DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CS8494 SOFTWARE ENGINEERING

Question Bank

III YEAR A & B / BATCH : 2017 -21

SYLLABUS

CS 8494 - SOFTWARE ENGINEERING

UNIT I- SOFTWARE PROCESS AND PROJECT MANAGEMENT 9

Introduction to Software Engineering, Software Process, Perspective and Specialized Process Models .Introduction to Agility-Agile process-Extreme programming-XP Process.

UNIT II-REQUIREMENTS ANALYSIS AND SPECIFICATION 9

Software Requirements:Functional and Non-Functional, User requirements, System requirements,Software Requirements Document –Requirement Engineering Process: Feasibility Studies,Requirements elicitation and analysis, requirements validation, requirements management-Classicalanalysis: Structured system Analysis, Petri Nets-Data Dictionary

UNIT III-SOFTWARE DESIGN 9

Design process –Design Concepts-Design Model–Design Heuristic –Architectural Design –Architectural styles, Architectural Design, Architectural Mapping using Data Flow-User nterface Design: Interface analysis, Interface Design –Component levelDesign: Designing Class based components, traditional Components

UNIT IV -TESTING AND IMPLEMENTATION 9

Software testing fundamentals-Internal and external views of Testing-white box testing-basis pathtesting-control structure testing-black box testing-Regression Testing –Unit Testing –

IntegrationTesting –Validation Testing –System Testing And Debugging –Software ImplementationTechniques: Coding practices-ING

UNIT V -PROJECT MANAGEMENT 9

Estimation –FP Based, LOC Based, Make/Buy Decision, COCOMO II -Planning –Project Plan, Planning Process, RFP Risk Management –Identification, Projection,RMMM -Scheduling and Tracking –Relationship between people and effort, Task Set & Network, Scheduling, EVA –Processand Project Metrics

TEXT BOOKS

- Roger S. Pressman, "Software Engineering – A practitioner's Approach", Sixth Edition, McGraw-Hill International Edition, 2005
- Ian Sommerville, "Software engineering", Seventh Edition, Pearson Education Asia, 2007.

REFERENCES:

1. Rajib Mall, ―Fundamentals of Software Engineering‖, Third Edition, PHI Learning PrivateLimited, 2009.
2. PankajJalote, ―Software Engineering, A Precise Approach‖, Wiley India, 2010.
3. Kelkar S.A., ―Software Engineering‖, Prentice Hall of India Pvt Ltd, 2007.
4. Stephen R.Schach, ―Software Engineering‖, Tata McGraw-Hill Publishing Company Limited,2007.

INDEX PAGE

UNIT – 1

<u>PART –A</u>

S.NO **QUESTIONS**

1 **Write down the generic process framework that is applicable to any software project / relationship between work product, task, activity and system NOV/DEC-10,NOV/DEC2016, NOV/DEC 2017**

Common process frame work

- Process frame work activities
- Umbrella activities
- Frame work activities
- Task sets

2 **List the goals of software engineering? APR/MAY-11**

Satisfy user requirements , High reliability , Low maintenance cost , Delivery on time , Low production cost , High performance , Ease of reuse.

3 **What is the difference between verification and validation? NOV/DEC-10 , APR/MAY-11 , NOV/DEC-11, MAY/JUN-13**

- Verification refers to the set of activities that ensure that software correctly implements a specific function. Verification*:* "Are we building the product right?"
- Validation refers to a different set of activities that ensure that the software that has been built is traceable to customer requirements. Validation*:* "Are we building the right product?"

4 **For the scenario described below, which life cycle model would you choose? Give the reason why you would choose this model.** **NOV/DEC-11 ,**

You are interacting with the MIS department of a very large oil company with multiple departments. They have a complex regency system. Migrating the data from this legacy system is not an easy task and would take a considerable time. The oil company is very particular about processes, acceptance criteria and legal contracts.

Spiral model Proactive problem prevention. Each iteration has a risk analysis, sector that evaluates. Alternatives for proactive problem avoidance.

5 **Give two reasons why system engineers must understand the environment of a system?** **APR/MAY-12**

1. The reason for the existence of a system is to make some changes i its environment.

2. The functioning of a system can be very difficult to predict.

6 **What are the two types of software products?** **APR/MAY-12**

1. Generic products: these are stand-alone systems that are produced by a development Organization and sold in the open market to any customer who wants to buy it.
2. Customized products: these are systems that are commissioned by a specific customer and developed specially by some contractor to meet a special need.

7 **What is the advantage of adhering to life cycle models for software?** **NOV/DEC-12**

It helps to produce good quality software products without time and cost over runs.It encourages the development of software in a systematic & disciplined manner.

8 **Is it always possible to realize win-win spiral model for software? Justify. <u>NOV/DEC-12</u>**

- Must identify stake holder and their win condition
- Developing buy-in to the model is important than the model itself
- Eliminating the clashes between customers is important.

9 **What is software process? List its activities. <u>MAY/JUN-13</u>**

Software process is defined as the structured set of activities that are required to develop the software system.

Activities – Specification, design & implementation, validation & evolution.

10 **What are the various categories of software?**

- System software
- Application software
- Engineering/Scientific software
- Embedded software
- Web Applications
- Artificial Intelligence software

11 **What are the umbrella activities of a software process? <u>APR/MAY 2015</u>**

- Software project tracking and control.
- Risk management.
- Software Quality Assurance.
- Formal Technical Reviews.
- Software Configuration Management.
- Work product preparation and production.
- Reusability management.
- Measurement

12 **What are the merits of incremental model?**

i. The incremental model can be adopted when tere are less number of people involved in the project.

ii. Technical risks can be managed with each increment.

iii. For a very small time span,at least core product can be delivered to the customer.

13 **List the task regions in the Spiral model**.

- Customer communication – In this region it is suggested to establish customer communication.
- Planning – All planning activities are carried out in order to define resources timeline and otherproject related activities.
- Risk analysis – The tasks required to calculate technical and management risks.
- Engineering – In this the task region,tasks required to build one or more representations of applications are carried out.
- Construct and release – All the necessary tasks required to construct,test,install the applications are conducted. ¾_Customer evaluation – Customer‵‵ s feedback is obtained and based on the customer evaluation required tasks are performed and implemented at installation stage.

14 **Characteristics of software contrast to characteristics of hardware? <u>APR/MAY 2016</u>**

- Software is easier to change than hardware. The cost of change is much higher for hardware than for software.
- Software products evolve through multiple releases by adding new features and re-writing existing logic to support the new features. Hardware products consist of physical components that cannot be “refactored” after manufacturing, and cannot add new capabilities that require hardware changes.
- Specialized hardware components can have much longer lead times for acquisition than is true for software.
- Hardware design is driven by architectural decisions. More of the architectural work must be done up front compared to software products.
- The cost of development for software products is relatively flat over time.

However, the cost of hardware development rises rapidly towards the end of the development cycle.

- Testing software commonly requires developing thousands of test cases. Hardware testing involves far fewer tests.

Hardware must be designed and tested to work over a range of time and environmental conditions, which is not the case for software.

15 List the process maturity levels in SEIs CMM. NOV/DEC2015

Level 1:Initial– Few processes are defined and individual efforts are taken. Level 2:Repeatable– To track cost schedule and functionality basic project management processes are established.

Level 3:Defined– The process is standardized, documented and followed.

Level 4:Managed– Both the software process and product are quantitatively understood and controlled using detailed measures.

16 What does Verification represent?

Verification represents the set of activities that are carried out to confirm that the software correctly implements the specific functionality.

17 What does Validation represent?

Validation represents the set of activities that ensure that the software that has been built is satisfying the customer requirements.

18 What are the steps followed in testing? MAY/JUNE 2016

i. Unit testing – The individual components are tested in this type of testing.

ii. Module testing – Related collection of independent components are tested.

iii. Sub-system testing – This is a kind of integration testing. Various modules are integrated into a subsystem and the whole subsystem is tested.

iv. System testing – The whole system is tested in this system.

v. Acceptance testing – This type of testing involves testing of the system with customer data.If the system behaves as per customer need then it is accepted.

19 State the advantages and disadvantages in LOC based cost estimation? APR/MAY 2015

Advantages of LOC

- It is straight forward (simple)
- Easily can be automated (plenty of tools are available)

Disadvantages of LOC

- Its Language dependent
- Penalizes the well designed short programs
- Cannot easily accommodate nonprocedural languages
- Need a level of detail that may not be available at the early stages of development.

20 **What is requirement engineering?**

Requirement engineering is the process of establishing the services that the customer requires from the system and the constraints under which it operates and is developed.

21 **What are the various types of traceability in software engineering?**

i. Source traceability – These are basically the links from requirement to stakeholders who propose these requirements.

ii. Requirements traceability – These are links between dependant requirements.

iii. Design traceability – These are links from requirements to design.

22 **If you have to develop a word processing software product, what process models will you choose? Justify your answer. NOV/DEC 2016**

We will choose the incremental model for word processing software. It focuses on the aspects of the word processing software that are visible to the customer / end user. The feedback is used to refine the prototype.

23 **What led to the transition from product to process oriented development in software engineering? APR/MAY 2016**

Product techniques to designing software - Large numbers of software projects do not meet their expectations in terms of functionality, cost, or delivery schedule. Process - Composed of line practitioners who have varied skills, the group is at the center of the collaborative effort of everyone in the organization who is involved with software engineering process improvement.

Process-oriented view on cooperating software components based on the concepts and terminology of a language/action perspective on cooperative work provides a more suitable foundation for the analysis, design and implementation of software components in business applications.

24 **What are the advantages and disadvantages of iterative software development model <u>NOV/DEC 2015</u>**

Advantages

- In iterative model we can only create a high-level design of the application before we actually begin to build the product and define the design solution for the entire product.
- Building and improving the product step by step.
- can get the reliable user feedback
- Less time is spent on documenting and more time is given for designing.

Disadvantages

- Each phase of an iteration is rigid with no overlaps
- Costly system architecture or design issues may arise because not all requirements are gathered up front for the entire lifecycle

25 **What are the issues in measuring the software size using LOC as metric <u>NOV/DEC 2015, NOV/DEC 2017</u>**

- Lack of Accountability.
- Lack of Cohesion with Functionality.
- Adverse Impact on Estimation.
- Difference in Languages.
- Advent of GUI Tools
- Lack of Counting Standards.

26 **What is System Engineering? April/may 2018**

System Engineering means designing, implementing, deploying and operating systems which include hardware, software and people.

27 **What is the use of CMM? NOV/DEC2015**

Capability Maturity Model is used in assessing how well an organization's processes allow to complete and manage new software projects.

28 **What is meant by Software engineering paradigm?**

The development strategy that encompasses the process, methods and tools and generic phases is often referred to as a process model or software engineering paradigm.

29 **Define agility and agile team. April /May 2015**

- Agility-Effective (rapid and adaptive) response to change (team members, new technology, requirements)
- Effective communication in structure and attitudes among all team members, technological and business people, software engineers and managers。
- Drawing the customer into the team. Eliminate "us and them" attitude. Planning in an uncertain world has its limits and plan must be flexible.
- **Organizing a team** so that it is in control of the work performed
- The development guidelines stress delivery over analysis and design although these activates are not discouraged, and active and continuous communication between developers and customers
- Eliminate all but the most essential work products and keep them lean.

Emphasize an incremental delivery strategy as opposed to intermediate products that gets working software to the customer as rapidly as feasible

30 **Write any two characteristics of software as a product. April /May 2015**

1. Software is developed or engineered, it is not manufactured in the classical sense
2. Software doesn't "wear out."
3. Although the industry is moving toward component-based assembly, most software continues to be custom built.

31 **Write the IEEE definition of software engineering . NOV/DEC 2017**

According to **IEEE's definition software engineering** can be **defined** as the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of **software**, and the study of these approaches; that is, the application of **engineering** to **software**.

32 **List two deficiencies in waterfall model . Which process model do you suggest to overcome each deficiency. APRIL/MAY 2017**

•Once an application is in the testing stage, it is very difficult to go back and change something that was not well-thought out in the concept stage.

- No working software is produced until late during the life cycle.

33 **What is Agile?**

The word 'agile' means −

- Able to move your body quickly and easily.
- Able to think quickly and clearly.

In business, 'agile' is used for describing ways of planning and doing work wherein it is understood that making changes as needed is an important part of the job. Business'agililty' means that a company is always in a position to take account of the market changes.

In software development, the term 'agile' is adapted to mean 'the ability to respond to changes − changes from Requirements, Technology and People.'

34 **What is Agile Manifesto?**

The Agile Manifesto states that −

We are uncovering better ways of developing software by doing it and helping others do it. Through this work, we have come to value −

- **Individuals and interactions** over processes and tools.
- **Working software** over comprehensive documentation.
- **Customer collaboration** over contract negotiation.
- **Responding to change** over following a plan.

That is, while there is value in the items on the right, we value the items on the left more.

35

What are the Characteristics of Agility?

following are the characteristics of Agility −

- Agility in Agile Software Development focuses on the culture of the whole team with multi-discipline, cross-functional teams that are empowered and selforganizing.
- It fosters shared responsibility and accountability.
- Facilitates effective communication and continuous collaboration.
- The whole-team approach avoids delays and wait times.
- Frequent and continuous deliveries ensure quick feedback that in in turn enable the team align to the requirements.
- Collaboration facilitates combining different perspectives timely in implementation, defect fixes and accommodating changes.

36

What are the principles of of agile methods? 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.

37

What are the Problems with agile methods?

- It can be difficult to keep the interest of **customers** who are involved in the process.
- **Team members** may be unsuited to the intense involvement that characterizes agile methods.
- Prioritizing changes can be difficult where there are **multiple stakeholders**.
- **Maintaining simplicity** requires extra work.
- **Contracts** may be a problem as with other approaches to iterative development.

38 **What is Extreme Programming?**

XP is a lightweight, efficient, low-risk, flexible, predictable, scientific, and fun way to develop a software.

e**X**treme **P**rogramming (XP) was conceived and developed to address the specific needs of software development by small teams in the face of vague and changing requirements.

Extreme Programming is one of the Agile software development methodologies. It provides values and principles to guide the team behavior. The team is expected to self-organize. Extreme Programming provides specific core practices where −

- Each practice is simple and self-complete.
- Combination of practices produces more complex and emergent behavior.

39 **HOW Embrace Change happens in Extreme programming?**

A key assumption of Extreme Programming is that the cost of changing a program can be held mostly constant over time.

This can be achieved with −

- Emphasis on continuous feedback from the customer
- Short iterations
- Design and redesign
- Coding and testing frequently
- Eliminating defects early, thus reducing costs
- Keeping the customer involved throughout the development
- Delivering working product to the customer

40 **How Extreme Programming usedin a Nutshell?**

Extreme Programming involves −

- Writing unit tests before programming and keeping all of the tests running at all times. The unit tests are automated and eliminate defects early, thus reducing the costs.
- Starting with a simple design just enough to code the features at hand and redesigning when required.
- Programming in pairs (called pair programming), with two programmers at one screen, taking turns to use the keyboard. While one of them is at the keyboard, the other constantly reviews and provides inputs.
- Integrating and testing the whole system several times a day.

41 **Why is it called "Extreme?**

Extreme Programming takes the effective principles and practices to extreme levels.

- Code reviews are effective as the code is reviewed all the time.
- Testing is effective as there is continuous regression and testing.
- Design is effective as everybody needs to do refactoring daily.
- Integration testing is important as integrate and test several times a day.
- Short iterations are effective as the planning game for release planning and iteration planning.

42

What are the Extreme Programming Advantages?

Extreme Programming solves the following problems often faced in the software development projects −

- **Slipped schedules** − and achievable development cycles ensure timely deliveries.
- **Cancelled projects** − Focus on continuous customer involvement ensures transparency with the customer and immediate resolution of any issues.
- **Costs incurred in changes** − Extensive and ongoing testing makes sure the changes do not break the existing functionality. A running working system always ensures sufficient time for accommodating changes such that the current operations are not affected.
- **Production and post-delivery defects: Emphasis is on** − the unit tests to detect and fix the defects early.

43

What is Scrum ?

The Scrum approach is a general agile method but its focus is on managing iterative development rather than specific agile practices. There are three phases in Scrum:

1. The initial phase is an outline planning phase where you establish the general objectives for the project and design the software architecture.
2. This is followed by a series of **sprint** cycles, where each cycle develops an increment of the system.
3. The project closure phase wraps up the project, completes required documentation such as system help frames and user manuals and assesses the lessons learned from the project.

44 **What are the** Advantages **of scrum ?**

- The product is broken down into a set of **manageable and understandable chunks**.
- Unstable requirements do not hold up **progress**.
- The whole team has visibility of everything and consequently **team communication** is improved.
- Customers see **on-time delivery** of increments and gain feedback on how the product works.
- **Trust** between customers and developers is established and a positive culture is created in which everyone expects the project to succeed.

45. **Mention the Two perspectives on scaling of agile methods?**

1. **Scaling up**
2. **Scaling out**

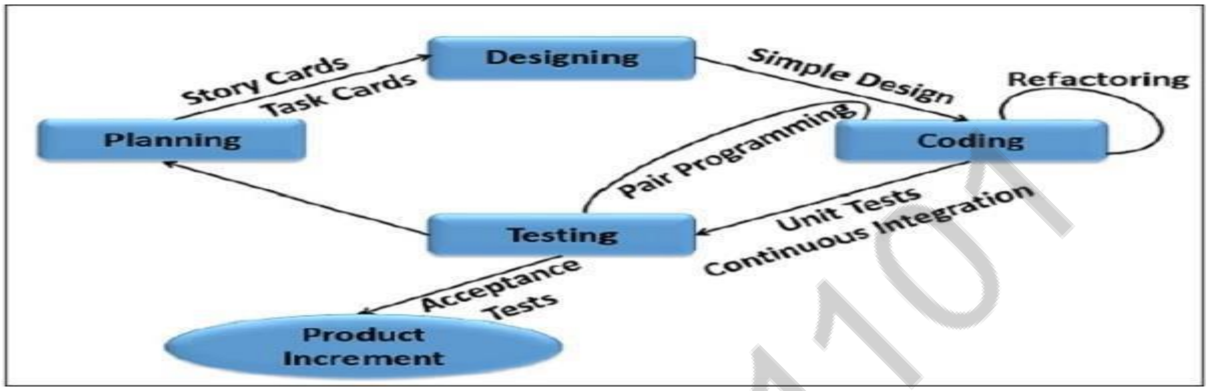
46. **What is Scaling up**

Using agile methods for developing large software systems that cannot be developed by a small team. For large systems development, it is not possible to focus only on the code of the system; you need to do more up- front design and system documentation. Cross-team communication mechanisms have to be designed and used, which should involve regular phone and video conferences between team members and frequent, short electronic meetings where teams update each other on progress. Continuous integration, where the whole system is built every time any developer checks in a change, is practically impossible; however, it is essential to maintain frequent system builds and regular releases of the system.

47. **What is Scaling out.**

How agile methods can be introduced across a large organization with many years of software development experience. Project managers who do not have experience of agile methods may be reluctant to accept the risk of a new approach. Large organizations often have quality procedures and standards that all projects are expected to follow and, because of their bureaucratic nature, these are likely to be incompatible with agile methods. Agile methods seem to work best when team members have a relatively high skill level. However, within large organizations,
there are likely to be a wide range of skills and abilities.

