODUCT - Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.
- ✧ 4. JUDGMENT - Software engineers shall maintain integrity and independence in their professional judgment.
- ✧ 5. MANAGEMENT - Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.
- ✧ 6. PROFESSION - Software engineers shall advance the integrity and reputation of the profession consistent with the public interest.

✧ 7. COLLEAGUES - Software engineers shall be fair to and supportive of their colleagues.

✧ 8. SELF - Software engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession

Q2(a) Discuss the principles of agile methods.

Ans: **The principles of agile methods**

Principle	**Description**
Customer involvement	Customers should be closely involved throughout the development process. Their role is provide and prioritize new system requirements and to evaluate 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 and so design the system 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.

Q2(b): Discuss rational unified process with a neat diagram.

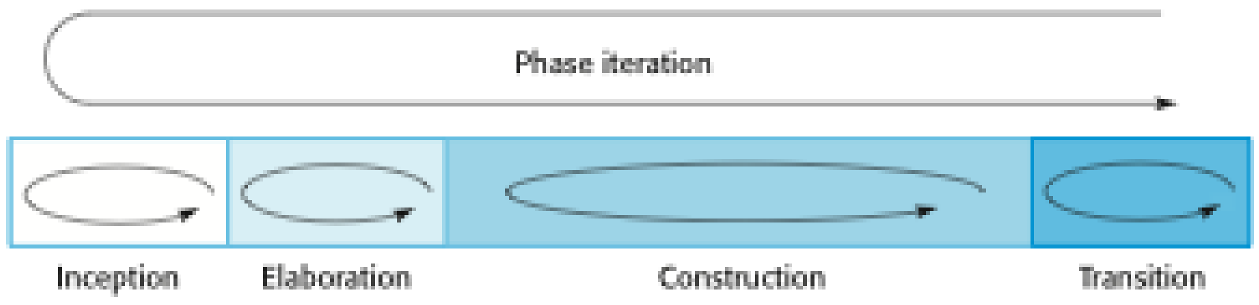
Ans: **The Rational Unified Process**

The rational unified process is a. modern generic process derived from the work on the UML and associated process. It brings together aspects of the 3 generic process models discussed previously.

It normally described from 3 perspectives:

- A dynamic perspective that shows phases over time;
- A static perspective that shows process activities;
- A practive perspective that suggests good practice.

Phases in the Rational Unified Process



RUP phases

- ✧ Inception
  - In this phase Software engineer establish the business case for the system.
- ✧ Elaboration
  - In this phase Software engineer develop an understanding of the problem domain and the system architecture.
- ✧ Construction
  - In this phase Software engineer will concentrate on system design, programming and testing.
- ✧ Transition
  - In this phase Software engineer will deploy the system in its operating environment.

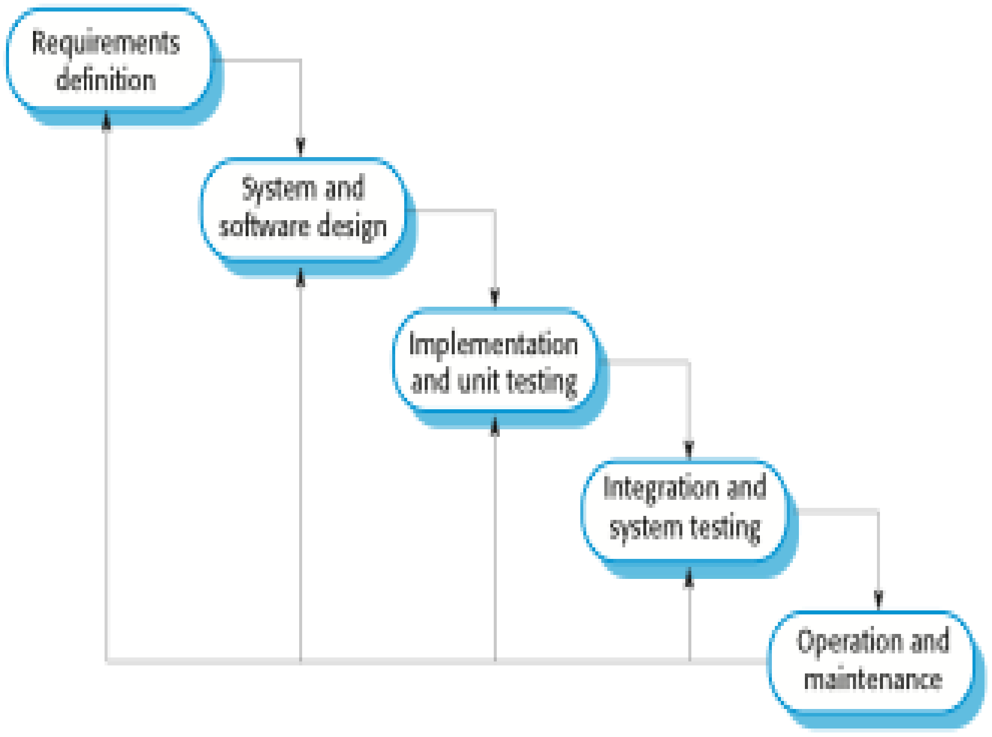
RUP iteration

- ✧ In-phase iteration
  - Each phase is iterative with results developed incrementally.
- ✧ Cross-phase iteration

- As shown by the loop in the RUP model, the whole set of phases may be enacted incrementally.

Q2(c): Explain waterfall model with its merits and demerits.

- Ans: The waterfall model is a plan-driven model. It has separate and distinct phases of specification and development.