48. **Draw the diagram of Extreme programming?**



49 **What is agile development?**

Specification, design, implementation and testing are inter-leaved and the outputs from the development process are decided through a process of negotiation during the software development process.

projects include elements of plan-driven and agile processes. Deciding on the balance depends on many technical, human, and organizational issues.

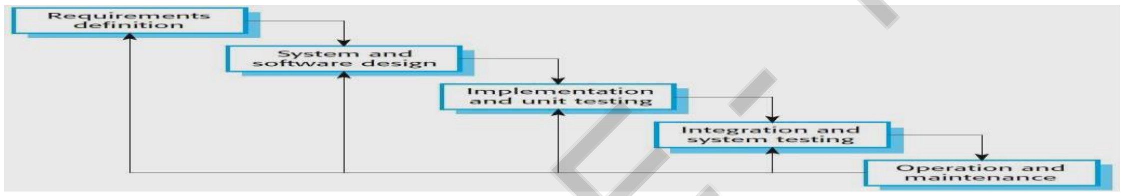
50. **What is Scrum master?**

The role of the Scrum Master is to protect the development team from external distractions. At the end of the sprint the work done is reviewed and presented to stakeholders (including the product owner).

PART –B

1 **Explain the following: (i) waterfall model (ii) Spiral model (iii)RAD model (iv) Prototyping model.** **NOV/DEC-12, NOV/DEC-15,**

- A Project management methodology based on a sequential design process
- Finishes one phase before another phase can begin
- SDLC Model
- Linear Sequential Model
- Simple to understand and easy to implement


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

Waterfall model phases

- There are separate identified phases in the waterfall model:

1. **Requirements analysis and definition**
2. **System and software design**
3. **Implementation and unit testing**
4. **Integration and system testing**
5. **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.

(ii)SPIRAL MODEL

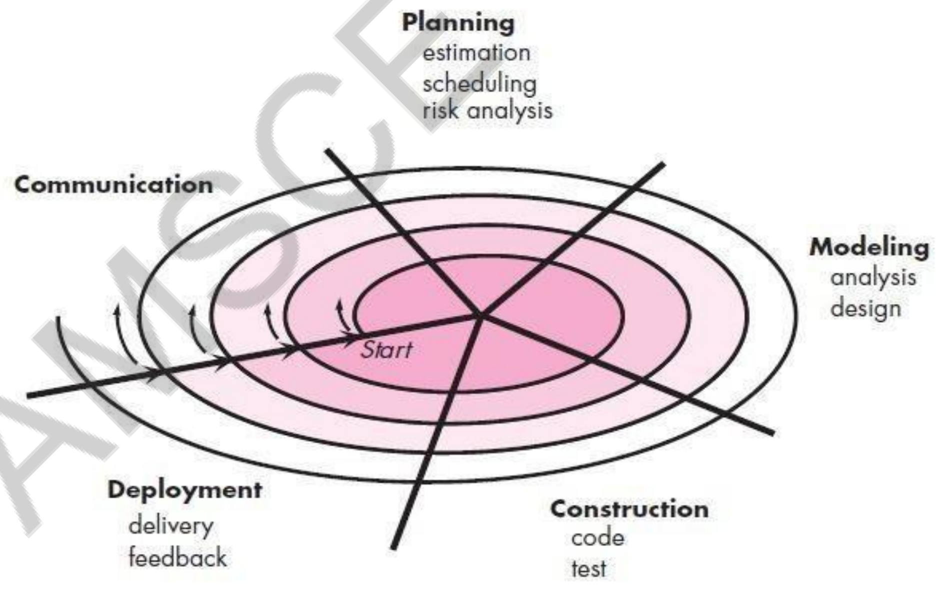
The spiral model is divided into number of frame works. These frameworks are denoted by task regions.

Usually there are six task regions. In spiral model project entry point axis is defined.

The task regions are:

1. Customer communication ,Planning Risk analysis., Engineering, Construct and release and Customer evaluation.

- Drawbacks
  1. It is based on customer communication.
  2. It demands considerable risk assessment.



It was originally proposed by Barry Boehm, the spiral model is anevolutionary software process model that couples the iterative nature of prototyping with the controlled and systematic aspects of the waterfallmodel.

- It provides the potential for rapid development of increasingly more complete versions of the software.
- The spiral model can be adopted to apply throughout the entire lifecycle of the application from concept development to maintenance.
- The spiral model is divided into set of framework activities defined bysoftware engineering team.
- The initial activity is shown from centre and developed inclockwise direction.

Advantages

- In this approach, the project monitoring is easy and more effective compared to other models.
- It reduces the number of risk in software development before they become serious problem.
- Suitable for very high risk.
- Schedule and cost is more realistic.
- Risk management is in-built in the spiral model.
- Changes can be accommodated in the later
- stages

(iii) RAD Model

RAD or Rapid Application Development process is an adoption of the waterfall model; it targets at developing software in a short span of time. RAD follows the iterative

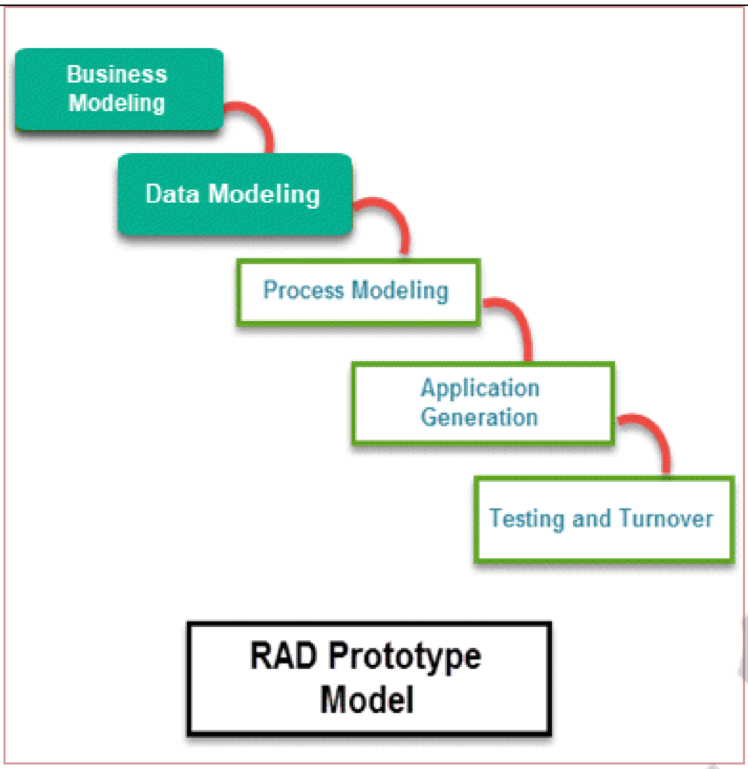
SDLC RAD model has following phases

Business Modeling
Data Modeling

Process Modeling

. Application Generation

viii. Testing and Turnover



iv) Prototyping Model

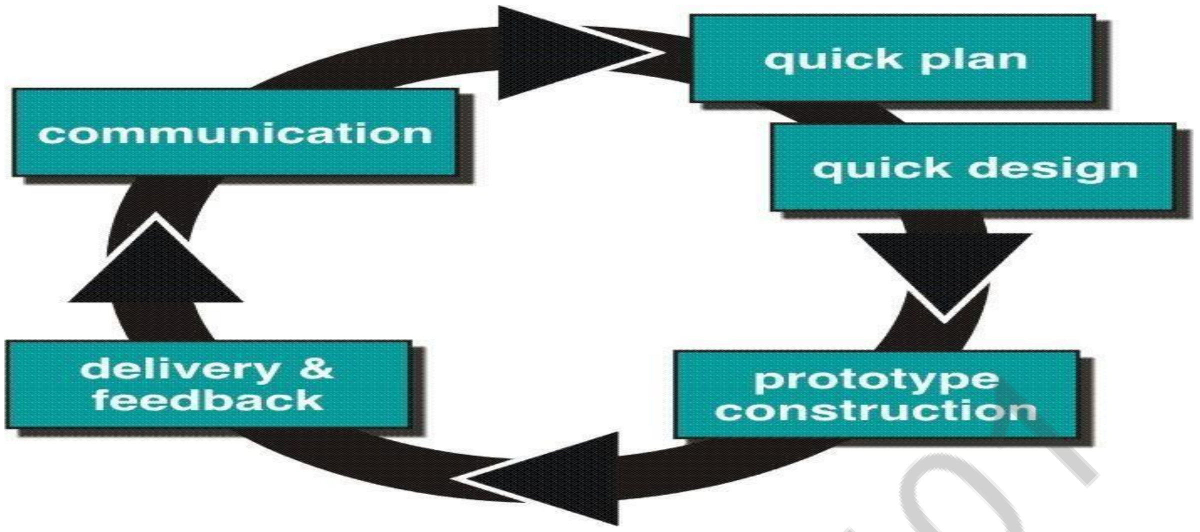
Prototype methodology is defined as a Software Development model in which a prototype is built, test, and then reworked when needed until an acceptable prototype is achieved. It also creates a base to produce the final system.

Software prototyping model works best in scenarios where the project's requirements are not known. It is an iterative, trial, and error method which take place between the developer and the client

Often, a customer defines a set of general objectives for software, but does not identify detailed requirements for functions and features.

- In this case Prototyping is best suited
- Prototyping can be used together with other models for elicitation requirements
- The prototype can serve as "the first system."
- Some prototypes are "Throw Away" while others also evolve become part of the actual system.
- Both customers and developers like the prototyping paradigm.

i. Customer/End user gets a feel for the actual system

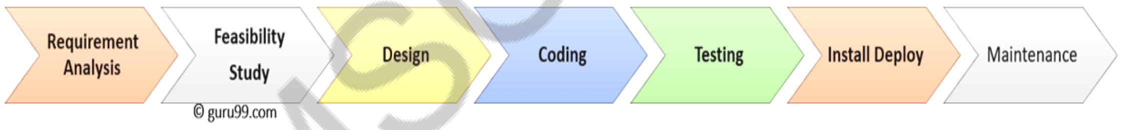


ii. Developer get to build something immediately.

2 Discuss the various life cycle models in software development? APR/MAY-16

- The **Software Development Lifecycle (SDLC)** is a systematic process for building software that ensures the quality and correctness of the software built.
- SDLC process aims to produce high-quality software which meets customer expectations.
- The system development should be complete in the pre-defined time frame and cost.

DLC consists of a detailed plan which explains how to plan, build, and maintain specific software. Every phase of the SDLC lifecycle has its own process and deliverables that feed into the next phase.



- Phase 1: Requirement collection and analysis
- Phase 2: Feasibility study:
- Phase 3: Design:
- Phase 4: Coding:
- Phase 5: Testing:
- Phase 6: Installation/Deployment:
- Phase 7: Maintenance:

Phase 1: Requirement collection and analysis:

The requirement is the first stage in the SDLC process. It is conducted by the senior team members with inputs from all the stakeholders and domain experts in the industry. Planning for the quality assurance requirements and recognization of the risks involved is also done at this stage.

This stage gives a clearer picture of the scope of the entire project and the anticipated issues, opportunities, and directives which triggered the project.

Requirements Gathering stage need teams to get detailed and precise requirements. This helps companies to finalize the necessary timeline to finish the work of that system.

Phase 2: Feasibility study:

Once the requirement analysis phase is completed the next step is to define and document software needs. This process conducted with the help of 'Software Requirement Specification' document also known as 'SRS' document. It includes everything which should be designed and developed during the project life cycle.

There are mainly five types of feasibilities checks:

- **Economic:** Can we complete the project within the budget or not?
- **Legal:** Can we handle this project as cyber law and other regulatory framework/compliances.
- **Operation feasibility:** Can we create operations which is expected by the client?
- **Technical:** Need to check whether the current computer system can support the software
- **Schedule:** Decide that the project can be completed within the given schedule or not.

Phase 3: Design:

In this third phase, the system and software design documents are prepared as per the requirement specification document. This helps define overall system architecture.

This design phase serves as input for the next phase of the model.

There are two kinds of design documents developed in this phase:

High-Level Design (HLD)

- Brief description and name of each module
- An outline about the functionality of every module
- Interface relationship and dependencies between modules
- Database tables identified along with their key elements
- Complete architecture diagrams along with technology details

Low-Level Design(LLD)

- Functional logic of the modules

- Database tables, which include type and size
- Complete detail of the interface
- Addresses all types of dependency issues
- Listing of error messages
- Complete input and outputs for every module

Phase 4: Coding:

Once the system design phase is over, the next phase is coding. In this phase, developers start build the entire system by writing code using the chosen programming language. In the coding phase, tasks are divided into units or modules and assigned to the various developers. It is the longest phase of the Software Development Life Cycle process.

In this phase, Developer needs to follow certain predefined coding guidelines. They also need to use programming tools like compiler, interpreters, debugger to generate and implement the code.

Phase 5: Testing:

Once the software is complete, and it is deployed in the testing environment. The testing team starts testing the functionality of the entire system. This is done to verify that the entire application works according to the customer requirement.

During this phase, QA and testing team may find some bugs/defects which they communicate to developers. The development team fixes the bug and send back to QA for a re-test. This process continues until the software is bug-free, stable, and working according to the business needs of that system.

Phase 6: Installation/Deployment:

Once the software testing phase is over and no bugs or errors left in the system then the final deployment process starts. Based on the feedback given by the project manager, the final software is released and checked for deployment issues if any.

Phase 7: Maintenance:

Once the system is deployed, and customers start using the developed system, following 3 activities occur

- ■ Bug fixing - bugs are reported because of some scenarios which are not tested at all
- ■ Upgrade - Upgrading the application to the newer versions of the Software
- ■ Enhancement - Adding some new features into the existing software

3 **What is the difference between information engineering & product engineering? Also explain the product engineering hierarchy in detail.** <u>**MAY/JUN-13**</u>

Product engineering

- It refers to the process of designing and developing a device, assembly, or system such that it is produced as an item for sale through some production manufacturing process.
- Product engineering usually entails activity dealing with issues of cost, producibility, quality, performance, reliability, serviceability, intended lifespan and user features.
- These product characteristics are generally all sought in the attempt to make the resulting product attractive to its intended market and a successful contributor to the business of the organization that intends to offer the product to that market.
- It includes design, development and transitioning to manufacturing of the product. The term encompasses developing the concept of the product and the design and development of its mechanical, electronics and software components.
- After the initial design and development is done, transitioning the product to manufacture it in volumes is considered part of product engineering.
- Product engineers are the technical interface between the component development team and the production side (Front End and Back End), especially after the development phase and qualifications when the high volume production is running.
- Product engineers improve the product quality and secure the product reliability by balancing the cost of tests and tests coverage that could impact the production fall-off. They support failure analysis request from customers.
- For example, the engineering of a digital camera would include defining the feature set, design of the optics, the mechanical and ergonomic design of the packaging, developing the electronics that control the various component and developing the software that allows the user to see the pictures, store them in memory and download them to a computer.
- Product engineering is an engineering discipline that deals with both design and manufacturing aspects of a product.

The job requires the product engineer to have a very good working knowledge of:

- Statistical methods and tools
- Manufacturing process
- Software, hardware and systems implementation
- Product reliability and qualification
- Physical analysis methods
- Computer-aided design and simulation programs
- Specific technology
- Strong product Knowledge

- Strong analytic work methodology and problem solving skills
- Continuous Improvement Knowledge

Information engineering (**IE**)

- It also known as **Information technology engineering** (**ITE**), **information engineering methodology** (**IEM**) or **data engineering**, is a software engineering approach to designing and developing information systems.
- Information technology engineering involves an architectural approach for planning, analyzing, designing, and implementing applications.
- It has been defined by Steven M Davis as: "An integrated and evolutionary set of tasks and techniques that enhance business communication throughout an enterprise enabling it to develop people, procedures and systems to achieve its vision

There are two variants of information technology engineering. These are called the DP-driven variant and the business-driven variant.

- **DP-driven:** The DP-driven variant of information technology engineering was designed to enable IS Departments to develop information systems that satisfied the information needs of the 1980s - which was largely a DP-driven development environment. Most of the CASE tools available today support this DP-driven variant of ITE.

i. **Business-driven:** ITE was extended into strategic business planning for the business-driven variant of information technology engineering. This variant was designed for rapid change in the client/server, object-oriented environment of the business-driven 1990's.

BTL6

4 (a) List the principles of agile software development. NOV/DEC 2016

1. ***It describes a set of principles for software developmentwhere,***

Requirements and solutions evolve through the collaborative effort of self-**organizing + cross-functional teams**

It advocates

Adaptive planning

1. Evolutionary development
2. Early delivery
3. Continuous improvement
4. Encourages rapid and flexible response to change

These principles support the definition and continuing evolution of many software development methods

Every project needs to be **handled differently**

Existing methods need to be tailored to best suit the project requirements

Tasks are **divided to time boxes** (small time frames) to **deliver specific features for a release**

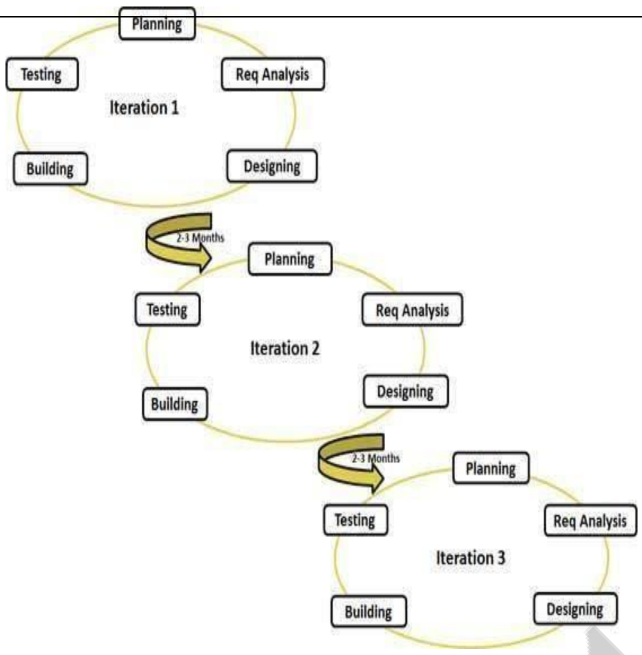
Iterative approach is taken & working software build is delivered after each iteration

Agile Model Pros andCons

It widely accepted in the software world recently, however, this method may not always be suitable for all products.

(k) **Pros**	(l) **Cons**
Is a very realistic approach to software development Promotes teamwork and cross training. Functionality can be developed rapidly anddemonstrated. Resource requirements are minimum. Suitable for fixed or changing requirements Delivers early partial workingsolutions. Good model for environments that change steadily. Minimal rules, documentation easily employed. Enables concurrent development and delivery within an overall planned context. Little or no planningrequired Easy to manage Gives flexibility to developers	Not suitable for handling complex dependencies. More risk of sustainability, maintainability and extensibility. An overall plan, an agile leader and agile PM practice is a must without which it will not work. Strict delivery management dictates the scope, functionality to be delivered, and adjustments to meet the deadlines. Depends heavily on customer interaction, so if customer is not clear, team can be driven in the wrongdirection. There is very high individual dependency, since there is minimum documentation generated. Transfer of technology to new team members may be quite challenging due to lack of documentation.

)



Each **build is incremental** in terms of features

Final build holds all the features required by the customer

Agile Principles

The Agile Manifesto is based on twelve principles

1. Customer satisfaction by early and continuous delivery of valuable software
2. Welcome changing requirements, even in late development
3. Working software is delivered frequently (weeks rather than months)
4. Close, daily cooperation between business people and developers
5. Projects are built around motivated individuals, who should be trusted
6. Face-to-face conversation is the best form of communication (co-location)
7. Working software is the principal measure of progress
8. Sustainable development, able to maintain a constant pace
9. Continuous attention to technical excellence and good design
10. Simplicity—the art of maximizing the amount of work not done—is essential
11. Best architectures, requirements, and designs emerge from self-organizing teams
12. Regularly, the team reflects on how to become more effective, and adjusts accordingly

5 **Write note on business process engineering and product engineering? <u>MAY/JUN-13 , APRIL/MAY-15</u>**

Business Engineering

Business process engineering is a way in which organizations study their current business processes and develop new methods to improve productivity, efficiency, and operational costs.

As a business process engineer, you will examine the way an organization operates, its long-term performance goals, and recommend ways it can work more seamlessly to achieve overall improvement.

. Many business process engineers work as consultants contracted by companies seeking improvements to their methodology and infrastructure.

Product engineering

4. It refers to the process of designing and developing a device, assembly, or system such that it is produced as an item for sale through some production manufacturing process.
5. Product engineering usually entails activity dealing with issues of cost, reducibility, quality, performance, reliability, serviceability, intended lifespan and user features.
6. These product characteristics are generally all sought in the attempt to make the resulting product attractive to its intended market and a successful contributor to the business of the organization that intends to offer the product to that market.
7. It includes design, development and transitioning to manufacturing of the product. The term encompasses developing the concept of the product and the design and development of its mechanical, electronics and software components.
8. After the initial design and development is done, transitioning the product to manufacture it in volumes is considered part of product engineering.
9. Product engineers are the technical interface between the component development team and the production side (Front End and Back End), especially after the development phase and qualifications when the high volume production is running.
10. Product engineers improve the product quality and secure the product reliability by balancing the cost of tests and tests coverage that could impact the production fall-off. They support failure analysis request from customers.
11. For example, the engineering of a digital camera would include defining the feature set, design of the optics, the mechanical and ergonomic design of the packaging, developing the electronics that control the various component and developing the software that allows the user to see the pictures, store them in memory and download them to a computer.
12. Product engineering is an engineering discipline that deals with both design and manufacturing aspects of a product.

The job requires the product engineer to have a very good working knowledge of:

ii. Statistical methods and tools

iii. Manufacturing process

iv. Software, hardware and systems implementation

v. Product reliability and qualification

vi. Physical analysis methods

vii. Computer-aided design and simulation programs

viii. Specific technology

ix. Strong product Knowledge

x. Strong analytic work methodology and problem solving skills

xi. Continuous Improvement Knowledge

6 Explain in detail about spiral model with a neat sketch and describe why this model comes under both evolutionary and RAD models. **<u>APRIL/MAY-15, NOV/DEC 2017</u>**

REFER PART B Q1

7 Which process model is best suited for risk management? Discuss in detail with an example. Give its advantages and disadvantages? **<u>NOV/DEC 2016,APRIL/MAY 2018</u>**

There are two characteristics of risk i.e. uncertainty and loss.

Following are the categories of the risk:

1. Project risk

1. If the project risk is real then it is probable that the project schedule will slip and the cost of the project will increase.
2. It identifies the potential schedule, resource, stakeholders and the requirements problems and their impact on a software project.

2. Technical risk

- If the technical risk is real then the implementation becomes impossible.
- It identifies potential design, interface, verification and maintenance of the problem.

3. Business risk

If the business risk is real then it harms the project or product.

There are five sub-categories of the business risk:

1. Market risk - Creating an excellent system that no one really wants.
2. Strategic risk - Creating a product which no longer fit into the overall business strategy for companies.
3. Sales risk - The sales force does not understand how to sell a creating product.
4. Management risk - Loose a support of senior management because of a change in focus.
5. Budget risk - losing a personal commitment.

Other risk categories

These categories suggested by Charette.

1. Known risks : These risk are unwrapped after the project plan is evaluated.
2. Predictable risks : These risks are estimated from previous project experience.
3. Unpredictable risks : These risks are unknown and are extremely tough to identify in advance.

Principles of risk management

Maintain a global perspective - View software risks in the context of a system and the business problem planned to solve.

Take a forward looking view – Think about the risk which may occur in the future and create future plans for managing the future events.

Encourage open communication – Encourage all the stakeholders and users for suggesting risks at any time.

Integrate – A consideration of risk should be integrated into the software process.

Emphasize a continuous process – Modify the identified risk than the more information is known and add new risks as better insight is achieved.

Develop a shared product vision – If all the stakeholders share the same vision of the software then it is easier for better risk identification.

Encourage teamwork – While conducting risk management activities pool the skills and experience of all stakeholders.

Risk Identification

It is a systematic attempt to specify threats to the project plans.

Two different types of risk:

1. Generic risks
o These risks are a potential threat to each software project.
2. Product-specific risks
- These risks are recognized by those with a clear understanding of the technology, the people and the environment which is specific to the software that is to be built.
- A method for recognizing risks is to create item checklist.

8 **Consider 7 functions with their estimated lines of code. Average productivity based on historical data is 620 LOC/pm and labour rate is Rs. 8000 per mnth. Find the total estimates project cost and effort? F1 – 2340 , F2 – 5380, F3 – 6800 , F4 –3350 , F5 -4950 , F6 -2140 , F7 – 8400**

There are many techniques that can be used to rigorously estimate or measure effortand cost for a software project, such as:

-Function Point (FP)

-Source Lines of Code (SLOC).

-COnstructive COst MOdel (COCOMO)

-Delphi

SLOC Technique(Source Line of Code Technique)-The SLOC technique is an objective method of estimating or calculating the size of the project.

-The project size helps determine the resources, effort, cost, and duration required to complete the project.

-It is also used to directly calculate the effort to be spent on a project.

-W e can use it when the programming language and the technology to be used are predefined.

-This technique includes the calculation of lines of codes(LOC), documentation of pages, inputs, outputs, and components of a software program.

Counting SLOC-The use of SLOC techniques can be used in the case of the technology or language remains unchanged throughout the project.

Generally, it can be used when you are using third-generation language, such as FORTRAN or COBOL.

-To count the SLOC the following must be considered: The count includes:-

The SLOC delivered to client.

-The SLOC written only by the development team are counted-The declaration statements are counted as source lines of code

The count excludes:-The code created by application generators.

-The comments inserted to improve the readability of program.

-Once, you get the numbers of line of code of SLOC, you can estimate or calculate the total effort and cost to complete the given project.

Example:-Assume estimated lines of code of a system is: 33,600 LOC -Average productivity for system of this type is: 620 LOC/person-month-

There are 7 developers-Labor rate is: $ 8000per person-month Calculate the total effort and cost required to complete the above project

Solution+Way1=>

Total Effort =Total LOC/Productivity = 33600/620=54.19 ≈ 54 person-months=> 7developers

Effort = Total Effort/6= 54/7= 7months=> Total Cost=Total Effort * Labor Rate = 54 * 8000≈ $43,2000+Way2=> Cost per LOC =Labor Rate/Productivity=8000/620=$12.9≈ $13

> Total Cost = Total LOC * Cost perLOC =33,600* 13=$436800

9 (i) What is the impact of reusability in software development process?

(ii) Explain the component based software development model with a neat sketch.

NOV/DEC 2017

Component-based software engineering (**CBSE**)

- It also called **components-based development** (**CBD**), is a branch of software engineering that emphasizes the separation of concerns with respect to the wide-ranging functionality available throughout a given software system.
- It is a reuse-based approach to defining, implementing and composing loosely coupled independent components into systems.
- This practice aims to bring about an equally wide-ranging degree of benefits in both the short-term and the long-term for the software itself and for organizations that sponsor such software.
- Software engineering practitioners regard components as part of the starting platform for service-orientation.
- Components play this role, for example, in web services, and more recently, in service-oriented architectures (SOA), whereby a component is converted by the web service into a *service* and subsequently inherits further characteristics beyond that of an ordinary component.
- COTS(Commercial Off The Shelf) software components, developed by vendors who offer them as products can be used when software is to built.
- Provides interfaces targeted functionality with well defined
- Incorporates model many of the characteristics of Spiral model
- Regardless of technology to be used, it must follow the steps like– Available component based products are researched and evaluated for the current application

  Component integration issues is to dealt

  – A software architecture is designed to accommodate the components.

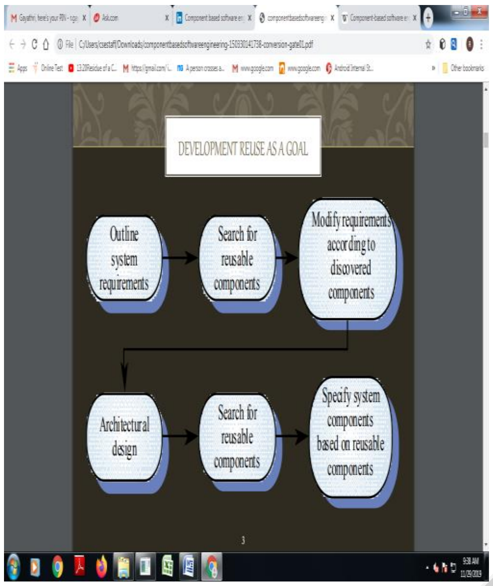
  – Components are integrated into the architecture

  – Comprehensive testing is conducted to ensure proper functionality.

  – Component Based Software Engineering(CBSE) is a process that emphasis the design and construction of computer based system using reusable software "components".

• It emerged from the failure of object-oriented development to support effective reuse. Single object classes are too detailed and specific.

• CBSE embodies the “buy , don’t built” philosophy.


DEVELOPMENT REUSE AS A GOAL
Outline system requirements
Search for reusable components
Modify requirements according to discovered components
Architectural design
Search for reusable components
Specify system components based on reusable components

- The process begins when a software team establish requirements for a system to be built using conventional requirements elicitation techniques. An architectural design is established , but rather than moving immediately into more detailed tasks , the team examines requirements to determine what subset is directly amenable to composition , rather than construction.
- For those requirements that can be addressed with available components the following activities take place:

1. Component qualification
2. Component adaptation
3. Component composition
4. Component update

COMPONENT CHARACTERISTICS

1. Standardised
2. Independent
3. Compassable
4. Deployable

5. Documented

10 **(i) How function point analysis methodology is applied in estimation of software size ?Explain. Why FPA methodology is better than LOC methodology ?**

If LOC is simply a count of the number of lines then figure shown below contains 18 LOC **.**

(ii) A line of code is any line of program text that is not a comment or blank line, regardless of the number of statements or fragments of statements on the line. This specifically includes all lines containing program header, declaration, and executable and non-executable statements".

Function Count

Alan Albrecht while working for IBM, recognized the problem in size measurement in the 1970s, and developed a technique (which he called Function Point Analysis), which appeared to be a solution to the size
measurement problem.

The five functional units are divided in two categories:

(i) Data function types

# Internal Logical Files (ILF): A user identifiable group of logical related data or control information maintained within the system.

# External Interface files (EIF): A user identifiable group of logically related data or control information referenced by the system, but maintained within another system. This means that EIF counted for one system, may be an ILF in another system.

The weighting factors are identified for all functional units and multiplied with the functional units accordingly. The procedure for the calculation of Unadjusted Function Point (UFP) is given in table shown above.

he procedure for the calculation of UFP in mathematical

(iii) form is given below:UFP = ∑∑ Z ij w ij

i = 1 J = 1

Where i indicate the row and j indicates the column of Table 1

W ij : It is the entry of the i th row and j th column of the table 1

Zij : It is the count of the number of functional units of Type i that have been classified as having the complexity corresponding to column j.

Organizations that use function point methods develop a criterion for determining whether a particular entry is Low, Average or High.

Nonetheless, the determination of complexity is somewhat subjective.

FP = UFP * CAF

Where CAF is complexity adjustment factor and is equal to [0.65 + 0.01 x ΣF i]. The F i (i=1 to 14) are the degree of influence

(iv) An application has the following:10 low external inputs, 12 high external outputs, 20 low internal logical files, 15 high external interface files, 12 average external inquiries and a value adjustment factor of 1.10 . What is the unadjusted and adjusted function point count ? APRIL/MAY 2017

Solution

Unadjusted function point counts may be calculated using as:

UFP = ∑∑ Z ij w ij
i = 1 J = 1

FP
= 10 x 3 + 12 x 7 + 20 x 7 + 15 + 10 + 12 x 4
= 30 + 84 +140 + 150 + 48
= 452
= UFP x CAF
= 452 x 1.10 = 497.2.

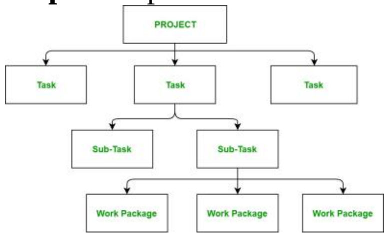
11 What is a process model ? Describe the process model that you would choose to manufacture a car. Explain giving suitable reasons. APRIL/MAY 2017

A structured set of activities required to develop a software system.
• Many different software processes but all involve:
•Specification – defining what the system should do;
Design and implementation – defining the organization of the system and implementing the system;
Validation – checking that it does what the customer wants;
Evolution – changing the system in response to changing customer needs.
• A software process model is an abstract representation of a process. It presents a description of a process from some particular perspective
When we describe and discuss processes, we usually talk about the activities in these processes such as specifying a data model, designing a user interface, etc. and the ordering of these activities.
• Process descriptions may also include:
•Products, which are the outcomes of a process activity;
Roles, which reflect the responsibilities of the people involved in the process;
Pre- and post-conditions, which are statements that are true before and after a process activity has been enacted or a product produced.

12 Explain how breakdown structure is used in software engineering .Discuss how software project scheduling helps in timely release of a product. APRIL/MAY 2018

A **Work Breakdown Structure** includes dividing a large and complex project into simpler, manageable and independent tasks. The root of this tree (structure) is labelled by the Project name itself. For constructing a work breakdown structure, each node is recursively decomposed into smaller sub-activities, until at the leaf level, the activities becomes undividable and independent. It follows a Top-Down approach.

Steps:

3. **Step-1:** Identify the major activities of the project.
4. **Step-2:** Identify the sub-activities of the major activities.
5. **Step-3:** Repeat till undividable, simple and independent activities are created.
6. 

Construction of Work Breakdown Structure:

Firstly, the project managers and top level management identifies the main deliverables of the project. After this important step, these main deliverables are broke down into smaller higher-level tasks and this complete process is done recursively to produce much smaller independent tasks. It depends on the project manager and team that upto which level of detail they want to break down their project. Generally the lowest level tasks are the most simplest and independent tasks and takes less than two weeks worth of work. Hence, there is no rule for upto which level we may build the work breakdown structure of the project as it totally depends upon the type of project we are working on and the management of the company. The efficiency and success of the whole project majorly depends on the quality of the Work Breakdown Structure of the project and hence, it implies its importance.

Uses:

- It allows doing a precise cost estimation of each activity.
- It allows estimating the time that each activity will take more precisely.
- It allows easy management of the project.
- It helps in proper organization of the project by the top management.

13 Give detail explanation about agile process?

Combination of iterative and incremental process models

 Focus of adaptability and customer satisfaction

 Break into small incremental builds

 iteration typically lasts 1-3 weeks

 Cross functional teams working

 End of the iteration, a working product is displayed to the customer

Advantage

 Realistic approach

 Promotes teamwork and cross training.

 Functionality developed rapidly and demonstrated.

 Resource requirements are minimum.

 Suitable for fixed or changing requirements

 Delivers early partial working solutions.

 Good model for environments that change steadily.

 Minimal rules, documentation easily employed.

 Little or no planning required.

 Easy to manage.

 Gives flexibility to developers.

 Not suitable for handling complex dependencies.

 Strict delivery management dictates the scope, functionality to be delivered, and adjustments to meet the deadlines.

 Depends heavily on customer interaction, so if customer is not clear, team can be driven in the wrong direction.

 Transfer of technology to new team members may be quite challenging due to lack of documentation.

Agile Framework

 Rational Unified Process (1994),

 Scrum (1995),

 Extreme Programming (1996),

 Adaptive Software Development,

 Feature Driven Development,

 Dynamic Systems Development Method (DSDM) (1995).

 Rational Unified Process (1994),

 Scrum (1995),

 Extreme Programming (1996),

 Adaptive Software Development,

 Feature Driven Development,

 Dynamic Systems Development Method (DSDM) (1995).

14 Describe in detail about Extreme programming ?

ExtremeProgramming(XP)

- **Management-Practices**

  On-SiteCustomer: Acentral customer contact must alwaysbe accessiblein order to clarify requirements and questions directly.

  Planning Game: Projects, in accordancewith XP,runiteratively (repeatedly) and incrementally (gradually build on eachother). The contentsof the next steparеplannedbefore each iteration. Allproject members(incl. the customer)participate.

  ShortReleases: Newdeliveries shouldbemadeat short intervals. Consequently,customers receive the required functions quickerandcantherefore givefeedbackon the developmentquicker.

- **Team-Practices**

  Metaphor: Only a few clear metaphors should describethe system beingdevelopedsothat the nitty-gritty of the systemis clearto all of the projectmembers.

  CollectiveOwnership: The whole team is responsible for the system, not individuals. Eachdeveloper musthaveaccessto all lines of code so that eachdeveloperis ableto takeover the taskof another developer.

- **ContinuousIntegration:** All changesto the systemareintegrated

  Promptly so that not too many dependencies between changes occur.

  CodingStandards: Regardingthe commonresponsibility for the code, there shouldbeagivencommonstandardfor writing the code.

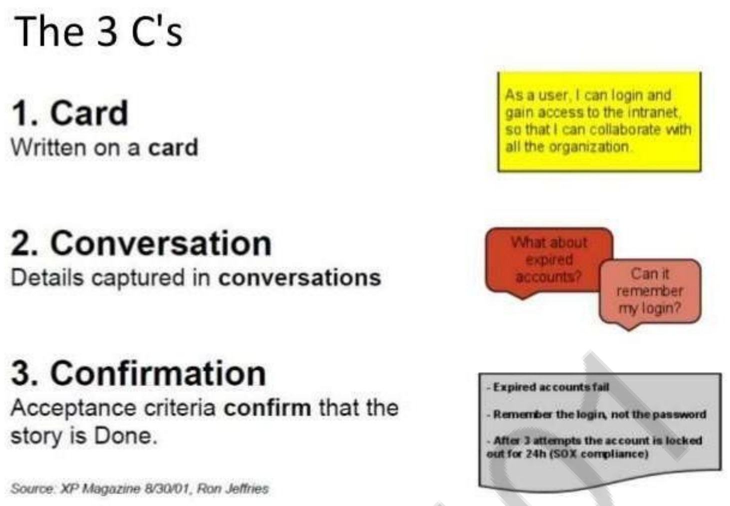
  Sustainable Pace: XP builds on the creativity of the individual project members. This creativity cannot be achieved if the project team constantlyworksovertime. Overtime isto be avoided.

- Design

  Writing unit tests before programming and keepingallof the tests runningatalltimes.Theunit testsareautomatedandeliminates defects early,thus reducingthecosts.