Requirements definition
System and software design
Implementation and unit testing
Integration and system testing
Operation and maintenance

✧ There are separate identified phases in the waterfall model:

- Requirements analysis and definition
- System and software design
- Implementation and unit testing
- Integration and system testing
- Operation and maintenance

✧ The main drawback of the waterfall model is the difficulty of accommodating change after the process is underway. In principle, a phase has to be complete before moving onto the next phase.

Waterfall model problems

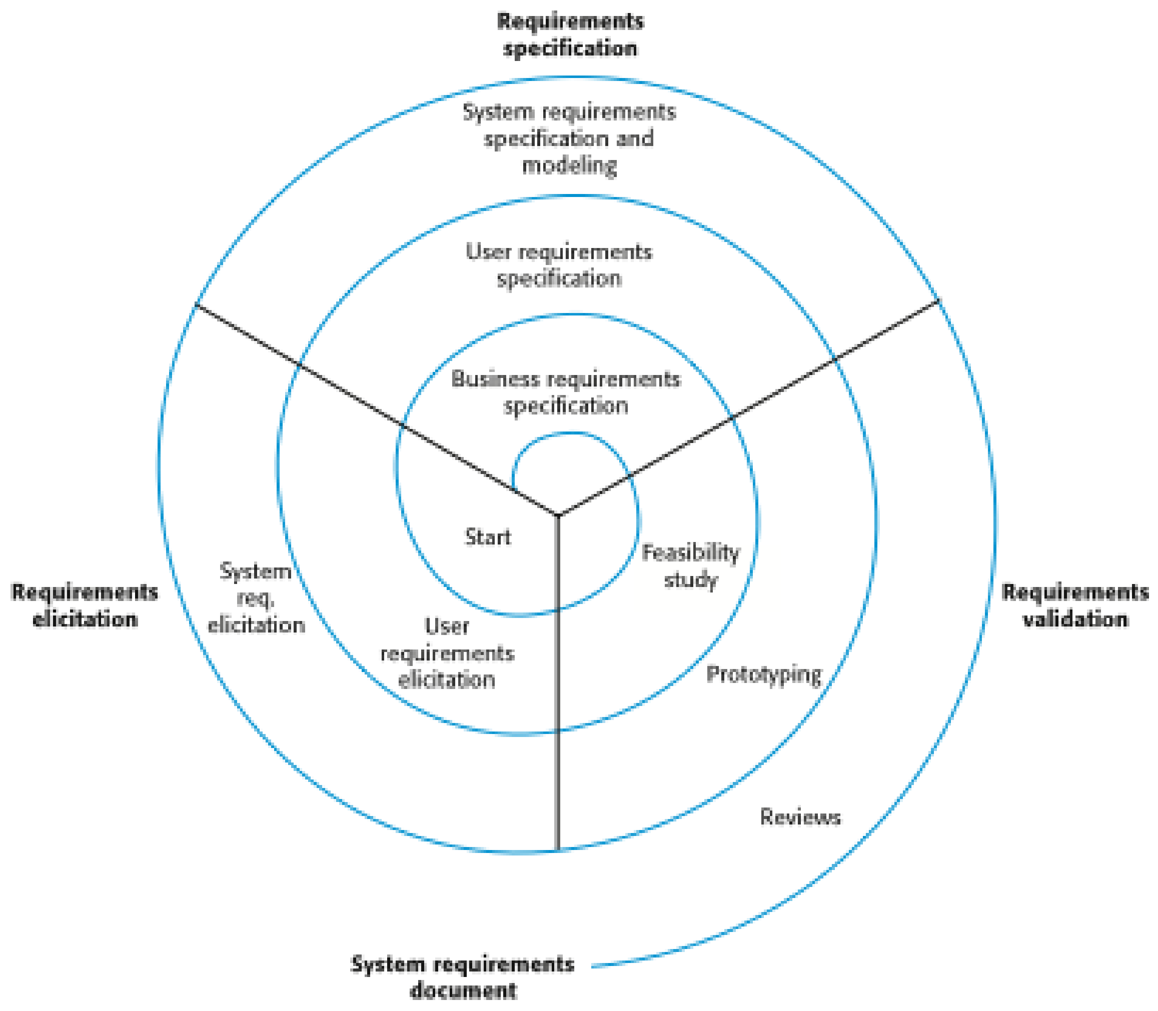
- Inflexible partitioning of the project into distinct stages makes it difficult to respond to changing customer requirements.
  - Therefore, this model is only appropriate when the requirements are well-understood and changes will be fairly limited during the design process.
  - Few business systems have stable requirements.
- The waterfall model is mostly used for large systems engineering projects where a system is developed at several sites.
  - In those circumstances, the plan-driven nature of the waterfall model helps coordinate the work.

Q3(a): Discuss requirement engineering process with a neat diagram.

Requirements engineering processes:

- The processes used for RE vary widely depending on the application domain, the people involved and the organisation developing the requirements.
- However, there are a number of generic activities common to all processes
  - Requirements elicitation;
  - Requirements analysis;
  - Requirements validation;
  - Requirements management.
- In practice, RE is an iterative activity in which these processes are interleaved.

A spiral view of the requirements engineering process:


Requirements specification
System requirements specification and modeling
User requirements specification
Business requirements specification
Start
Feasibility study
Requirements elicitation
System req. elicitation
User requirements elicitation
Prototyping
Requirements validation
Reviews
System requirements document

Requirements elicitation and analysis is sometimes called requirements elicitation or requirements discovery. It involves technical staff working with customers to find out about the application domain, the services that the system should provide and the system's operational constraints. It may involve end-users, managers, engineers involved in maintenance, domain experts, trade unions, etc. These are called *stakeholders.* Software engineers work with a range of system stakeholders to find out about the application domain, the services that the system should provide, the required system performance, hardware constraints, other systems, etc.

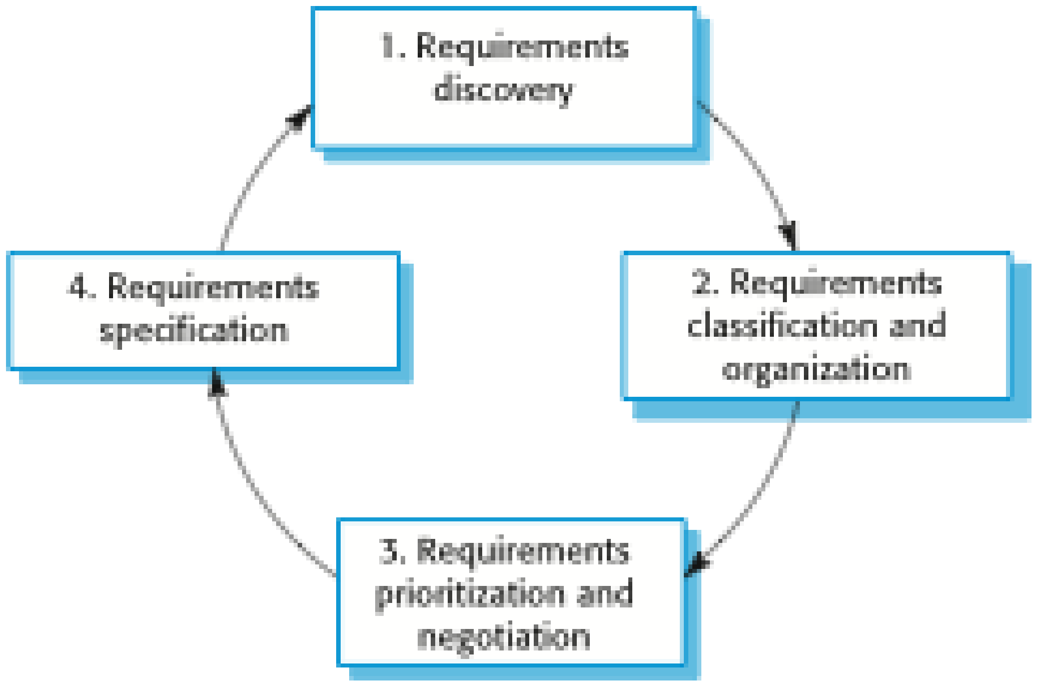
Requirement validation concerned with demonstrating that the requirements define the system that the customer really wants. Requirements error costs are high so validation is very important fixing a requirements error after delivery may cost up to 100 times the cost of fixing an implementation error.

Requirements management is the process of managing changing requirements during the requirements engineering process and system development. New requirements emerge as a system is being developed and after it has gone into use.You need to keep track of individual requirements and maintain links between dependent requirements so that you can assess the impact of requirements changes. You need to establish a formal process for making change proposals and linking these to system requirements.

Q3(b): Explain requirements elicitation and analysis process.

Ans : Requirements elicitation and analysis is sometimes called requirements elicitation or requirements discovery. It involves technical staff working with customers to find out about the application domain, the services that the system should provide and the system's operational constraints. It may involve end-users, managers, engineers involved in maintenance, domain experts, trade unions, etc. These are called *stakeholders.* Software engineers work with a range of system stakeholders to find out about the application domain, the services that the system should provide, the required system performance, hardware constraints, other systems, etc.

- ✧ Stages include:
    - Requirements discovery,
    - Requirements classification and organization,
    - Requirements prioritization and negotiation,
    - Requirements specification.


1. Requirements discovery
2. Requirements classification and organization
3. Requirements prioritization and negotiation
4. Requirements specification

- ✧ Requirements discovery

    - Interacting with stakeholders to discover their requirements. Domain requirements are also discovered at this stage.

- ✧ Requirements classification and organisation
    - Groups related requirements and organises them into coherent clusters.
- ✧ Prioritisation and negotiation
    - Prioritising requirements and resolving requirements conflicts.
- ✧ Requirements specification
    - Requirements are documented and input into the next round of the spiral.

Q3(c): What are the requirement validation techniques? Explain briefly.

Ans: Requirement validation concerned with demonstrating that the requirements define the system that the customer really wants. Requirements error costs are high so validation is very important fixing a requirements error after delivery may cost up to 100 times the cost of fixing an implementation error.

Requirements checking: Following things should be checked for requirement:

- ✧ Validity. Does the system provide the functions which best support the customer's needs?
- ✧ Consistency. Are there any requirements conflicts?
- ✧ Completeness. Are all functions required by the customer included?
- ✧ Realism. Can the requirements be implemented given available budget and technology
- ✧ Verifiability. Can the requirements be checked?

Requirements validation techniques:

- ✧ Requirements reviews: Systematic manual analysis of the requirements will be done.
- ✧ Prototyping: Using an executable model of the system will check the requirements.
- ✧ Test-case generation: Develop tests for requirements to check testability of the requirements.

Requirements reviews:

- ✧ Regular reviews should be held while the requirements definition is being formulated.
- ✧ Both client and contractor staff should be involved in reviews.
- ✧ Reviews may be formal (with completed documents) or informal. Good communications between developers, customers and users can resolve problems at an early stage.

Q4 (a) Explain system models with suitable examples.

Ans: System modeling is the process of developing abstract models of a system, with each model presenting a different view or perspective of that system. System modeling has now come to mean representing a system using some kind of graphical notation, which is now almost always based on notations in the Unified Modeling Language (UML). System modelling helps the analyst to understand the functionality of the system and models are used to communicate with customers.

System perspectives: We can develop different models to represent the system from different perspectives.

- ✧ An external perspective, where you model the context or environment of the system.
- ✧ An interaction perspective, where you model the interactions between a system and its environment, or between the components of a system.
- ✧ A structural perspective, where you model the organization of a system or the structure of the data that is processed by the system.
- ✧ A behavioral perspective, where you model the dynamic behavior of the system and how it responds to events.