  Startingwith asimpledesignjust enoughtocode the features at hand and redesigning when required.

- **Design: UserStories**



The 3 C's

1. Card
Written on a **card**

2. Conversation
Details captured in **conversations**

3. Confirmation
Acceptance criteria **confirm** that the story is Done.

Source: XP Magazine 8/30/01, Ron Jeffries

As a user, I can login and gain access to the intranet, so that I can collaborate with all the organization.
What about expired accounts?
Can it remember my login?
- Expired accounts fail
- Remember the login, not the password
- After 3 attempts the account is locked out for 24h (SOX compliance)

•

- Development

  Programming in pairs (called pair programming), with two programmers at one screen, taking turns to use the keyboard. While one of them is at the keyboard, the other constantly reviews and provides inputs.

  Integrating and testing the whole system several times a day.

- PairProgramming

  two programmers work together at one workstation.

  One, the ***driver***, writes code while the other, the ***observer*** or ***navigator***, reviews each line of code as it is typed in.

  The two programmers switch roles frequently.

  While reviewing, the observer also considers the "strategic" direction of the work, coming up with ideas for improvements and likely future problems to address.

  This frees the driver to focus all of their attention on the "tactical" aspects of completing the current task, using the observer as a safety net and guide

- PairProgramming

  Pair programming increases the man-hours required to deliver code compared to programmers working individually from up to between 15% and 100%.

  However, the resulting code has about 15% fewer defects.

- Production

Putting a minimal working system into the production quickly and upgrading it whenever required.

Keeping the customer involved all the time and obtaining constant feedback.

Extreme Programming − A way to handle the common shortcomings

Software Engineering involves −

- Creativity
- Learning and improving through trials and errors
- Iterations

Extreme Programming builds on these activities and coding. It is the detailed (not the only) design activity with multiple tight feedback loops through effective implementation, testing and refactoring continuously.

Extreme Programming is based on the following values −

- Communication
- Simplicity
- Feedback
- Courage
- Respect

What is Extreme Programming?

XP is a lightweight, efficient, low-risk, flexible, predictable, scientific, and fun way to develop software.

e**X**treme **P**rogramming (XP) was conceived and developed to address the specific needs of software development by small teams in the face of vague and changing requirements.

Extreme Programming is one of the Agile software development methodologies. It provides values and principles to guide the team behavior. The team is expected to self-organize. Extreme Programming provides specific core practices where −

- Each practice is simple and self-complete.
- Combination of practices produces more complex and emergent behavior.

Embrace Change

A key assumption of Extreme Programming is that the cost of changing a program can be held mostly constant over time.

This can be achieved with −

- Emphasis on continuous feedback from the customer
- Short iterations
- Design and redesign
- Coding and testing frequently
- Eliminating defects early, thus reducing costs
- Keeping the customer involved throughout the development
- Delivering working product to the customer

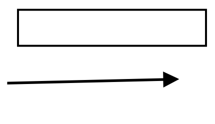
15 Explain about Extreme Programming using nutshell.?

Extreme Programming involves −

- Writing unit tests before programming and keeping all of the tests running at all times. The unit tests are automated and eliminates defects early, thus reducing the costs.
- Starting with a simple design just enough to code the features at hand and redesigning when required.
- Programming in pairs (called pair programming), with two programmers at one screen, taking turns to use the keyboard. While one of them is at the keyboard, the other constantly reviews and provides inputs.
- Integrating and testing the whole system several times a day.
- Putting a minimal working system into the production quickly and upgrading it whenever required.
- Keeping the customer involved all the time and obtaining constant feedback.

Iterating facilitates the accommodating changes as the software evolves with the changing requirements.

UNIT – 2 PART –A

S.NO	QUESTIONS
1	**What is Software Prototyping?** **<u>NOV/DEC-10 , APR/MAY-11, MAY/JUNE-13</u>** It is a rapid software development for validating the requirements. It is to help customers & developers to understand the system requirements.
2	**Define functional and non- Functional requirements.** **<u>NOV/DEC-10</u>** Functional requirements describe all the functionality or system services. It should be clear how system should react to particular inputs and how particular systems behave in particular situation. Non functional requirements define the system properties and constraints. It is divided in to product, organizational & external requirements.
3	**What is meant by functional requirement?** **<u>APR/MAY-11</u>** Functional requirements describe all the functionality or system services. It should be clear how system should react to particular inputs and how particular systems behave in particular situation.
4	**Name the metrics for specifying Non-functional requirements?** **<u>NOV/DEC-11</u>** Speed, size, ease of use, reliability, robustness, portability
5	**Draw the DFD for the following (i) External entity (ii) Data items** **<u>NOV/DEC-11</u>** External entity Data items 

6 **What do requirements processes involve? <u>APR/MAY-12</u>**

It involves feasibility study, discovery, analysis &validation of system requirements.

7 **Define non-functional requirements. <u>APR/MAY-12</u>**

Non functional requirements define the system properties and constraints. It is divided in to product, organizational &

external requirements

8 **Distinguish between the term inception, elicitation, & elaboration with reference to requirements? <u>NOV/DEC-12</u>**

Inception – set of questions are asked to establish basic understanding of problem.

Elicitation - collaborative requirements gathering & quality function deployment

Elaboration – It focuses on developing a refined technical model of software function, features & constraints.

9 **An SRS is traceable ?comment <u>NOV/DEC-12,MAY/JUNE 2016</u>** An SRS is correct if, and only if, every requirement stated therein is one that the software shall meet. Traceability makes this procedure easier and less prone to error.

10 **What is data dictionary? <u>MAY/JUN-13 , APR/MAY 2016 ,</u> <u>NOV/DEC 2016, APRIL/MAY 2017</u>**

It is organized collection of all the data elements of the system with precise and rigorous definition so that user & system analyst will have a common understanding of inputs, outputs, components of stores and intermediate calculations.

11 **What are the benefits of prototyping?**

i. Prototype serves as a basis for deriving system specification. ii. Design quality can be improved.

iii. System can be maintained easily.
iv. Development efforts may get reduced.

v. System usability can be improved.

12 **What are the prototyping approaches in software process?**<u>**MAY/JUNE 2016,APRIL/MAY 2018**</u>

i. Evolutionary prototyping – In this approach of system development, the initial prototype is prepared and it is then refined through number of stages to final stage.

ii. Throw-away prototyping – Using this approach a rough practical implementation of the system is produced. The requirement problems can be identified from this implementation. It is then discarded.System is then developed using some different engineering paradigm.

13 **List the characteristics of good SRS?** <u>**APR/MAY 2016**</u>

- Correct
- Unambiguous
- Complete
- Consistent
- Ranked for importance and/or stability
- Verifiable
- Modifiable
- Traceable

14 **Classify the following as functional / non-functional requirements for a banking system?** <u>**NOV / DEC 2016**</u>

(a) Verifying bank balance – **functional requirements**

(b) Withdrawing money from bank – **functionalrequirements**

(c) Completion of transaction in less than 1 sec – non**-functional requirements**

(d) Extending system by providing more tellers for customers -

non-functional requirements

15 **What is the linkage between Dataflow and ER diagram?APR/MAY 2016**

An ER diagram is the Entity Relationship Diagram, showing the relationship between different entities in a process. A Data Flow diagram is a symbolic structure showing how the flow of data is used in different process

16 **List the steps in user interface design? Golden rules of UI design APR/MAY 2015, NOV/DEC2015**

Place the User in Control

Reduce the User's Memory Load Make the Interface Consistent

17 **How are requirements validated?APR/MAY 2015 Requirements validation:** Have we got the requirements right?

In the validation phase, the work products produced as a consequence of requirements engineering are examined for consistency, omissions, and ambiguity. The basic objective is to ensure that the SRS reflects the actual requirements accurately and clearly.

18 **What is a state transition diagram?**

State transition diagram is basically a collection of states and events. The events cause the system to change its state. It also represents what actions are to be taken based on the transition.

19 **What is DFD?**

Data Flow Diagram depicts the information flow and the transforms that are applied on the data as it moves from input to output.

20 **What is waterfall model?**

The Waterfall Model was first Process Model to be introduced. It is also referred to as a **linear-sequential life cycle model**.

It is very simple to understand and use.

In a waterfall model, each phase must be completed fully before the next phase can begin. This type of model is basically used for the for the project which is small and there are no uncertain requirements.

In this model the testing starts only after the development is complete.

In **waterfall model phases** do not overlap.

21 What is ERD?

Entity Relationship Diagram is the graphical representation of the object relationship pair. It is mainly used in database applications.

22 What is data modeling?

Data modeling is the basic step in the analysis modeling. In data modeling the data objects are examined independently of processing. The data model represents how data are related with one another.

23 What is requirement engineering?

Requirement engineering is the process of establishing the services that the customer requires from the system and the constraints under which it operates and is developed.

24 What are the various Rapid prototyping techniques? <u>April /May 2015</u>

i. Dynamic high level language development.

ii. Database programming.

iii. Component and application assembly.

25 What is data modeling?

Data modeling is the basic step in the analysis modeling. In data modeling the data objects are examined independently of processing. The data model represents how data are related with one another.

26 What are the various types of traceability in software engineering? April/may 2018

iv. Source traceability – These are basically the links from requirement to stakeholders

v. Requirements traceability – These are links between dependant requirements.

vi. Design traceability – These are links from requirements to design.

27 What is cardinality in data modeling?

Cardinality in data modeling, cardinality specifies how the number of occurrences of one object is related to the number of occurrences of another object.

28 What are the objectives of Analysis modeling?

i. To describe what the customer requires.

ii. To establish a basis for the creation of software design.

iii. To devise a set of valid requirements after which the software can be built.

29 How the limitations of waterfall model overcome? April /May 2015

This type of model is basically used for the for the project which is small and there are no uncertain requirements.Where no overlapping of phases.

At the end of each phase, a review takes place to determine if the project is on the right path and whether or not to continue or discard the project.

30 What is feasibility study? NOV/DEC2015 , APR/MAY 2016

software feasibility has four solid dimensions:

Technology— Is a project technically feasible? Is it within the state of the art? Can defects be reduced to a level matching the application's needs?

Finance—Is it financially feasible? Can development be completed at a cost the software organization, its client, or the market can afford?

Time—Will the project's time-to-market beat the competition? Resources—Does the organization have the resources needed to succeed?

Before starting any project the feasibility study team ought to carry initial architecture and design of the high-risk requirements to the point at which it can answer these questions. In some cases, when the team gets negative answers, a reduction in requirements may be negotiated.

31 Define Quality function decelopment(QFD). NOV/DEC 2017

Quality Function Deployment (**QFD**) is a structured approach to defining customer needs or requirements and translating them into specific plans to produce products to meet those needs. The "voice of the customer" is the term to describe these stated and unstated customer needs or requirements.

32 Differentiate between normal and exciting requirements ? APRIL/MAY 2017

Normal requirements

- The objective and goal are stated for the system through the meetings with the customer.
- For the customer satisfaction these requirements should be there.

Exciting requirements

- These features are beyond the expectation of the customer.
- The developer adds some additional features or unexpected feature into the software to make the customer more satisfied.

For example, the mobile phone with standard features, but the developer adds few additional functionalities like voice searching, multi-touch screen etc. then the customer more exited about that feature.

33 **How do you design a software project for reuse? (Nov/Dec 2007)**

• A clear and well-defined product vision is an essential foundation to an software project.

• An evolutionary implementation strategy would be a more pragmatic strategy for the company.

• There exist a need for continuous management support and leadership to ensure success.

34 **What are the standards for documentation? Briefly explain (Nov/Dec 2007)**
IEEE Std 1028-2008
This standard defines five types of software reviews and procedures for their execution. Review types include management reviews, technical reviews, inspections, walk-throughs and audits. IEEE Std 1012-2004
This standard describes software verification and validation processes that are used to determine if software products of an activity meets the requirements of the activity and to determine if software satisfies the user's needs for the intended usage. The scope includes analysis, evaluation, review, inspection, assessment and testing of both products and processes.

35 **What are context free questions? How it differs from meta questions? (Nov/Dec 2009)**
Context free questions are questions that can be used regardless of the project under consideration. They are general questions about the nature of the project and the environment in which the final product will be used.Meta questions are very complex and detailed questions about the project model

36 **Define behaviouralmodelling(Nov/Dec 2012)**
All behavioural models really do is describe the control structure of a system. This can be things like:
- **Sequence of operations**
- **Object states**
- **and Object interactions**

Furthermore, this modelling layer can also be called Dynamic Modelling. The activity of creating a behavioural model is commonly known as behavioural modelling. As well as this, a system should also only have one behavioural model – much like functional modelling.

37 **what are the types of prototypes**
• Evolutionary prototyping – the initial prototype is prepared and it is then refined through number of stages to final stage.
• Throw-away prototyping – a rough practical implementation of the system is produced. The requirement problems can be identified from

this implementation

38 **Define behaviouralmodelling(Nov/Dec 2012)**
All behavioural models really do is describe the control structure of a system. This can be things like:
- **Sequence of operations**
- **Object states**
- **and Object interactions**

Furthermore, this modelling layer can also be called Dynamic Modelling. The activity of creating a behavioural model is commonly known as behavioural modelling. As well as this, a system should also only have one behavioural model – much like functional modelling.

39 **What is the major distinction between user requirement and system requirement? (April/May 2008)**
User requirements may be a set of statements or use case scenarios presented by the client in layman’s terms of which the client can easily

elaborate and are
usually free of technical jargon. System requirements are built from the clients input being what they have specified in the user requirements.

40 **Which style of prototyping is most appropriate when the requirement are not well-understood? (April/May 2008)**

User Interface prototyping is most appropriate.This prototyping is used to prespecify the look and feel of user interface in an effective way.

41 **Specify at least four questionnaire which supports to select the prototyping approach. (Nov/Dec 2009)**

- Prototype serves as a basis for deriving system specification.
- Design quality can be improved.
- System can be maintained easily.
- Development efforts may get reduced.
- System usability can be improved.

42 **What is the purpose of domain analysis. (April/May 2010)**

Domain analysis, or product line analysis, is the process of analysing related software systems in a domain to find their common and variable parts. It is a model of wider business context for the system

43 **what are the types of prototypes**

• Evolutionary prototyping – the initial prototype is prepared and it is then refined through number of stages to final stage.

• Throw-away prototyping – a rough practical implementation of the system is produced. The requirement problems can be identified from

this implementation

44 **list two advantage of employing prototyping in software process?**

- Prototype serves as a basis for deriving system specification.
- Design quality can be improved.
- System can be maintained easily.
- Development efforts may get reduced.
- System usability can be improved.

45 **State the different criteria applied to evaluate an effective modular system. (May/June 2006)**

- A system is considered modular if it consists of discreet components so that each component can be implemented separately, and a change to one component has minimal impact on other components.
- Modularity is a clearly a desirable property in a system. Modularity helps in system debugging. Isolating the system problem to a component is easier if the system is modular.

46 **What is meant by structural analysis?**

The structural analysis is mapping of problem domain to flows and transformations. The system can be modeled by using Entity Relationship diagram, Data flow diagram and Control flow

diagrams.

47 **What is the outcome of feasibility study?**

The outcome of feasibility study is the results obtained from the following questions: x Which system contributes to organizational objectives? x Whether the system can be engineered? Is it within the budget? x Whether the system can be integrated with other

existing system?

48 **What are nonfunctional requirements?**

Nonfunctional requirements are constraints on the services or functions offered by the system such as timing constraints,

constraints on the development process, standards, etc…

49 **What are the advantages of evolutionary prototyping?**

i. Fast delivery of the working system. ii. User is involved while developing the system. iii. More useful system can be delivered. iv. Specification, design and implementation work in co-ordinate

manner.

50 **What are the various Rapid prototyping techniques?**

i. Dynamic high level language development. ii. Database programming. iii. Component and application assembly.

PART –B

S.NO	QUESTIONS
1	**Discuss any four process models with suitable application. NOV/DEC-10 , APR/MAY-11, NOV/DEC-12, MAY/JUN-13**

A software process model is a standardised format for

- planning
- organising, and
- running a development project

Hundreds of different models exist and are used, but many are minor variations on a small number of basic models.

1.1. Planning with Models

(a) SE projects usually live with a fixed financial budget. (An exception is maintenance?) Additionally, time-to-market places a strong time constraint. There will be other project constraints such as staff.

Project planning is the art of scheduling/constraint solving the project parameters, along various dimensions: time, money, staff … in order to optimize:

- project risk [low]
- profit [high]
- customer satisfaction [high]
- Worker satisfaction [high]
- long/short-term company goals

Project parameters describe the whole project, but we must at least describe:

- resources needed (people, money, equipment, etc)
- dependency & timing of work (flow graph, work packages)
- rate of delivery (reports, code, etc)

In addition to project members, the following may need access to parts of the project plan:

- Management
- Customers

• *Subcontractors (outsourcing)*

• Suppliers (e.g. licenses, strategic partners)

• Investors (long term investment)

• Banks (short term cash)

1.2. Project Visibility

Unlike other engineers (e.g. civil, electronic, chemical … etc.) software engineers do not produce anything physical.

This means that SE projects must produce additional deliverables (artifacts) which are visible, such as:

• Design documents/ prototypes

• Reports

• Project/status meetings

• Client surveys (e.g. satisfaction level)

A (software/system) process model is a description of the sequence of activities carried out in an SE project, and the relative order of these activities.

It provides a fixed generic framework that can be tailored to a specific project. Project specific parameters will include:

• Size, (person-years)

• *Budget,*

Duration.

project plan = process model + project parameters

There are hundreds of different process models to choose from, e.g:

• waterfall,

• code-and-fix

• spiral

• rapid prototyping

• unified process (UP)

• agile methods, extreme programming (XP)

• COTS …

But most are minor variations on a small number of basic models.

By changing the process model, we can improve and/or tradeoff:

• Development speed (time to market)

• *Product quality*

Project visibility

• Administrative overhead

• Risk exposure

• Customer relations, etc.

Normally, a process model covers the entire lifetime of a product.

From birth of a commercial idea to final de-installation of last release i.e.

The three main phases:

• design,

• build,

• maintain.

We can sometimes combine process models

e.g. 1. waterfall inside evolutionary – onboard shuttle software

2. Evolutionary inside waterfall – e.g. GUI prototyping

We can also evolve the process model together with the product to account for product maturity, e.g. rapid prototyping → waterfall

2 **Explain the execution of seven distinct functions accomplished in requirement engineering process / Explain briefly the requirement engineering process with neat sketch and describe each process with an example. APRIL/MAY-15 NOV/DEC-15, NOV/DEC 2017, APRIL/MAY 2017**

Introduction to requirement engineering

iv. The process of collecting the software requirement from the client then understand, evaluate and document it is called as requirement engineering.

v. Requirement engineering constructs a bridge for design and construction.

Requirement engineering consists of seven different tasks as follow:

1. Inception

- Inception is a task where the requirement engineering asks a set of questions to establish a software process.
- In this task, it understands the problem and evaluates with the proper solution.
- It collaborates with the relationship between the customer and the developer.
- *The developer and customer decide the overall scope and the nature of the question.*

2. Elicitation

Elicitation means to find the requirements from anybody.
The requirements are difficult because the **following problems occur in elicitation**.

Problem of scope: The customer give the unnecessary technical detail rather than clarity of the overall system objective.

Problem of understanding: Poor understanding between the customer and the developer regarding various aspect of the project like capability, limitation of the computing environment.