UML diagram types: Five diagrams that could represent the essentials of a system:

- ✧ Activity diagrams, which show the activities involved in a process or in data processing .
- ✧ Use case diagrams, which show the interactions between a system and its environment.
- ✧ Sequence diagrams, which show interactions between actors and the system and between system components.
- ✧ Class diagrams, which show the object classes in the system and the associations between these classes.
- ✧ State diagrams, which show how the system reacts to internal and external events.

There are different kinds of models that can be developed:

Context models:

- ✧ Context models are used to illustrate the operational context of a system - they show what lies outside the system boundaries.
- ✧ Social and organisational concerns may affect the decision on where to position system boundaries.
- ✧ Architectural models show the system and its relationship with other systems.

Interaction models:

- ✧ Modeling user interaction is important as it helps to identify user requirements.
- ✧ Modeling system-to-system interaction highlights the communication problems that may arise.
- ✧ Modeling component interaction helps us understand if a proposed system structure is likely to deliver the required system performance and dependability.
- ✧ Use case diagrams and sequence diagrams may be used for interaction modeling.

Structural models:

- ✧ Structural models of software display the organization of a system in terms of the components that make up that system and their relationships.
- ✧ Structural models may be static models, which show the structure of the system design, or dynamic models, which show the organization of the system when it is executing.
- ✧ You create structural models of a system when you are discussing and designing the system architecture.

Behavioral models:

- ✧ Behavioral models are models of the dynamic behavior of a system as it is executing. They show what happens or what is supposed to happen when a system responds to a stimulus from its environment.
- ✧ You can think of these stimuli as being of two types:
  - ▪ Data: Some data arrives that has to be processed by the system.
  - ▪ Events: Some event happens that triggers system processing. Events may have associated data, although this is not always the case.

Q4(b) : What is architectural design? Explain the repository model and client- server model with an example for each.

Architectural design provides a very high level view of the parts of the system and how they are related to form the whole system. It partitions the system in logical parts.There is no unique structure of the system that can be described by its architecture; there are many possible structures.

The software architecture of a system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.

Some of the important uses that software architecture descriptions play are:

Understanding and communication: An architecture description is primarily to communicate the architecture to its various stakeholders, which include the users who will use the system, the clients who commissioned the system, the builders who will build the system, and, of course, the architects.
Reuse: The architecture has to be chosen in a manner such that the components which have to be reused can fit properly and together with other components that may be developed.
Construction and Evolution: As architecture partitions the system into parts, some architecture-provided partitioning can naturally be used for constructing the system, which also requires that the system be broken into parts such that different teams (or individuals) can separately work on different parts.
Repository Model or Shared-Data Style

In this style, there are two types of components—data repositories and data assessors. Components of data repository type are where the system stores shared data—these could be file systems or databases. These components provide a reliable and permanent storage, take care of any synchronization needs for concurrent access, and provide data access support. Components of data assessors type access data from the repositories, perform computation on the data obtained, and if they want to share the results with other components, put the results back in the depository.

There are two variations of this style possible. In the blackboard style, if some data is posted on the data repository, all the assessor components that need to know about it are informed

The other is the repository style, in which the data repository is just a passive repository which provides permanent storage and related controls for data accessing. The components access the repository as and when they want.

As an example of a system using this style of architecture, let us consider a student registration system in a university. The system clearly has a central repository which contains information about courses, students, prerequisites, etc. It has an Administrator component that sets up the repository, rights to different people, etc. The Registration component allows students to register and update the information for students and courses. The Approvals component is for granting approvals for those courses that require instructor's consent. The Reports component produces the report regarding the students registered in different courses at the end of the registration. The component Course Feedback is used for taking feedback from students at the end of the course.
.

Client-Server Style

In this style, there are two component types—clients and servers. A constraint of this style is that a client can only communicate with the server, and cannot communicate with other clients. The communication between a client component and a server component is initiated by the client when the client sends a request for some service that the server supports. The server receives the request at its defined port, performs the service, and then returns the results of the computation to the client who requested the service.

There is one connector type in this style—the request/reply type. A connector connects a client to a server.

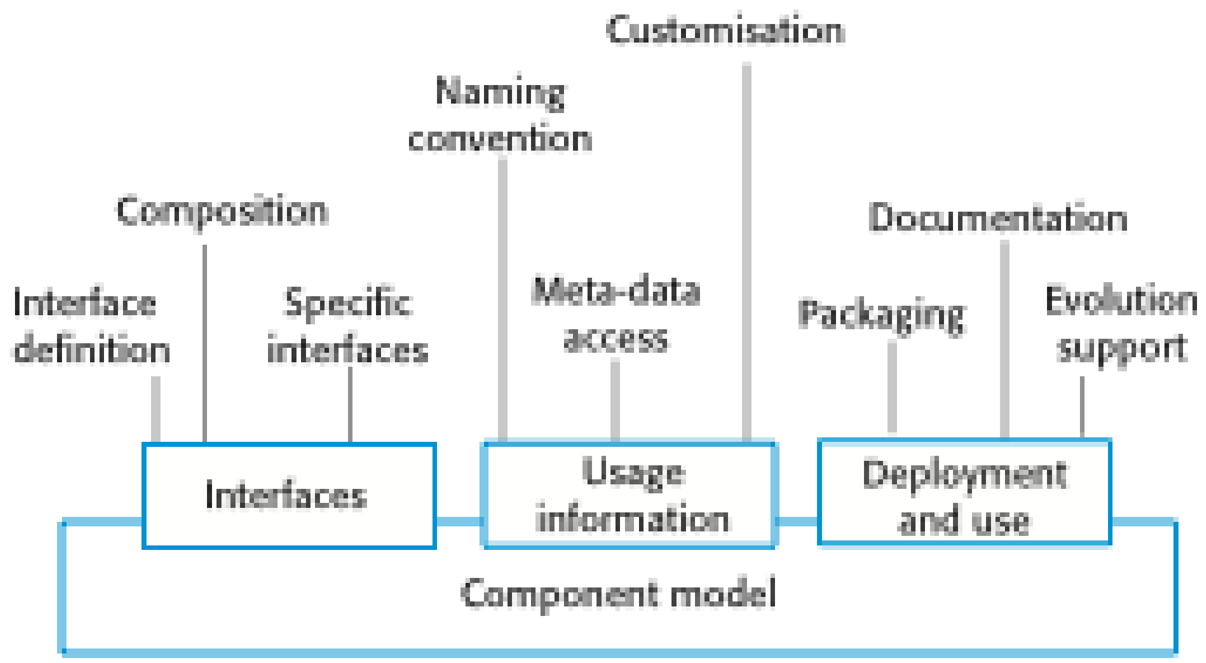
For example suppose we have to design and build a simple system for taking an on-line survey of students on a campus. There is a set of multiple-choice questions, and the proposed system will provide the survey form to the student, who can fill and submit it on-line. We also want that when the user submits the form, he/she is also shown the current result of the survey, that is, what percentage of students so far have filled which options for the different questions. The system is best built using the Web; this is the likely choice of any developer. For this simple system, traditional 3-tier architecture is proposed. It consists of a client which will display the form that the student can complete and submit, and will also display the results. The second component is the server, which processes the data submitted by the student, and saves it on the database, which is the third component. The server also queries the database to get the outcome of the survey and sends the results in proper format (HTML) back to the client, which then displays the result.

Q5(a) : Explain basic elements of a component model with a neat diagram.

Ans : **Component models:** A component model is a definition of standards for component implementation, documentation and deployment.

- ✧ Examples of component models
    - ▪ EJB model (Enterprise Java Beans)
    - ▪ COM+ model (.NET model)
    - ▪ Corba Component Model
- ✧ The component model specifies how interfaces should be defined and the elements that should be included in an interface definition.

  Basic elements of a component model



- **Interfaces**
  - Components are defined by specifying their interfaces. The component model specifies how the interfaces should be defined and the elements, such as operation names, parameters and exceptions, which should be included in the interface definition.
- **Usage**
  - In order for components to be distributed and accessed remotely, they need to have a unique name or handle associated with them. This has to be globally unique.
- **Deployment**
  - The component model includes a specification of how components should be packaged for deployment as independent, executable entities.

Q5(b) : List out the advantages and disadvantages of using a distributed approach to systems development.

Ans : a collection of independent computers that appears to the user as a single coherent system is called distributed system.

The benefits of distributed systems are that they can be scaled to cope with increasing demand, can continue to provide user services if parts of the system fail, and they enable resources to be shared.

Issues to be considered in the design of distributed systems include transparency, openness, scalability, security, quality of service and failure management.

Distributed system advantages:

Resource sharing: In distributed system sharing of hardware and software resources will be done among systems.

Openness: Distributed system will use of equipment and software from different vendors.

Concurrency: Distributed system uses concurrent processing to enhance performance.

Scalability: In distributed system we can increase throughput by adding new resources.

Fault tolerance: The ability to continue in operation after a fault has occurred is called fault tolerance.

Distributed systems issues:

- ✧ Distributed systems are more complex than systems that run on a single processor.
- ✧ Complexity arises because different parts of the system are independently managed as is the network.
- ✧ There is no single authority in charge of the system so top-down control is impossible.

Design issues: There are several design issues needs to be considered while developing distributed system:

- ✧ *Transparency* To what extent should the distributed system appear to the user as a single system?
- ✧ *Openness:* Should a system be designed using standard protocols that support interoperability?
- ✧ *Scalability:* How can the system be constructed so that it is sclable?
- ✧ *Security* How can usable security policies be defined and implemented?
- ✧ *Quality of service* How should the quality of service be specified.

- ✧ *Failure management* How can system failures be detected contained and repaired?

Q6(a) : Differentiate between black box testing and white box testing.

Ans:

S N	**Black Box Testing**	**White Box Testing**
1	In this testing knowledge of programming is not necessarily essential.	In this form of testing knowledge of programming is must means it is essential.
2	Normally independent software testers are responsible for doing Black Box Testing.	Normally software developers are responsible for doing White Box Testing.
3	In this form of testing Knowledge of implementation is not required.	In this form of testing Implementation knowledge is required.
4	In Black Box Testing, testers may or may not be **technically sound**.	Normally software developers are involved in this testing, but if it is performed by software testers, then testers should be **technically sound**.
5	In this sort of testing testers mainly focuses on the functionality of the system.	In this sort of testing developers mainly focuses on the structure means program/code of the system.
6	This testing is done by testers.	This testing is mostly done by developers.
7	This type of testing always focuses on what is performing/ carried out.	This type of testing always focuses on how it is performing/ carried out.
8	In Black Box Testing no knowledge regarding internal logic of code is needed means no need of programming is necessary.	In White Box Testing knowledge regarding internal logic of code is needed means need of programming is mandatory.
9	Other names of this testing include means synonyms of **black box testing** are testing regarding functionality means Functional testing, Behavioral testing, and Opaque-box/ Closed-box testing that is the reason why in this testing no knowledge of programming is needed.	Other names of this testing include means synonyms of **white box testing** are testing regarding code means Structural testing, Glass-box/ Clear-box testing, Open-box testing/ Transparent-box testing, Logic-driven testing and Path-oriented testing that is the reason why in this testing knowledge of programming is needed.
10	Black box testing means functional test or external test.	White box testing means structural test or interior test.