Problem of volatility: In this problem, the requirements change from time to time and it is difficult while developing the project.

3. Elaboration

- In this task, the information taken from user during inception and elaboration and are expanded and refined in elaboration.
- Its main task is developing pure model of software using functions, feature and constraints of a software.

4. Negotiation

- In negotiation task, a software engineer decides the how will the project be achieved with limited business resources.
- To create rough guesses of development and access the impact of the requirement on the project cost and delivery time.

5. Specification

- In this task, the requirement engineer constructs a final work product.
- The work product is in the form of software requirement specification.
- In this task, formalize the requirement of the proposed software such as informative, functional and behavioral.
- The requirement are formalize in both graphical and textual formats.

6. Validation

- The work product is built as an output of the requirement engineering and that is accessed for the quality through a validation step.
- The formal technical reviews from the software engineer, customer and other stakeholders helps for the primary requirements validation mechanism.

7. Requirement management

- It is a set of activities that help the project team to identify, control and track the requirements and changes can be made to the requirements at any time of the ongoing project.
- *These tasks start with the identification and assign a unique identifier to each of the requirement.*

- After finalizing the requirement traceability table is developed.
- *The examples of traceability table are the features, sources, dependencies, subsystems and interface of the requirement*

3 **What is data dictionary? Explain. How to select the appropriate prototyping approach?APR/MAY-11, APR/MAY-12, NOV/DEC2015**

Is a reference work of data about data (metadata), one that is compiled by the systems analyst to guide them through the analysis and design.
Is the information you see in the data dictionary.
• It is where the systems analyst goes to define or look up information about entities, attributes and relationships on the ERD (Entity Relationship Design).
Importance of a Data Dictionary
Avoid duplication
• Allows better communication between organizations who shares the same database.
• Makes maintenance straightforward
• It is valuable for their capacity to cross-referencing data items.
Uses of Data Dictionary
Validates the date flow diagram for completeness and accuracy
• Provides starting point for developing screen and reports.
• Determine the contents of data stored files
• Develop the logic for data flow diagram processes.
The Data Repository

4 **How does the analysis modeling help to capture unambiguous & consistent requirements? Discuss several methods for requirements validation? NOV/DEC-11**

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 modelers working closely with business stakeholders, as well as potential users of the information system.

There are three different types of data models produced while progressing from requirements to the actual database to be used for the information system.[2] 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 a physical 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.[3]

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.

- Strategic data modeling: This is part of the creation of an information systems strategy, which defines an overall vision and architecture for information systems. Information technology 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.

5 Explain prototyping in the software process.APRIL/MAY-15 MAY/JUNE 2016

The **prototyping model** is applied when detailed information related to input and output requirements of the system is not available. In this model, it is assumed that all the requirements may not be known at the start of the development of the system. It is usually used when a system does not exist or in case of a large and complex system where there is no manual process to determine the requirements. This model allows the users to interact and experiment with a working model of the system known as **prototype.** The prototype gives the user an actual feel of the system.

At any stage, if the user is not satisfied with the prototype, it can be discarded and an entirely new system can be developed. Generally, prototype can be prepared by the approaches listed below.

• By creating main user interfaces without any substantial coding so that users can get a feel of how the actual system will appear.
• By abbreviating a version of the system that will perform limited subsets of functions.
• By using system components to illustrate the functions that will be included in the system to be developed .

Using the prototype, the client can get an actual feel of the system. So, this case of model is beneficial in the case when requirements cannot be freezed initially.

This prototype is developed based on the currently known requirements. Development of the prototype obviously undergoes design, coding, and testing, but each of these phases is not done very formally or thoroughly.

By using this prototype, the client can get an actual feel of the system, because the interactions with the prototype can enable the client to better understand the requirements of the desired system.

6 Explain the functional & behavioral model for software requirements process? <u>NOV/DEC-12, MAY/JUN- 13,NOV/DEC 2013</u>

functional requirement specifies something that the application or system should do. Often, this is defined as a behavior of the system that takes input and provides output. For example, a traveler fills out a form in an airline's mobile application with his/her name and passport details (input), submits the form, and the application generates a boarding pass with the traveler's details (output).

Non-functional requirements, sometimes also called quality requirements, describe how the system should be, as opposed to what it should do. Non-functional requirements of a system include performance (e.g., response time), maintainability and scalability, among many others. In the airline application example, the requirement that the application must display the boarding pass after a maximum of five seconds from the time the traveler presses the 'submit' button would be a non-functional requirement.

7 Explain metrics for specifying non-functional requirements? IEEE standarad software requirement document? <u>MAY/JUN- 13</u>

According to IEEE standard 729, a requirement is defined as follows:

- A condition or capability needed by a user to solve a problem or achieve an objective
- A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification or other formally imposed documents
- A documented representation of a condition or capability as in 1 and 2.

A software requirement can be of 3 types:

- Functional requirements
- Non-functional requirements
- Domain requirements

Functional Requirements: These are the requirements that the end user specifically demands as basic facilities that the system should offer. All these functionalities need to be necessarily incorporated into the system as a part of the contract. These are represented or stated in the form of input to be given to the system, the operation performed and the output expected. They are basically the requirements stated by the user which one can see directly in the final product, unlike the non-functional requirements.

For example, in a hospital management system, a doctor should be able to retrieve the information of his patients. Each high-level functional requirement may involve several interactions or dialogues between the system and the outside world. In order to accurately describe the functional requirements, all scenarios must be enumerated.

There are many ways of expressing functional requirements e.g., natural language, a structured or formatted language with no rigorous syntax and formal specification language with proper syntax.

Non-functional requirements: These are basically the quality constraints that the system must satisfy according to the project contract. The priority or extent to which these factors are implemented varies from one project to other. They are also called non-behavioral requirements. They basically deal with issues like:

- Portability
- Security
- Maintainability
- Reliability
- Scalability
- Performance
- Reusability
- Flexibility

NFR's are classified into following types:

- Interface constraints
- Performance constraints: response time, security, storage space, etc.
- Operating constraints
- Life cycle constraints: maintainability, portability, etc.
- Economic constraints

The process of specifying non-functional requirements requires the knowledge of the functionality of the system, as well as the knowledge of the context within which the system will operate.

Domain requirements: Domain requirements are the requirements which are characteristic of a particular category or domain of projects. The basic functions that a system of a specific domain must necessarily exhibit come under this category. For instance, in academic software that maintains records of a school or college, the functionality of being able to access the list of faculty and list of students of each grade is a domain requirement. These requirements are therefore identified from that domain model and are not user specific.

8 What is requirements elicitation? Explain various activities performed in it with watch system that facilitates to set time and alarm as an example? <u>NOV/DEC 2016, APRIL/MAY 2017, APRIL/MAY 2018</u>

Requirements elicitation is perhaps the most difficult, most error-prone and most communication intensive software development. It can be successful only through an effective customer-developer partnership. It is needed to know what the users really need.

There are a number of requirements elicitation methods. Few of them are listed below –

1. Interviews
2. Brainstorming Sessions
3. Facilitated Application Specification Technique (FAST)
4. Quality Function Deployment (QFD)
5. Use Case Approach

The success of an elicitation technique used depends on the maturity of the analyst, developers, users and the customer involved.

1. Interviews:

Objective of conducting an interview is to understand the customer's expectations from the software.

It is impossible to interview every stakeholder hence representatives from groups are selected based on their expertise and credibility.

Interviews maybe be open ended or structured.

1. In open ended interviews there is no pre-set agenda. Context free questions may be asked to understand the problem.
2. In structured interview, agenda of fairly open questions is prepared. Sometimes a proper questionnaire is designed for the interview.

2. Brainstorming Sessions:

- It is a group technique
- It is intended to generate lots of new ideas hence providing a platform to share views
- A highly trained facilitator is required to handle group bias and group conflicts.
- Every idea is documented so that everyone can see it.
- Finally a document is prepared which consists of the list of requirements and their priority if possible.

3. Facilitated Application Specification Technique:

It's objective is to bridge the expectation gap – difference between what the developers think they are supposed to build and what customers think they are going to get.
A team oriented approach is developed for requirements gathering.
Each attendee is asked to make a list of objects that are-

1. Part of the environment that surrounds the system
2. Produced by the system
3. Used by the system

Each participant prepares his/her list, different lists are then combined, redundant entries are eliminated, team is divided into smaller sub-teams to develop mini-specifications and finally a draft of specifications is written down using all the inputs from the meeting.

4. Quality Function Deployment:

In this technique customer satisfaction is of prime concern, hence it emphasizes on the requirements which are valuable to the customer.
3 types of requirements are identified –

- **Normal requirements –** In this the objective and goals of the proposed software are discussed with the customer. Example – normal requirements for a result management system may be entry of marks, calculation of results etc
- **Expected requirements –** These requirements are so obvious that the customer need not explicitly state them. Example – protection from unauthorised access.
- **Exciting requirements –** It includes features that are beyond customer's expectations and prove to be very satisfying when present. Example – when an unauthorised access is detected, it should backup and shutdown all processes.

The major steps involved in this procedure are –

1. Identify all the stakeholders, eg. Users, developers, customers etc
2. List out all requirements from customer.
3. A value indicating degree of importance is assigned to each requirement.
4. In the end the final list of requirements is categorised as –
    - It is possible to achieve
    - It should be deferred and the reason for it
    - It is impossible to achieve and should be dropped off

5. Use Case Approach:

This technique combines text and pictures to provide a better understanding of the requirements.
The use cases describe the 'what', of a system and not 'how'. Hence they only give a functional view of the system.
The components of the use case deign includes three major things – Actor, Use cases, use case

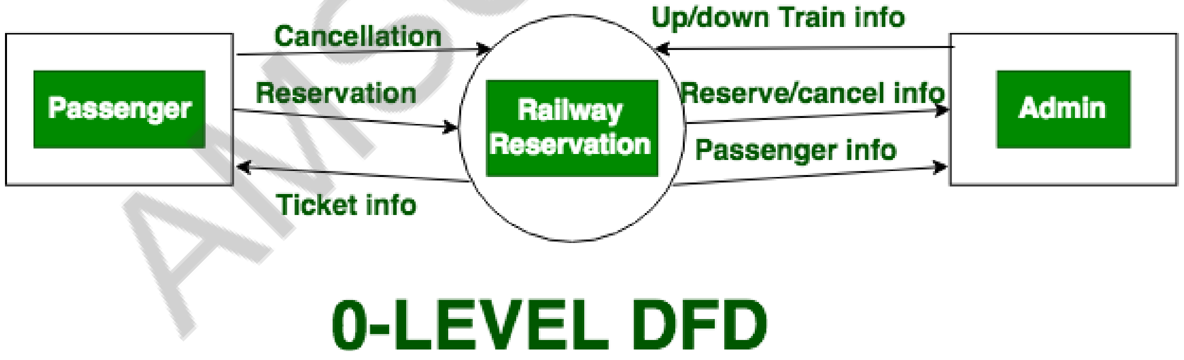
diagram.

1. **Actor –** It is the external agent that lies outside the system but interacts with it in some way. An actor maybe a person, machine etc. It is represented as a stick figure. Actors can be primary actors or secondary actors.
   - Primary actors – It requires assistance from the system to achieve a goal.
   - Secondary actor – It is an actor from which the system needs assistance.
2. **Use cases –** They describe the sequence of interactions between actors and the system. They capture who(actors) do what(interaction) with the system. A complete set of use cases specifies all possible ways to use the system.
3. **Use case diagram –** A use case diiagram graphically represents what happens when an actor interacts with a system. It captures the functional aspect of the system.
   - A stick figure is used to represent an actor.
   - An oval is used to represent a use case.
   - A line is used to represent a relationship between an actor and a use case

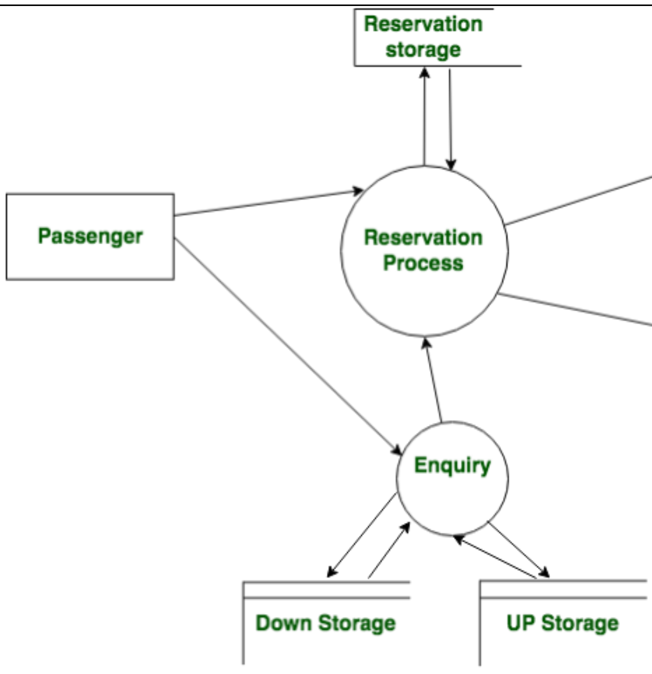
9 What is the purpose of data flow diagrams? What are the notations used for the same. Explain by constructing a context flow diagram level -0 DFD and level-1 DFD for a library management system? <u>NOV/DEC 2016</u>

In Software engineering DFD(data flow diagram) can be drawn to represent the system of different levels of abstraction. Higher level DFDs are partitioned into low levels-hacking more information and functional elements. Levels in DFD are numbered 0, 1, 2 or beyond. Here, we will see mainly 3 levels in data flow diagram, which are: 0-level DFD, 1-level DFD, and 2-level DFD.

0-level DFD:
It is also known as context diagram.It's designed to be an abstraction view, showing the system as a single process with its relationship to external entities. It represent the entire system as single bubble with input and output data indicated by incoming/outgoing arrows.



1-level DFD:
In 1-level DFD, context diagram is decomposed into multiple bubbles/processes.in this level we highlight the main functions of the system and breakdown the high level process of 0-level DFD into subprocesses.

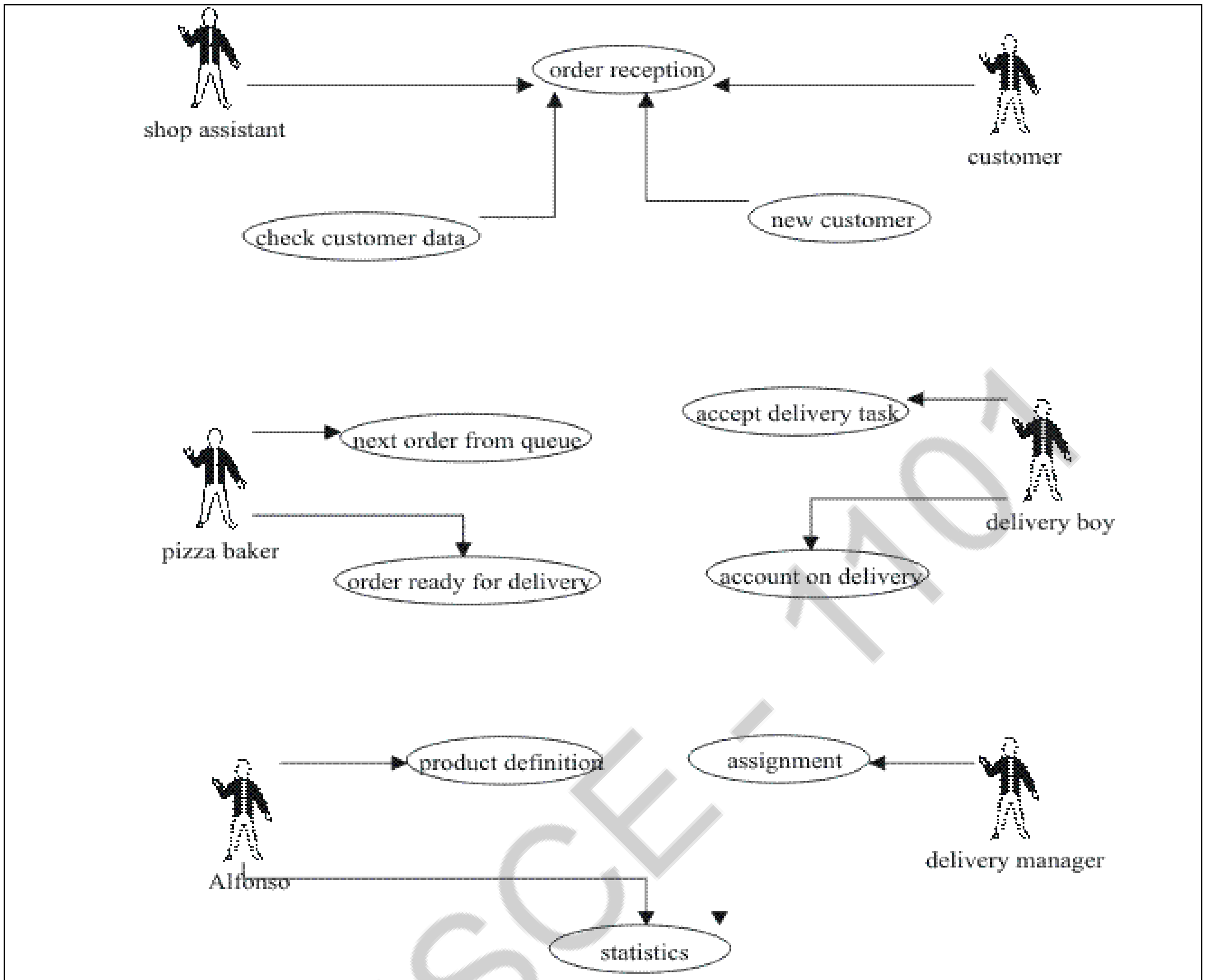


1-LEVEL DFD

2-level DFD:

2-level DFD goes one step deeper into parts of 1-level DFD.It can be used to plan or record the specific/necessary detail about the system's functioning.

10	**Consider the process of ordering a pizza over the phone. Draw the use case diagram and also sketch the activity diagram representing each step of the process, from the moment you pick up the phone to the point where you start eating the pizza. Include activities that others need to perform. Add exception handling to the activity diagram you developed. Consider at least two exceptions.(Ex : Delivery person wrote down wrong address, deliver person brings wrong pizza). NOV/DEC 2017**


shop assistant
order reception
customer
check customer data
new customer
accept delivery task
next order from queue
pizza baker
delivery boy
order ready for delivery
account on delivery
product definition
assignment
delivery manager
Alfonso
statistics

11 Explain the feasibility studies. What are the outcomes? Does it have implicit or explicit effects on software requirement collection. APRIL/MAY 2017

A feasibility study is carried out to select the best system that meets performance requirements. The main aim of the feasibility study activity is to determine whether it would be financially and technically feasible to develop the product. The feasibility study activity involves the analysis of the problem and collection of all relevant information relating to the product such as the different data items which would be input to the system, the processing required to be carried out on these data, the output data required to be produced by the system as well as various constraints on the behaviour of the system.

Technical Feasibility

This is concerned with specifying equipment and software that will successfully satisfy the user requirement. The technical needs of the system may vary considerably, but might include :

• The facility to produce outputs in a given time.
• Response time under certain conditions.
• Ability to process a certain volume of transaction at a particular speed.
• Facility to communicate data to distant locations.