Q6(b) : Name the various estimation techniques in software systems.

Ans: Types of effort estimation approaches

- Top- Down Estimation Approach
- Bottom-Up Estimation Approach

Top- Down Estimation Approach

a. Consider effort as a function of project size.
b. First determine the nature of the function.
c. Then estimate the size of the function.

Past productivity on similar projects can be used as the estimation function.If productivity is P KLOC/PM, then

effort estimate = SIZE/P person-months

More general function

$$\text{EFFORT}= a*\text{SIZE}^b$$

Where a and b are constant and determined through regression analysis.

Bottom-Up Estimation Approach

a. Project is first divided into tasks
b. And then estimates for the different tasks of the project are obtained.
c. From the estimates of the different tasks, the overall estimate is determined.
d. This type of approach is also called activity-based estimation.

Q6(c) : Discuss project scheduling and staffing.

Ans: **Project scheduling and staffing**

For a project with some estimated effort, multiple schedules (or project duration) are indeed possible. Once the effort is fixed, there is some flexibility in setting the schedule by appropriately staffing the project, but this flexibility is not unlimited. The overall schedule can be determined as a function of effort. Such function can be determined from data from completed projects using statistical techniques like fitting a regression curve

M, in calendar months can also be estimated by $M = 4.1E^{.36}$.

In COCOMO, the equation for schedule for an organic type of software is $M = 2.5E^{.38}$

Another method for medium-sized projects is the rule of thumb called the square root check .

The proposed schedule can be around the square root of the total effort in person months.

For example, if the effort estimate is 50 person-months, a schedule of about 7 to 8 months will be suitable.

Q7(a) Explain risk management process with a neat diagram.

Following are the steps involved in Risk Management and Planning Approach:

1. For each risk, rate the probability of its happening as low, medium, or high.
2. For each risk, assess its impact on the project as low, medium, or high.
3. Rank the risks based on the probability and effects on the project; for example, a high-probability, high-impact item will have higher rank than a risk item with a medium probability and high impact. In case of conflict,use judgment.
4. Select the top few risk items for mitigation and tracking.

Q7(b) Explain functional and non- functional requirements.

Ans:

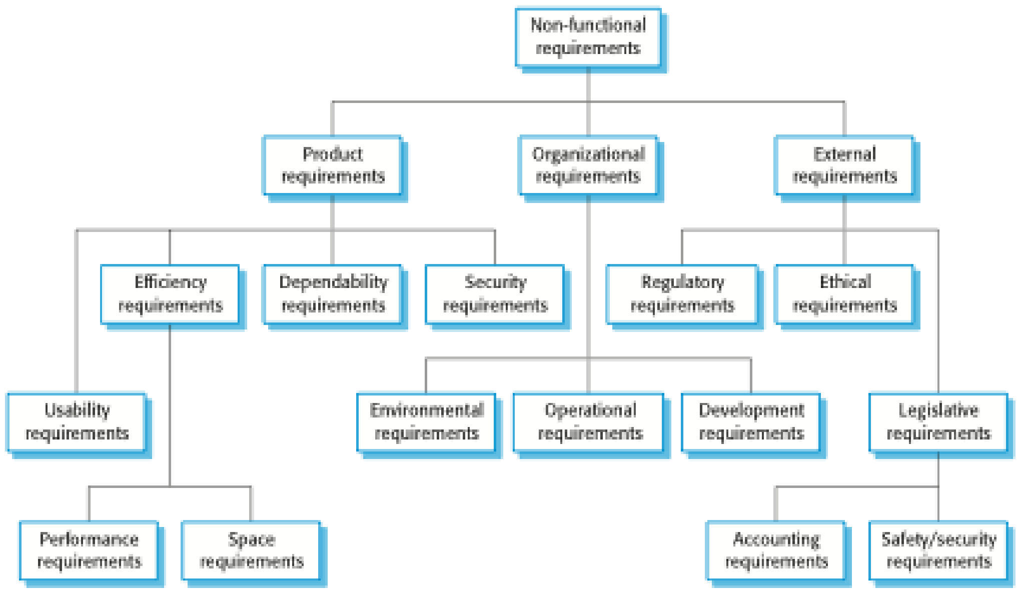
Functional requirements are

- Statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations.
- May state what the system should not do.

- Describe functionality or system services.
- Depend on the type of software, expected users and the type of system where the software is used.
- Functional user requirements may be high-level statements of what the system should do.
- Functional system requirements should describe the system services in detail.

Non-functional requirements are

- Constraints on the services or functions offered by the system such as timing constraints, constraints on the development process, standards, etc.
- Often apply to the system as a whole rather than individual features or services.
- These define system properties and constraints e.g. reliability, response time and storage requirements. Constraints are I/O device capability, system representations, etc.
- Process requirements may also be specified mandating a particular IDE, programming language or development method.
- Non-functional requirements may be more critical than functional requirements. If these are not met, the system may be useless.



Q7(c) : What are the practices followed in extreme programming?

Ans : **Extreme programming practices:**

Principle or practice	**Description**
Incremental planning	**Requirements are recorded on story cards and the stories to be included in a release are determined by the time available and their relative priority. The developers break these stories into development 'Tasks'. See Figures 3.5 and 3.6.**
Small releases	**The minimal useful set of functionality that provides business value is developed first. Releases of the system are frequent and incrementally add functionality to the first release.**
Simple design	**Enough design is carried out to meet the current requirements and no more.**
Test-first development	**An automated unit test framework is used to write tests for a new piece of functionality before that functionality itself is implemented.**
Refactoring	**All developers are expected to refactor the code continuously as soon as possible code improvements are found. This keeps the code simple and maintainable.**

Q 8 : Briefly explain the following:

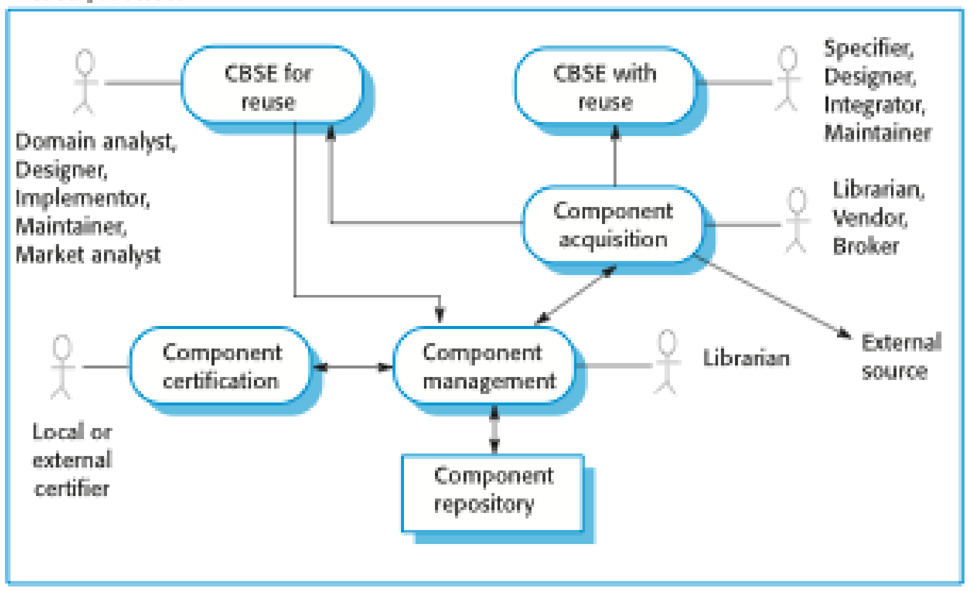
a. **CBSE process**:

Ans CBSE processes are software processes that support component-based software engineering. They take into account the possibilities of reuse and the different process activities involved in developing and using reusable components.

- ✧ Development for reuse
    - This process is concerned with developing components or services that will be reused in other applications. It usually involves generalizing existing components.
- ✧ Development with reuse

- This process is the process of developing new applications using existing components and services.

CBSE processes


CBSE for reuse
CBSE with reuse
Specifier, Designer, Integrator, Maintainer
Domain analyst, Designer, Implementor, Maintainer, Market analyst
Component acquisition
Librarian, Vendor, Broker
External source
Component certification
Component management
Librarian
Local or external certifier
Component repository

- Component acquisition is the process of acquiring components for reuse or development into a reusable component. It may involve accessing locally-developed components or services or finding these components from an external source.

✧ Component management is concerned with managing a company's reusable components, ensuring that they are properly catalogued, stored and made available for reuse.

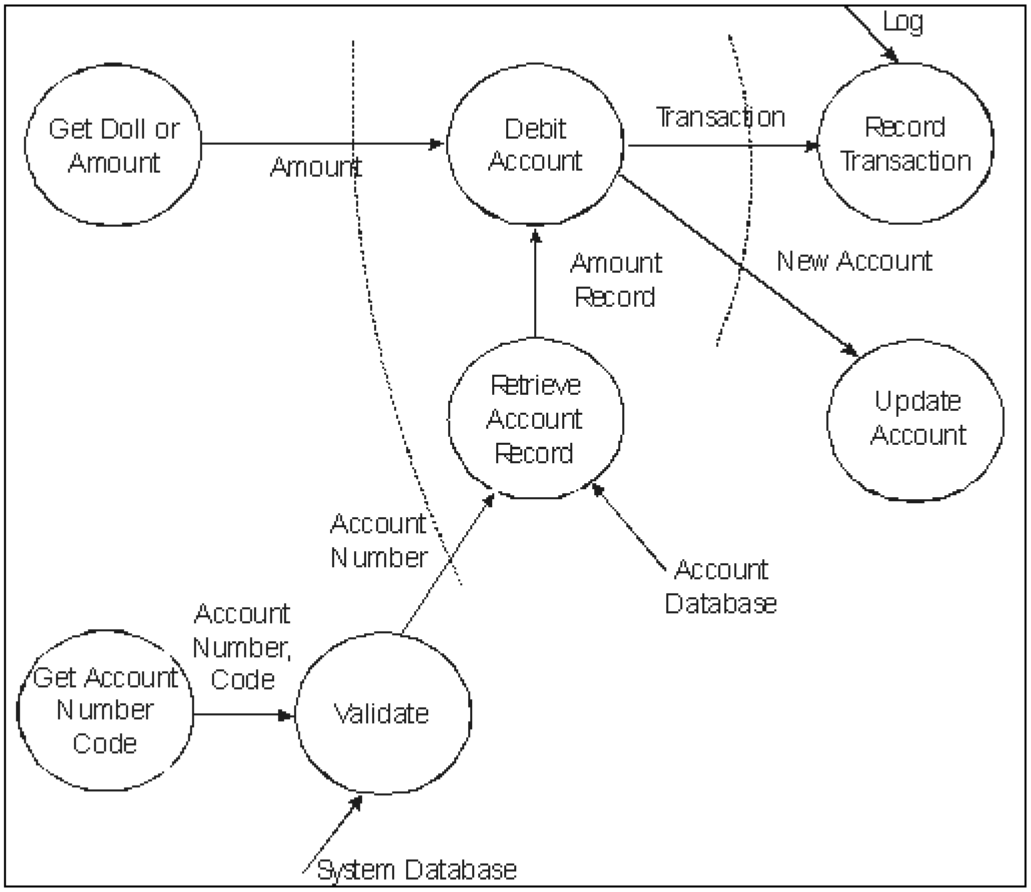
✧ Component certification is the process of checking a component and certifying that it meets its specification.