In examining technical feasibility, configuration of the system is given more importance than the actual make of hardware. The configuration should give the complete picture about the system's requirements:

How many workstations are required, how these units are interconnected so that they could operate and communicate smoothly.

What speeds of input and output should be achieved at particular quality of printing.

Economic Feasibility

Economic analysis is the most frequently used technique for evaluating the effectiveness of a proposed system. More commonly known as Cost / Benefit analysis, the procedure is to determine the benefits and savings that are expected from a proposed system and compare them with costs. If benefits outweigh costs, a decision is taken to design and implement the system. Otherwise, further justification or alternative in the proposed system will have to be made if it is to have a chance of being approved. This is an outgoing effort that improves in accuracy at each phase of the system life cycle.

Operational Feasibility

This is mainly related to human organizational and political aspects. The points to be considered are:

• What changes will be brought with the system?
• What organizational structure are disturbed?

• What new skills will be required? Do the existing staff members have these skills? If not, can they be trained in due course of time?
This feasibility study is carried out by a small group of people who are familiar with information system technique and are skilled in system analysis and design process.
Proposed projects are beneficial only if they can be turned into information system that will meet the operating requirements of the organization. This test of feasibility asks if the system will work when it is developed and installed.

12 **What is SRS?Explain in detail about various component of an SRS.**
The output of the requirements phase of the software development process is **Software Requirements Specification (SRS)** (also known as **requirements document).** This document lays a foundation for software engineering activities and is created when entire requirements are elicited and analyzed. SRS is a formal document, which acts as a representation of software that enables the users to review whether it (SRS) is according to their requirements. In addition, it includes user requirements for a system as well as detailed specifications of the system requirements.

IEEE defines software requirements specification as, 'a document that clearly and precisely describes each of the essential requirements (functions, performance, design constraints and quality attributes) of the software and the external interfaces. Each requirement is defined in such a way that its achievement can be objectively verified by a prescribed method, for example, inspection, demonstration, analysis or test.' Note that requirements specification can be in the form of a written document, a mathematical model, a collection of graphical models, a prototype, and so on.

Essentially, what passes from requirements analysis activity to the specification activity is the knowledge acquired about the system. The need for maintaining a requirements document is that the modeling activity essentially focuses on the problem structure and not its structural behavior. While in SRS, performance constraints, design constraints, and standard compliance recovery are clearly specified. This information helps in developing a proper design of the system. Various other purposes served by SRS are listed below.

Feedback: Provides a feedback, which ensures to the user that the organization (which develops the software) understands the issues or problems to be solved and the software behavior necessary to address those problems.
Decompose problem into components: Organizes the information and divides the problem into its component parts in an orderly manner.
Validation: Uses validation strategies applied to the requirements to acknowledge that requirements are stated properly.
Input to design: Contains sufficient detail in the functional system requirements to devise a design solution.
Basis for agreement between the user and **the organization:** Provides a complete description of the functions to be performed by the system. In addition, it helps the users to determine whether the specified requirements are accomplished.
Reduce the development effort: Enables developers to consider user requirements before the designing of the system commences. As a result, 'rework' and inconsistencies in the later stages can be reduced.
Estimating costs and schedules: Determines the requirements of the system and thus enables the developer to have a 'rough' estimate of the total cost and schedule of the project.

SRS is used by various individuals in the organization. System customers need SRS to specify and verify whether requirements meet the desired needs. In addition, SRS enables the managers to plan for the system development processes. System engineers need a requirements document to understand what system is to be developed. These engineers also require this document to develop validation tests for the required system. Lastly, requirements document is needed by system maintenance engineers to use the requirement and the relationship between its parts.

Characteristics of SRS

Software requirements specification should be accurate, complete, efficient, and of high quality, so that it does not affect the entire project plan. An SRS is said to be of high quality when the developer and user easily understand the prepared document. Other characteristics of SRS are discussed below.

Correct: SRS is correct when all user requirements are stated in the requirements document. The stated requirements should be according to the desired system. This implies that each requirement is examined to ensure that it (SRS) represents user requirements. Note that there is no specified tool or procedure to assure the correctness of SRS. Correctness ensures that all specified requirements are performed correctly.

Unambiguous: SRS is unambiguous when every stated requirement has only one interpretation. This implies that each requirement is uniquely interpreted. In case there is a term used with multiple meanings, the requirements document should specify the meanings in the SRS so that it is clear and easy to understand.

Complete: SRS is complete when the requirements clearly define what the software is required to do. This includes all the requirements related to performance, design and functionality.

Ranked for importance/stability: All requirements are not equally important, hence each requirement is identified to make differences among other requirements. For this, it is essential to clearly identify each requirement. Stability implies the probability of changes in the requirement in future.

Modifiable: The requirements of the user can change, hence requirements document should be created in such a manner that those changes can be modified easily, consistently maintaining the structure and style of the SRS.

Traceable: SRS is traceable when the source of each requirement is clear and facilitates the reference of each requirement in future. For this, forward tracing and backward tracing are used. Forward tracing implies that each requirement should be traceable to design and code elements. Backward tracing implies defining each requirement explicitly referencing its source.

Verifiable: SRS is verifiable when the specified requirements can be verified with a cost-effective process to check whether the final software meets those requirements. The requirements are verified with the help of reviews. Note that unambiguity is essential for verifiability.

Consistent: SRS is consistent when the subsets of individual requirements defined do not conflict with each other. For example, there can be a case when different requirements can use different terms to refer to the same object. There can be logical or temporal conflicts between the specified requirements and some requirements whose logical or temporal characteristics are not satisfied. For instance, a requirement states that an event 'a' is to occur before another event 'b'. But then another set of requirements states (directly or indirectly by transitivity) that event 'b' should occur before event 'a'.

Structure of SRS

The requirements document is devised in a manner that is easier to write, review, and maintain. It is organized into independent sections and each section is organized into modules or units. Note that the level of detail to be included in the SRS depends on the type of the system to be developed and the process model chosen for its development. For example, if a system is to be developed by an external contractor, then critical system specifications need to be precise and detailed. Similarly, when flexibility is required in the requirements and where an in-house development takes place, requirements documents can be less detailed.

Since the requirements document serves as a foundation for subsequent software development phases, it is important to develop the document in the prescribed manner. For this, certain guidelines are followed while preparing SRS. These guidelines are listed below.

Functionality: It should be separate from implementation.
Analysis model: It should be developed according to the desired behavior of a system. This should include data and functional response of a system to various inputs given to it.
Cognitive model: It should be developed independently of design or implementation model. This model expresses a system as perceived by the users.
The content and structure of the **specification:** It should be flexible enough to accommodate changes.
Specification: It should be robust. That is, it should be tolerant towards incompleteness and complexity.

The information to be included in SRS depends on a number of factors, for example, the type of software being developed and the approach used in its development. If software is developed using the iterative development process, the requirements document will be less detailed as compared to that of the software developed for critical systems. This is because specifications need to be very detailed and accurate in these systems. A number of standards have been suggested to develop a requirements document. However, the most widely used standard is by IEEE, which acts as a general framework. This general framework can be customized and adapted to meet the needs of a particular organization.

Each SRS fits a certain pattern; thus, it is essential to standardize the structure of the requirements document to make it easier to understand. For this IEEE standard is used for SRS to organize requirements for different projects, which provides different ways of structuring SRS. Note that in all requirements documents, the first two sections are the same.

This document comprises the following sections.

Introduction: This provides an overview of the entire information described in SRS. This involves purpose and the scope of SRS, which states the functions to be performed by the system. In addition, it describes definitions, abbreviations, and the acronyms used. The references used in SRS provide a list of documents that is referenced in the document.
Overall description: It determines the factors which affect the requirements of the system. It provides a brief description of the requirements to be defined in the next section called 'specific requirement'. It comprises the following sub-sections.
Product perspective: It determines whether the product is an independent product or an integral part of the larger product. It determines the interface with hardware, software, system, and communication. It also defines memory constraints and operations utilized by the user.
Product functions: It provides a summary of the functions to be performed by the software. The functions are organized in a list so that they are easily understandable by the user:
User characteristics: It determines general characteristics of the users.
Constraints: It provides the genera1 description of the constraints such as regulatory policies, audit functions, reliability requirements, and so on.
Assumption and dependency: It provides a list of assumptions and factors that affect the requirements as stated in this document.
Apportioning of requirements: It determines the requirements that can be delayed until release of future versions of the system.
Specific requirements: These determine all requirements in detail so that the designers can design the system in accordance with them. The requirements include description of every input and output of the system and functions performed in response to the input provided. It comprises the

following subsections.

External interface: It determines the interface of the software with other systems, which can include interface with operating system and so on. External interface also specifies the interaction of the software with users, hardware, or other software. The characteristics of each user interface of the software product are specified in SRS. For the hardware interface, SRS specifies the logical characteristics of each interface among the software and hardware components. If the software is to be executed on the existing hardware, then characteristics such as memory restrictions are also specified.

Functions: It determines the functional capabilities of the system. For each functional requirement, the accepting and processing of inputs in order to generate outputs are specified. This includes validity checks on inputs, exact sequence of operations, relationship of inputs to output, and so on.

Performance requirements: It determines the performance constraints of the software system. Performance requirement is of two types: static requirements and dynamic requirements. **Static requirements** (also known as **capacity requirements)** do not impose constraints on the execution characteristics of the system. These include requirements like number of terminals and users to be supported. **Dynamic requirements** determine the constraints on the execution of the behavior of the system, which includes response time (the time between the start and ending of an operation under specified conditions) and throughput (total amount of work done in a given time).

Logical database of requirements: It determines logical requirements to be stored in the database. This includes type of information used, frequency of usage, data entities and relationships among them, and so on.

Design constraint: It determines all design constraints that are imposed by standards, hardware limitations, and so on. Standard compliance determines requirements for the system, which are in compliance with the specified standards. These standards can include accounting procedures and report format. Hardware limitations implies when the software can operate on existing hardware or some pre-determined hardware. This can impose restrictions while developing the software design. Hardware limitations include hardware configuration of the machine and operating system to be used.

Software system attributes: It provide attributes such as reliability, availability, maintainability and portability. It is essential to describe all these attributes to verify that they are achieved in the final system.

Organizing Specific Requirements: It determines the requirements so that they can be properly organized for optimal understanding. The requirements can be organized on the basis of mode of operation, user classes, objects, feature, response, and functional hierarchy.

Change management process: It determines the change management process in order to identify, evaluate, and update SRS to reflect changes in the project scope and requirements.

Document approvals: These provide information about the approvers of the SRS document with the details such as approver's name, signature, date, and so on.

Supporting information: It provides information such as table of contents, index, and so on. This is necessary especially when SRS is prepared for large and complex projects.

13 **What is requirement engineering? State its process and explain requirement elicitation problem. (April/May 2008)**

Requirement Engineering

The process to gather the software requirements from client, analyze and document them is known as requirement engineering.

The goal of requirement engineering is to develop and maintain sophisticated and descriptive 'System Requirements Specification' document.

Requirement Engineering Process

It is a four step process, which includes –

- Feasibility Study
- Requirement Gathering
- Software Requirement Specification
- Software Requirement Validation

Let us see the process briefly -

Feasibility study

When the client approaches the organization for getting the desired product developed, it comes up with rough idea about what all functions the software must perform and which all features are expected from the software.

Referencing to this information, the analysts does a detailed study about whether the desired system and its functionality are feasible to develop.

This feasibility study is focused towards goal of the organization. This study analyzes whether the software product can be practically materialized in terms of implementation, contribution of project to organization, cost constraints and as per values and objectives of the organization. It explores technical aspects of the project and product such as usability, maintainability, productivity and integration ability.

The output of this phase should be a feasibility study report that should contain adequate comments and recommendations for management about whether or not the project should be undertaken.

Requirement Gathering

If the feasibility report is positive towards undertaking the project, next phase starts with gathering requirements from the user. Analysts and engineers communicate with the client and end-users to know their ideas on what the software should provide and which features they want the software to include.

Software Requirement Specification

SRS is a document created by system analyst after the requirements are collected from various stakeholders.

SRS defines how the intended software will interact with hardware, external interfaces, speed of operation, response time of system, portability of software across various platforms, maintainability, speed of recovery after crashing, Security, Quality, Limitations etc.

The requirements received from client are written in natural language. It is the responsibility of system analyst to document the requirements in technical language so that they can be comprehended and useful by the software development team.

SRS should come up with following features:

- User Requirements are expressed in natural language.
- Technical requirements are expressed in structured language, which is used inside the organization.
- Design description should be written in Pseudo code.
- Format of Forms and GUI screen prints.
- Conditional and mathematical notations for DFDs etc.

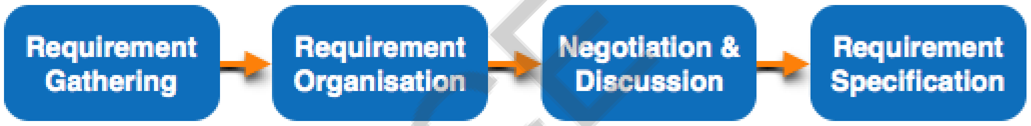
Software Requirement Validation

After requirement specifications are developed, the requirements mentioned in this document are validated. User might ask for illegal, impractical solution or experts may interpret the requirements incorrectly. This results in huge increase in cost if not nipped in the bud. Requirements can be checked against following conditions -

- If they can be practically implemented
- If they are valid and as per functionality and domain of software
- If there are any ambiguities
- If they are complete
- If they can be demonstrated

Requirement Elicitation Process

Requirement elicitation process can be depicted using the folloiwng diagram:



- **Requirements gathering -** The developers discuss with the client and end users and know their expectations from the software.
- **Organizing Requirements -** The developers prioritize and arrange the requirements in order of importance, urgency and convenience.
- **Negotiation & discussion -** If requirements are ambiguous or there are some conflicts in requirements of various stakeholders, if they are, it is then negotiated and discussed with stakeholders. Requirements may then be prioritized and reasonably compromised.

  The requirements come from various stakeholders. To remove the ambiguity and conflicts, they are discussed for clarity and correctness. Unrealistic requirements are compromised reasonably.

- **Documentation -** All formal & informal, functional and non-functional requirements are documented and made available for next phase processing.

Requirement Elicitation Techniques

Requirements Elicitation is the process to find out the requirements for an intended software system by communicating with client, end users, system users and others who have a stake in the software system development.

There are various ways to discover requirements

Interviews

Interviews are strong medium to collect requirements. Organization may conduct several types of interviews such as:

- Structured (closed) interviews, where every single information to gather is decided in advance, they follow pattern and matter of discussion firmly.
- Non-structured (open) interviews, where information to gather is not decided in advance, more flexible and less biased.
- Oral interviews
- Written interviews
- One-to-one interviews which are held between two persons across the table.
- Group interviews which are held between groups of participants. They help to uncover any missing requirement as numerous people are involved.

14 what is prototyping .explain its types types.(Nov/Dec 2009)

A prototype is a model version of a product. It's used as an early, inexpensive sample of a product that helps to test its features or identify defects so improvements can be made to its final version.

Prototypes provides the opportunity to gather valuable feedback from stakeholders, partners or customers about the product. This information can be used to build a product that meets their requirements.

Following are a few advantages of prototyping

- Collect feedback from users/ stakeholders about the functionality of the product before the public release
- Reveal areas for improvement and help identify faults and usability issues before the public release. Help reduce unnecessary costs.
- Improve team efficiency and collaboration
- Allow the user to interact with a working model of their product
- Help convert an abstract idea into a tangible product in a cost-effective way
- Identify if your product idea is a weak one and cost you heavily before actually moving forward with it

Prototyping Types

Prototyping methods and prototyping techniques can be categorized as low-fidelity prototypes and high-fidelity prototypes.

Based on the resources available to you and the purpose for prototyping, the prototyping method you choose can be either be low-fidelity or high-fidelity.

Low-Fidelity Prototypes

Low-fidelity prototypes represent a simple and incomplete version of the final product. In a low-fidelity prototype, not all visual features and content elements are conveyed.

While it doesn’t take much time or effort to translate a broad concept to a low-fidelity prototype, it can be used to gather user feedback during the early stage.

Low-fidelity prototyping methods

Wireframes

Wireframes are used to represent the basic structure of a website/ web page/ app. It serves as a blueprint, highlighting the layout of key elements on a page and its functionality.

With Creately, you can create clickable wireframes by adding links to the wireframe elements, that will allow your users to navigate from one interface to the other

UNIT – 3PART –A

S.NO	QUESTIONS
1	**What are the primary interaction styles and state their advantages? NOV/DEC-10** 1.Direct manipulation - Easiest to grasp with immediate feedback , Difficult to program 2. Menu selection - User effort and errors minimized, large numbers and combinations of choices a problem 3. Form fill-in - Ease of use, simple data entry, Tedious, takes a lot of screen space 4. Command language - Easy to program and process, Difficult to master for casual users 5. Natural language - Great for casual users, Tedious for expert users.
2	**List the architectural models that can be developed. NOV/DEC-10** Data-centered architectures, Data flow architectures, Call and return architectures Object-oriented architectures, Layered architectures.
3	**What is meant by real time system design? APR/MAY-11** A real-time system is a software system where the correct functioning of the system depends on the results produced by the system and the time at which these results are produced.
4	**List four design principles of a good design? APR/MAY-11APRIL/MAY 2018**

- Process should not suffer from tunnel vision.
- It should be traceable to the analysis model
- It should not reinvent the wheel
- It should exhibit uniformity & integration.

5 **List out design methods. <u>APR/MAY-12</u>**

Architectural design , data design , modular design.

6 **Define data acquisition <u>APR/MAY-12,MAY/JUN-13</u>**

Collect data from sensors for subsequent processing and analysis.

7 **How do you apply modularization criteria for a monolithic software <u>NOV/DEC-12</u>**

Modularity is achieved to various extents by different modularization approaches. Code based modularity allows developers to reuse and repair parts of the application, but development tools are required to perform these maintenance functions .Object based modularity provides the application as a collection of separate executable files which may be independently maintained and replaced without redeploying the entire application.

8 **What is the design quality attributes 'FURPS' meant? <u>NOV/DEC-12,</u> NOV/DEC2015, NOV/DEC2017**

FURPS is an acronym representing a model for classifying software quality attributes (functional and non- functional requirements)

Functionality, Usability, Reliability, Performance and Supportability model.

9 **Define data abstraction? <u>MAY/JUN-13</u>**

Data abstraction is a named collection of data that describes the data object.

Eg:- Door attribute – door type, swing direction, weight

10 **What are the elements of design model?**

i. Data design

ii. Architectural design

iii. Interface design

iv. Component-level design

11 **What is the benefit of modular design?**

Changes made during testing and maintenance becomes manageable and they do not affect other modules.

12 **Name the commonly used architectural styles.**

i. Data centered architecture. ii. Data flow architecture.

iii. Call and return architecture. iv. Object-oriented architecture. v. Layered architecture.

13 **What is a cohesive module?**

A cohesive module performs only "one task" in software
procedure with little interaction with other modules. In other words cohesive module performs only one thing.

14 What are the different types of Cohesion?

i. Coincidentally cohesive –The modules in which the set I\of tasks are related with each other loosely then such modules are called coincidentally cohesive.

ii. Logically cohesive – A module that performs the tasks that are logically related with each other is called logically cohesive.

iii. Temporal cohesion – The module in which the tasks need to be executed in some specific time span is called temporal cohesive.

iv. Procedural cohesion – When processing elements of a module are related with procedural cohesive.

v. Communicational cohesion – When the processing elements of a module share the data then such module is called
communicational cohesive.

15 What is Coupling?What are the various types of coupling <u>APRIL/MAY-15.</u>

Coupling is the measure of interconnection among modules in a program structure. It depends on the interface complexity between modules.

i. Data coupling – The data coupling is possible by parameter passing or data interaction.

ii. Control coupling – The modules share related control data in control coupling.

iii. Common coupling – The common data or a global data is shared among modules.
iv. Content coupling – Content coupling occurs when one module makes use of data or control information
maintained in another module.

16 What are the common activities in design process?

i. System structuring – The system is subdivided into principle subsystems components and communications between these subsystems are identified.

ii. Control modeling – A model of control relationships between different parts of the system is established.

iii. Modular decomposition – The identified subsystems are decomposed into modules

17 What are the benefits of horizontal partitioning?

i. Software that is easy to test.

ii. Software that is easier to maintain.

iii. Propagation of fewer sideeffects. iv. Software that is easier to extend.

18 What is vertical partitioning? What are the advantages?

Vertical partitioning often called factoring suggests that the control and work should be distributed top-down in program structure.

i. These are easy to maintain changes.

ii. They reduce the change impact and error propagation

19 If a module has logical cohesion, what kind of coupling is this module likely to have? APR/MAY 2016

If a module has logical cohesion, then content coupling can be done. In content coupling one module can make use of data or control information maintained in another

20 **Write the best practices for "coding"? APR/MAY 2015, NOV/DEC2015**

Best coding practices are a set of informal rules that the software development community has learned over time which can help improve the quality of software. "The first 90% of the code accounts for the first 90% of the development time. The remaining 10% of the code accounts for the other 90% of the development time." The size of a project or program has a significant effect on error rates, programmer productivity, and the amount of management needed.

21 **What architectural styles are preferred for the following system? Why? NOV/DEC2016**

(a) Networking – Data centered Architecture

(b) Web based systems – Call and return architecture

(c) Banking system - Data centered Architecture.

22 **What is DFD?**

Data Flow Diagram depicts the information flow and the transforms that are applied on the data as it moves from input to output.

23 **Name the commonly used architectural styles**.

i. Data centered architecture. ii. Data flow architecture.
iii. Call and return architecture. iv. Object-oriented architecture. v. Layered architecture.

24 **What is ERD?**

Entity Relationship Diagram is the graphical representation of the object relationship pair. It is mainly used in database applications.

25 **What UI design patters are used for the following? <u>NOV/DEC</u> <u>2016, APRIL/MAY 2017, APRIL/MAY 2018</u>**

(a) Page layout – interface design

(b) Tables - Design

(c) Navigation through menus and web pages – design

(d) Shopping cart – interface design, task analysis

26 **What are the various elements of data design?**

i. Data object – The data objects are identified and relationship among various data objects can be represented using ERD or data dictionaries.

ii. Databases – Using software design model, the data models are translated into data structures and data bases at the application level.

iii. Data warehouses – At the business level useful information is identified from various databases and the data warehouses are created.

27 **List the guidelines for data design.**

i. Apply systematic analysis on data.

ii. Identify data structures and related operations.

iii. Establish data dictionary.

iv. Use information hiding in the design of data structure.

v. Apply a library of useful data structures and operations.

28 **What is a Real time system?**

Real time system is a software system in which the correct functionalities of the system are dependent upon results produced by the system and the time at which these results are produced

29 **How do you describe software interface? <u>April /May 2015</u>**

Software interface - the languages and codes that the applications use to communicate with each other and also with the hardware.

Three types of interface may have to be defined

- Procedural interfaces;
- Data structures that are exchanged;
- Data representations.

The interface describes the behavior of a software component that is obtained by considering only the interactions of that interface and by hiding all other interactions.

30 **Explain the qualitative criteria for measuring independence? NOV/DEC-11**

1.C**ohesion:** Cohesion is a qualitative indication of the degree to which a module focuses on just one thing.

2. **Coupling:** Coupling is the measure of interconnection among modules in a program structure. It depends on the interface complexity between modules

31 **What is the purpose of a petrinet ? APRIL/MAY 2017**

A Petri net, also known as a place/transition (PT) net, is one of several mathematicalmodeling languages for the description of distributed systems. It is a class of discrete event dynamic system. Petri nets offer a graphical notation for stepwise processes that

include choice, iteration, and concurrent execution

32 **What is vertical partitioning?**

Vertical partitioning often called factoring suggests that the control and work should be distributed top-down in program structure.

33 **What are the benefits of horizontal partitioning?**

i. Software that is easy to test. ii. Software that is easier to maintain. iii. Propagation of fewer side effects. iv. Software that is easier to extend.

34 **What are data acquisition systems?**

Systems that collect data from sensors for subsequent processing and analysis are termed as data acquisition systems. Data collection processes and processing processes may have different
periods and deadlines.

35 **What is interface design?**

The interface design describes how the software communicates
within itself, with systems that interoperate with it, and with humans who use it.

36 **What are the elements of design model?**

Data design
ii. Architectural design
iii. Interface design
iv. Component-level design

37 **What is coupling?**

Coupling is the measure of interconnection among modules in a program structure. It depends on the interface complexity between modules.

38 **Define design process**.

Design process is a sequence of steps carried through which the requirements are translated into a system or software model.

39 **What is Transform mapping?**

The transform mapping is a set of design steps applied on the DFD
in order to map the transformed flow characteristics into specific architectural style.

40 **What is component level design?**

The component level design transforms structural elements of the
software architecture into a procedural description of software components.

41 **What are the objectives of Analysis modeling?**

i. To describe what the customer requires. ii. To establish a basis
for the creation of software design. iii. To devise a set of valid
requirements after which the software can be built.

42 **What are the various types of coupling?**

i**Data coupling** – The data coupling is possible by parameter passing or data interaction.

ii. **Control coupling** – The modules share related control data in control coupling.

iii. **Common coupling** – The common data or a global data is shared among modules.

iv. **Content coupling** – Content coupling occurs when one module makes use of data or control information maintained in another module.

43 What does modality in data modeling indicates?

Modality indicates whether or not a particular data object must participate in the relationship.

44 **What does Level0 DFD represent?**

Level 0 DFD is called as „fundamental system model‟ or „context model‟. In the context model the entire software system is represented by a single bubble with input and output indicated by
incoming and outgoing arrows.

45 **What are the elements of design model?**

i. Data design ii. Architectural design iii. Interface design iv. Component-level design

46 **What is data modeling?**

Data modeling is the basic step in the analysis modeling. In data modeling the data objects are examined independently of processing. The data model represents how data are related with
one another.

47 **What is a data object?**

Data object is a collection of attributes that act as an aspect,

characteristic,quality, or descriptor of the object

48 **What are attributes?**

Attributes are the one, which defines the properties of data object.

49 **What is cardinality in data modeling?**

Cardinality in data modeling, cardinality specifies how the number of occurrences of one object is related to the number of
occurrences of another object.

50 **What is ERD?**

Entity Relationship Diagram is the graphical representation of the object relationship pair. It is mainly used in database applications

PART –B

S.NO	QUESTIONS
1	**Explain the core activities involved in User Interface design process with necessary block diagrams<u>MAY/JUNE 2016 ,NOV/DEC2015,</u> <u>NOV/DEC 2017</u>**

User interface is the front-end application view to which user interacts in order to use the software. User can manipulate and control the software as well as hardware by means of user interface. Today, user interface is found at almost every place where digital technology exists, right from computers, mobile phones, cars, music players, airplanes, ships etc.

User interface is part of software and is designed such a way that it is expected to provide the user insight of the software. UI provides fundamental platform for human-computer interaction.

UI can be graphical, text-based, audio-video based, depending upon the underlying hardware and software combination. UI can be hardware or software or a combination of both.

The software becomes more popular if its user interface is:

- Attractive
- Simple to use
- Responsive in short time
- Clear to understand
- Consistent on all interfacing screens

UI is broadly divided into two categories:

- Command Line Interface
- Graphical User Interface

Command Line Interface (CLI)

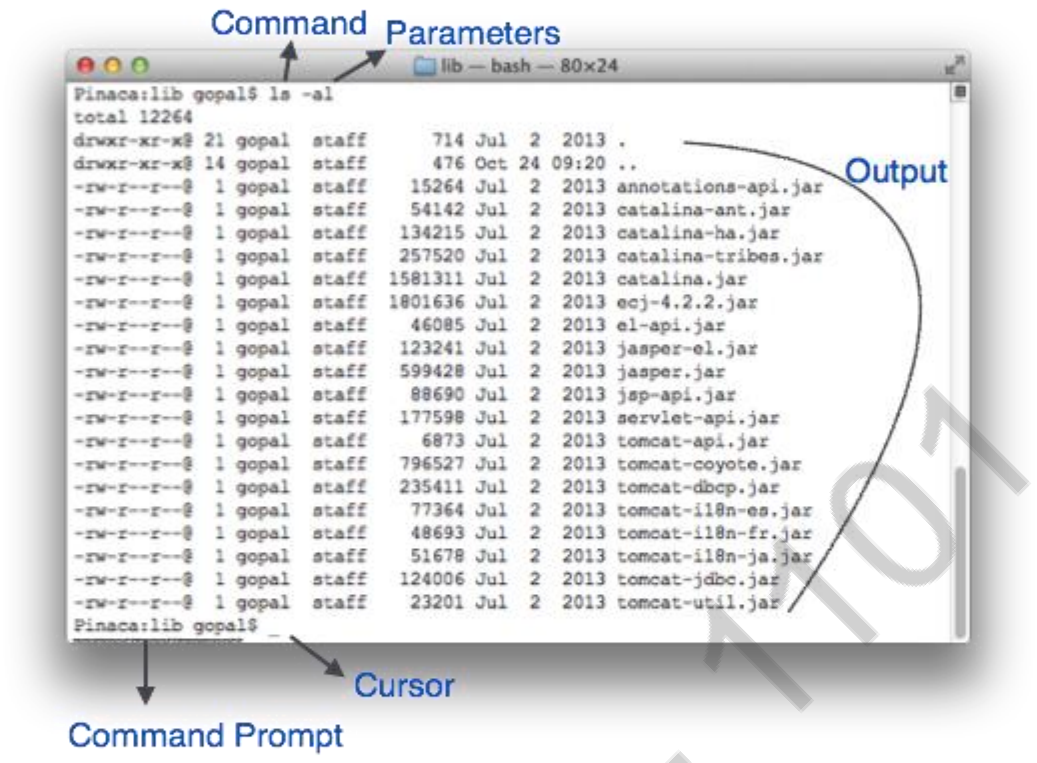
CLI has been a great tool of interaction with computers until the video display monitors came into existence. CLI is first choice of many technical users and programmers. CLI is minimum interface a software can provide to its users.

CLI provides a command prompt, the place where the user types the command and feeds to the system. The user needs to remember the syntax of command and its use. Earlier CLI were not programmed to handle the user errors effectively.

A command is a text-based reference to set of instructions, which are expected to be executed by the system. There are methods like macros, scripts that make it easy for the user to operate.

CLI uses less amount of computer resource as compared to GUI.

CLI Elements



```
Pinaca:lib gopal$ ls -al
total 12264
drwxr-xr-x@ 21 gopal  staff      714 Jul  2  2013 .
drwxr-xr-x@ 14 gopal  staff      476 Oct 24 09:20 ..
-rw-r--r--@  1 gopal  staff    15264 Jul  2  2013 annotations-api.jar
-rw-r--r--@  1 gopal  staff    54142 Jul  2  2013 catalina-ant.jar
-rw-r--r--@  1 gopal  staff   134215 Jul  2  2013 catalina-ha.jar
-rw-r--r--@  1 gopal  staff   257520 Jul  2  2013 catalina-tribes.jar
-rw-r--r--@  1 gopal  staff  1581311 Jul  2  2013 catalina.jar
-rw-r--r--@  1 gopal  staff  1801636 Jul  2  2013 ecj-4.2.2.jar
-rw-r--r--@  1 gopal  staff    46085 Jul  2  2013 el-api.jar
-rw-r--r--@  1 gopal  staff   123241 Jul  2  2013 jasper-el.jar
-rw-r--r--@  1 gopal  staff   599428 Jul  2  2013 jasper.jar
-rw-r--r--@  1 gopal  staff    88690 Jul  2  2013 jsp-api.jar
-rw-r--r--@  1 gopal  staff   177598 Jul  2  2013 servlet-api.jar
-rw-r--r--@  1 gopal  staff     6873 Jul  2  2013 tomcat-api.jar
-rw-r--r--@  1 gopal  staff   796527 Jul  2  2013 tomcat-coyote.jar
-rw-r--r--@  1 gopal  staff   235411 Jul  2  2013 tomcat-dbcp.jar
-rw-r--r--@  1 gopal  staff    77364 Jul  2  2013 tomcat-i18n-es.jar
-rw-r--r--@  1 gopal  staff    48693 Jul  2  2013 tomcat-i18n-fr.jar
-rw-r--r--@  1 gopal  staff    51678 Jul  2  2013 tomcat-i18n-ja.jar
-rw-r--r--@  1 gopal  staff   124006 Jul  2  2013 tomcat-jdbc.jar
-rw-r--r--@  1 gopal  staff    23201 Jul  2  2013 tomcat-util.jar
Pinaca:lib gopal$ _
```

A text-based command line interface can have the following elements:

- **Command Prompt -** It is text-based notifier that is mostly shows the context in which the user is working. It is generated by the software system.
- **Cursor -** It is a small horizontal line or a vertical bar of the height of line, to represent position of character while typing. Cursor is mostly found in blinking state. It moves as the user writes or deletes something.
- **Command -** A command is an executable instruction. It may have one or more parameters. Output on command execution is shown inline on the screen. When output is produced, command prompt is displayed on the next line.

Graphical User Interface

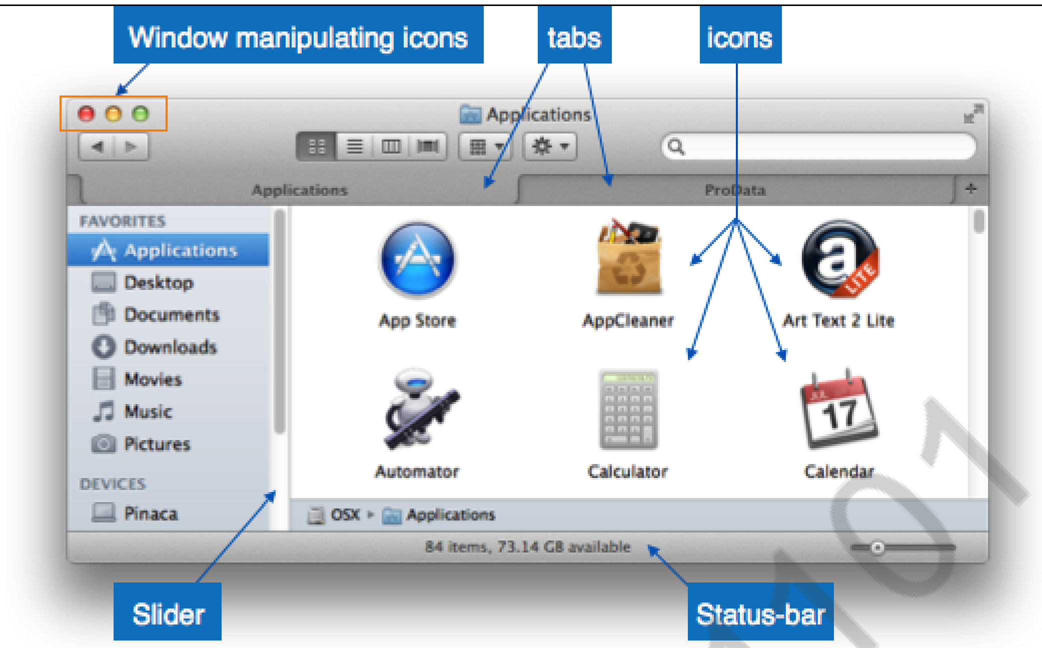
Graphical User Interface provides the user graphical means to interact with the system. GUI can be combination of both hardware and software. Using GUI, user interprets the software.

Typically, GUI is more resource consuming than that of CLI. With advancing technology, the programmers and designers create complex GUI designs that work with more efficiency, accuracy and speed.

GUI Elements

GUI provides a set of components to interact with software or hardware.

Every graphical component provides a way to work with the system. A GUI system has following elements such as:



- **Window** - An area where contents of application are displayed. Contents in a window can be displayed in the form of icons or lists, if the window represents file structure. It is easier for a user to navigate in the file system in an exploring window. Windows can be minimized, resized or maximized to the size of screen. They can be moved anywhere on the screen. A window may contain another window of the same application, called child window.
- **Tabs** - If an application allows executing multiple instances of itself, they appear on the screen as separate windows. **Tabbed Document Interface** has come up to open multiple documents in the same window. This interface also helps in viewing preference panel in application. All modern web-browsers use this feature.
- **Menu** - Menu is an array of standard commands, grouped together and placed at a visible place (usually top) inside the application window. The menu can be programmed to appear or hide on mouse clicks.
- **Icon** - An icon is small picture representing an associated application. When these icons are clicked or double clicked, the application window is opened. Icon displays application and programs installed on a system in the form of small pictures.
- **Cursor** - Interacting devices such as mouse, touch pad, digital pen are represented in GUI as cursors. On screen cursor follows the instructions from hardware in almost real-time. Cursors are also named pointers in GUI systems. They are used to select menus, windows and other application features.

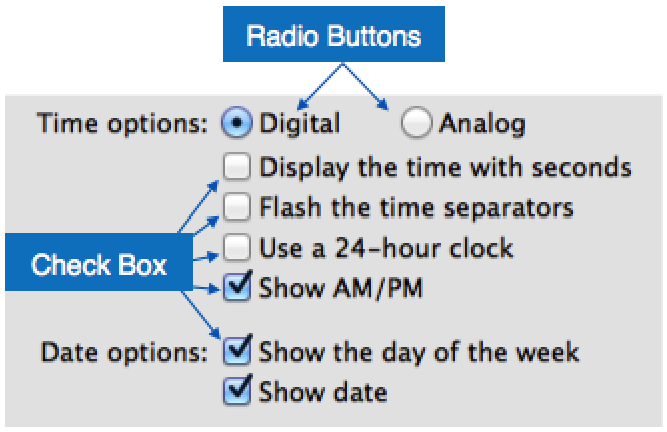
Application specific GUI components

A GUI of an application contains one or more of the listed GUI elements:

- **Application Window** - Most application windows uses the constructs supplied by operating systems but many use their own customer created windows to contain the contents of application.
- **Dialogue Box** - It is a child window that contains message for the user and request for some action to be taken. For Example: Application generate a dialogue to get confirmation

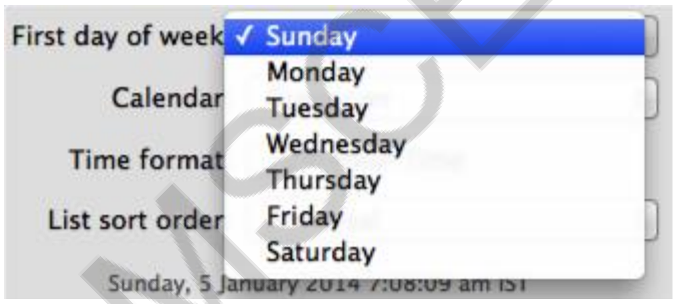
from user to delete a file.