✧ CBSE for reuse focuses on component development. Components developed for a specific application usually have to be generalised to make them reusable. A component is most likely to be reusable if it associated with a stable domain abstraction (business object). For example, in a hospital stable domain abstractions are associated with the fundamental purpose - nurses, patients, treatments, etc.

✧ CBSE with reuse process has to find and integrate reusable components. When reusing components, it is essential to make trade-offs between ideal requirements and the services actually provided by available components.

✧ This involves:
  - Developing outline requirements;
  - Searching for components then modifying requirements according to available functionality.
  - Searching again to find if there are better components that meet the revised requirements.
  - Composing components to create the system.

b. Data flow diagram of an ATM:



During design activity, we are no longer modeling the problem domain, but are dealing with the solution domain and developing a model for the eventual system. That is, the DFD during design represents how the data will flow in the system when it is built. In this modeling, the major transforms or functions in the software are decided, and the DFD shows the major transforms that the software will have and how the data will flow through different transforms. A DFD of an ATM is shown in Figure above. There are two major streams of input data in this diagram. The first is the account number and the code, and the second is the amount to be debited. Notice the use of * at different places in the DFD. For example, the transform "validate," which verifies if the account number and code are valid, needs not only the account number and code, but also information from the system database to do the validation. And the transform debit account has two outputs, one used for recording the transaction and the other to update the account.

c. Software as a service:

Software as a service (SaaS) involves hosting the software remotely and providing access to it over the Internet.

a. Software is deployed on a server (or more commonly a number of servers) and is accessed through a web browser. It is not deployed on a local PC.

b. The software is owned and managed by a software provider, rather than the organizations using the software.
c. Users may pay for the software according to the amount of use they make of it or through an annual or monthly subscription.

Software is deployed on a server (or more commonly a number of servers) and is accessed through a web browser. It is not deployed on a local PC.

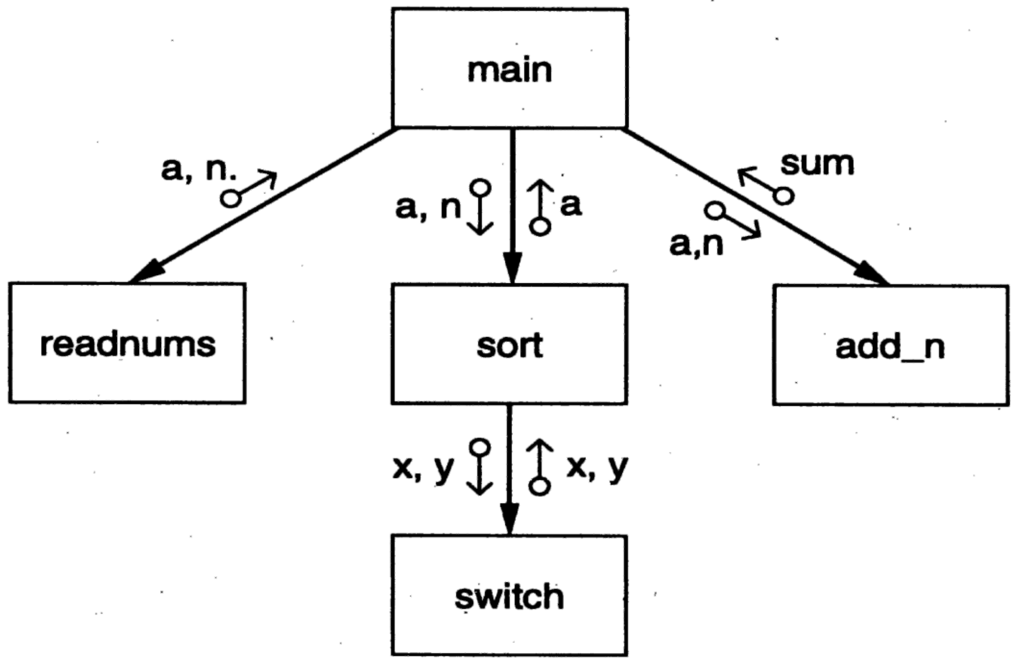
The software is owned and managed by a software provider, rather than the organizations using the software.

Users may pay for the software according to the amount of use they make of it or through an annual or monthly subscription. Sometimes, the software is free for anyone to use but users must then agree to accept advertisements, which fund the software service.

Software as a service is a way of providing functionality on a remote server with client access through a web browser. The server maintains the user's data and state during an interaction session. Transactions are usually long transactions e.g. editing a document.

d. Function oriented design:

Creating the software system design is the major concern of the design phase. We discuss the structured design methodology for developing function-oriented system designs. The methodology employs the structure chart notation for creating the design. For a function-oriented design, the design can be represented graphically by structure charts. The structure of a program is made up of the modules of that program together with the interconnections between modules. Every computer program has a structure, and given a program its structure can be determined. The structure chart of a program is a graphic representation of its structure. In a structure chart a module is represented by a box with the module name written in the box. An arrow from module A to module B represents that module A invokes module B. B is called the subordinate of A, and A is called the superordinate of B. The arrow is labeled by the parameters received by B as input and the parameters returned by B as output, with the direction
of flow of the input and output parameters represented by small arrows. The parameters can be shown to be data (unfilled circle at the tail of the label) or control (filled circle at the tail). As an example, consider the structure of the following program, whose structure is shown in Figure


main
a, n.
a, n
a
sum
a,n
readnums
sort
add_n
x, y
x, y
switch