- **ext-Box** - Provides an area for user to type and enter text-based data.
- **Buttons** - They imitate real life buttons and are used to submit inputs to the software.



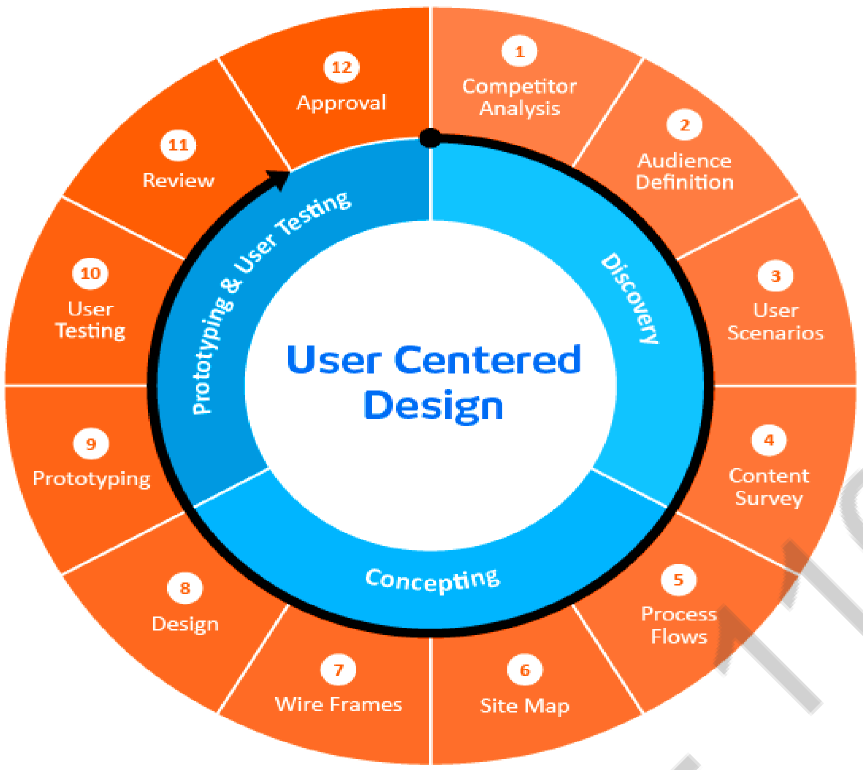
Time options: [✓ Digital] Analog
Display the time with seconds
Flash the time separators
Use a 24-hour clock
[✓ Show AM/PM]
Date options: [✓ Show the day of the week]
[✓ Show date]

- **Radio-button** - Displays available options for selection. Only one can be selected among all offered.
- **Check-box** - Functions similar to list-box. When an option is selected, the box is marked as checked. Multiple options represented by check boxes can be selected.
- **List-box** - Provides list of available items for selection. More than one item can be selected.


First day of week [✓ Sunday]
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Calendar
Time format
List sort order
Sunday, 5 January 2014 7:08:09 am IST

Other impressive GUI components are:

- Sliders
- Combo-box
- Data-grid
- Drop-down list
-



2 **Explain the various modular decomposition and control styles commonly used in any organizational model.MAY/JUNE 2016**

Modular decomposition

Another structural level where sub-systems are decomposed into modules. ● Two modular decomposition models covered • An object model where the system is decomposed into interacting object; • A pipeline or data-flow model where the system is decomposed into functional modules which transform inputs to outputs. ● If possible, decisions about concurrency should be delayed until modules are implemented

Modular decomposition styles

Styles of decomposing sub-systems into modules. ● No rigid distinction between system organisation and modular decomposition.

Sub-systems and modules

A sub-system is a system in its own right whose operation is independent of the services provided by other sub-systems. ● A module is a system component that provides services to other components but would not normally be considered as a separate system

Control styles

Are concerned with the control flow between sub-systems. Distinct from the system decomposition model.

● Centralised control • One sub-system has overall responsibility for control and starts and stops other sub-systems.

● Event-based control • Each sub-system can respond to externally generated events from other sub-systems or the system's environment.

Centralised control

A control sub-system takes responsibility for managing the execution of other sub-systems.

● Call-return model

• Top-down subroutine model where control starts at the top of a subroutine hierarchy and moves downwards. Applicable to sequential systems.

● Manager model

• Applicable to concurrent systems.

- One system component controls the stopping, starting and coordination of other system processes. Can be implemented in sequential systems as a case statement.

Event-driven systems

Driven by externally generated events where the timing of the event is outwith the control of the subsystems which process the event.

● Two principal event-driven models

• Broadcast models. An event is broadcast to all subsystems. Any sub-system which can handle the event may do so;

• Interrupt-driven models. Used in real-time systems where interrupts are detected by an interrupt handler and passed to some other component for processing

. ● Other event driven models include spreadsheets and production systems.

3 **Discuss the process of translating the analysis model in to a software design, List the golden rules of user interface design<u>NOV/DEC2015</u>**

The process of implementation of a software may be define as a process of translation old software to new software with a new developed software who have extra functions and making it operational without any interruption in an organization functioning system. The time period which starts from the acceptance of the tested design to its satisfactory operated it covers all the time period. The software implementation is a very big operation and for the implementation of a software better planning is must require. The planning of implementation of software should be implemented from a short point and after the success it implemented on whole area. For the implementation of a new software a good knowledge is must require and some requirement of a system are hardware, file conversion actions and some personal needs of software.

Activities involved in software implementation

When old software and new software is modified and implemented then it contains three basic actions.

1. **Personal training -** For the implementation of new software, the training of users and operators is necessary part. The training activity plays a major part in operating and maintaining the software by user. Thus we can say that operators and user both require

training.

- **Software operator training -** Most software run smoothly depends on the computer user. The training of computer operator gives the satisfaction that he can do every action and data entry. In the process of training a list of problem can be figured out and solution can be provided to then so that they can solve their problems on their base and build the knowledge about this. If they get unusual problem they can contact the concerned person. With the help of training they become friendly with software and solve can their problem easily.
- **User training -** User training helps the user in operating the system in efficient way. During the training a manual is given to every user so that they can understand the problem and solved it. The content of training is about the use of data that how they can edit, add, query and delete the records. If a user have not sufficient capability of working on system then many kind of errors and problems can occur.

2. **Conversion -** With the help of conversion process a old software can be replaced with new software. The process of conversion is useful in only that case where new software is fully tested and report is positive. It involves many kinds of actions which are:
    - From old to new software system all files and data base converted.
    - Providing the user training of the each staff of the organization which has the right of using new software.
    - Conversion of forms. This may involve discarding old data.
    - Converting administration. In the process of converting administration process the role of each member is divided according the needs and the responsibility is also divided according to their job regarding new software.
3. **Post implementation Review -** After the process of implementation and conversion of software some reviews are taken by the user and the experts. This is the normal process of getting the following points:
    - What is the working of a software system?
    - How it has been accepted by the user?
    - Area of updating

    Performance of a software measured with the help of a post implementation review. It helps in deciding that software gets the specification with how much efficiency.

Types of implementation

We have three types of implementation method which are given below:

1. **Fresh implementation -** Fresh implementation of software may be defined as a process where a manual record are replaced with new software. During the process of fresh implementation some problems come in the form of conversion of files, user training, accurate system of files etc.
2. **Replacement implementation -** When an old software is replaced with a new software implementation that the name of this process is Replacement implementation. This process is very difficult and a proper planning is needed for this, otherwise many problems can arise.

3. **Modified implementation -** When an old software is replaced by new software with some alteration then this process is called modified implementation. We can easily handle this type of implementation because area of modification is not so large in files.

User Interface Golden rules

The following rules are mentioned to be the golden rules for GUI design, described by Shneiderman and Plaisant in their book (Designing the User Interface).

- **Strive for consistency -** Consistent sequences of actions should be required in similar situations. Identical terminology should be used in prompts, menus, and help screens. Consistent commands should be employed throughout.
- **Enable frequent users to use short-cuts -** The user's desire to reduce the number of interactions increases with the frequency of use. Abbreviations, function keys, hidden commands, and macro facilities are very helpful to an expert user.
- **Offer informative feedback -** For every operator action, there should be some system feedback. For frequent and minor actions, the response must be modest, while for infrequent and major actions, the response must be more substantial.
- **Design dialog to yield closure -** Sequences of actions should be organized into groups with a beginning, middle, and end. The informative feedback at the completion of a group of actions gives the operators the satisfaction of accomplishment, a sense of relief, the signal to drop contingency plans and options from their minds, and this indicates that the way ahead is clear to prepare for the next group of actions.
- **Offer simple error handling -** As much as possible, design the system so the user will not make a serious error. If an error is made, the system should be able to detect it and offer simple, comprehensible mechanisms for handling the error.
- **Permit easy reversal of actions -** This feature relieves anxiety, since the user knows that errors can be undone. Easy reversal of actions encourages exploration of unfamiliar options. The units of reversibility may be a single action, a data entry, or a complete group of actions.
- **Support internal locus of control -** Experienced operators strongly desire the sense that they are in charge of the system and that the system responds to their actions. Design the system to make users the initiators of actions rather than the responders.
- **Reduce short-term memory load -** The limitation of human information processing in short-term memory requires the displays to be kept simple, multiple page displays be consolidated, window-motion frequency be reduced, and sufficient training time be allotted for codes, mnemonics, and sequences of actions.

4 **Explain the basic concepts of software design** **APR/MAY-11** **,** **NOV/DEC 2017**

Software design is a process to transform user requirements into some suitable form, which helps the programmer in software coding and implementation.

For assessing user requirements, an SRS (Software Requirement Specification) document is created whereas for coding and implementation, there is a need of more specific and detailed requirements in software terms. The output of this process can directly be used into implementation in programming languages.

Software design is the first step in SDLC (Software Design Life Cycle), which moves the concentration from problem domain to solution domain. It tries to specify how to fulfill the requirements mentioned in SRS.

Software Design Levels

Software design yields three levels of results:

- **Architectural Design -** The architectural design is the highest abstract version of the system. It identifies the software as a system with many components interacting with each other. At this level, the designers get the idea of proposed solution domain.
- **High-level Design-** The high-level design breaks the 'single entity-multiple component' concept of architectural design into less-abstracted view of sub-systems and modules and depicts their interaction with each other. High-level design focuses on how the system along with all of its components can be implemented in forms of modules. It recognizes modular structure of each sub-system and their relation and interaction among each other.
- **Detailed Design-** Detailed design deals with the implementation part of what is seen as a system and its sub-systems in the previous two designs. It is more detailed towards modules and their implementations. It defines logical structure of each module and their interfaces to communicate with other modules.

Modularization

Modularization is a technique to divide a software system into multiple discrete and independent modules, which are expected to be capable of carrying out task(s) independently. These modules may work as basic constructs for the entire software. Designers tend to design modules such that they can be executed and/or compiled separately and independently.

Modular design unintentionally follows the rules of 'divide and conquer' problem-solving strategy this is because there are many other benefits attached with the modular design of a software.

Advantage of modularization:

- Smaller components are easier to maintain
- Program can be divided based on functional aspects
- Desired level of abstraction can be brought in the program
- Components with high cohesion can be re-used again
- Concurrent execution can be made possible
- Desired from security aspect

Concurrency

Back in time, all software are meant to be executed sequentially. By sequential execution we mean that the coded instruction will be executed one after another implying only one portion of program being activated at any given time. Say, a software has multiple modules, then only one of all the modules can be found active at any time of execution.

In software design, concurrency is implemented by splitting the software into multiple independent units of execution, like modules and executing them in parallel. In other words, concurrency provides capability to the software to execute more than one part of code in parallel to each other.

It is necessary for the programmers and designers to recognize those modules, which can be made parallel execution.

Example

The spell check feature in word processor is a module of software, which runs along side the word processor itself.

Design Verification

The output of software design process is design documentation, pseudo codes, detailed logic diagrams, process diagrams, and detailed description of all functional or non-functional requirements.

The next phase, which is the implementation of software, depends on all outputs mentioned above.

It is then becomes necessary to verify the output before proceeding to the next phase. The early any mistake is detected, the better it is or it might not be detected until testing of the product. If the outputs of design phase are in formal notation form, then their associated tools for verification should be used otherwise a thorough design review can be used for verification and validation.

By structured verification approach, reviewers can detect defects that might be caused by overlooking some conditions. A good design review is important for good software design, accuracy and quality.

5 **Explain clearly the concept of coupling & cohesion? For each type of coupling give an example of two components coupled in that way?**

APRIL/MAY 2015, APRIL/MAY 2017, APRIL/MAY 2018

Describe the concept of cohesion and coupling. State the difference b/w cohesion and coupling with a suitable example.
(April/May Apr/May 2008)

Coupling and Cohesion

When a software program is modularized, its tasks are divided into several modules based on some characteristics. As we know, modules are set of instructions put together in order to achieve some tasks. They are though, considered as single entity but may refer to each other to work together. There are measures by which the quality of a design of modules and their interaction among them can be measured. These measures are called coupling and cohesion.

Cohesion

Cohesion is a measure that defines the degree of intra-dependability within elements of a module. The greater the cohesion, the better is the program design.

There are seven types of cohesion, namely –

- **Co-incidental cohesion -** It is unplanned and random cohesion, which might be the result of breaking the program into smaller modules for the sake of modularization. Because it is unplanned, it may serve confusion to the programmers and is generally not-accepted.
- **Logical cohesion -** When logically categorized elements are put together into a module, it is called logical cohesion.

- **Temporal Cohesion -** When elements of module are organized such that they are processed at a similar point in time, it is called temporal cohesion.
- **Procedural cohesion -** When elements of module are grouped together, which are executed sequentially in order to perform a task, it is called procedural cohesion.
- **Communicational cohesion -** When elements of module are grouped together, which are executed sequentially and work on same data (information), it is called communicational cohesion.
- **Sequential cohesion -** When elements of module are grouped because the output of one element serves as input to another and so on, it is called sequential cohesion.
- **Functional cohesion -** It is considered to be the highest degree of cohesion, and it is highly expected. Elements of module in functional cohesion are grouped because they all contribute to a single well-defined function. It can also be reused.

Coupling

Coupling is a measure that defines the level of inter-dependability among modules of a program. It tells at what level the modules interfere and interact with each other. The lower the coupling, the better the program.

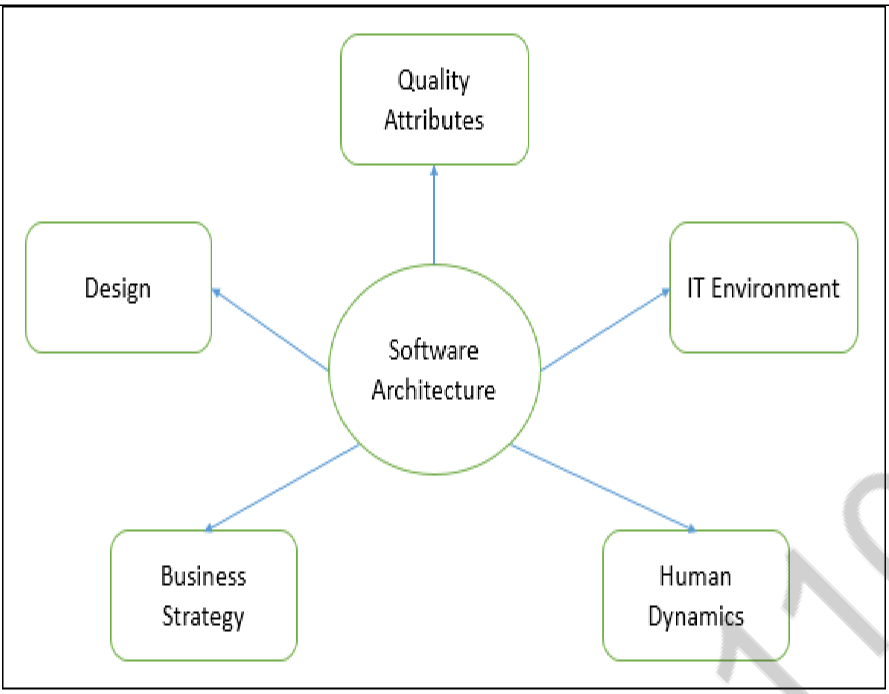
There are five levels of coupling, namely -

- **Content coupling -** When a module can directly access or modify or refer to the content of another module, it is called content level coupling.
- **Common coupling-** When multiple modules have read and write access to some global data, it is called common or global coupling.
- **Control coupling-** Two modules are called control-coupled if one of them decides the function of the other module or changes its flow of execution.
- **Stamp coupling-** When multiple modules share common data structure and work on different part of it, it is called stamp coupling.
- **Data coupling-** Data coupling is when two modules interact with each other by means of passing data (as parameter). If a module passes data structure as parameter, then the receiving module should use all its components.

Ideally, no coupling is considered to be the best.

6 **Write short notes on Architectural & component design.** **<u>MAY/JUN-15,NOV/DEC2015</u>**

The architecture of a system describes its major components, their relationships (structures), and how they interact with each other. Software architecture and design includes several contributory factors such as Business strategy, quality attributes, human dynamics, design, and IT environment.



We can segregate Software Architecture and Design into two distinct phases: Software Architecture and Software Design. In **Architecture**, nonfunctional decisions are cast and separated by the functional requirements. In Design, functional requirements are accomplished.

Software Architecture

Architecture serves as a **blueprint for a system**. It provides an abstraction to manage the system complexity and establish a communication and coordination mechanism among components.

- It defines a **structured solution** to meet all the technical and operational requirements, while optimizing the common quality attributes like performance and security.
- Further, it involves a set of significant decisions about the organization related to software development and each of these decisions can have a considerable impact on quality, maintainability, performance, and the overall success of the final product. These decisions comprise of −
    - Selection of structural elements and their interfaces by which the system is composed.
    - Behavior as specified in collaborations among those elements.
    - Composition of these structural and behavioral elements into large subsystem.
    - Architectural decisions align with business objectives.
    - Architectural styles guide the organization.

Software Design

Software design provides a **design plan** that describes the elements of a system, how they fit, and work together to fulfill the requirement of the system. The objectives of having a design plan are as follows −

- To negotiate system requirements, and to set expectations with customers, marketing, and management personnel.

- Act as a blueprint during the development process.
- Guide the implementation tasks, including detailed design, coding, integration, and testing.

It comes before the detailed design, coding, integration, and testing and after the domain analysis, requirements analysis, and risk analysis.

Goals of Architecture

The primary goal of the architecture is to identify requirements that affect the structure of the application. A well-laid architecture reduces the business risks associated with building a technical solution and builds a bridge between business and technical requirements.

Some of the other goals are as follows −

- Expose the structure of the system, but hide its implementation details.
- Realize all the use-cases and scenarios.
- Try to address the requirements of various stakeholders.
- Handle both functional and quality requirements.
- Reduce the goal of ownership and improve the organization's market position.
- Improve quality and functionality offered by the system.
- Improve external confidence in either the organization or system.

Limitations

Software architecture is still an emerging discipline within software engineering. It has the following limitations −

- Lack of tools and standardized ways to represent architecture.
- Lack of analysis methods to predict whether architecture will result in an implementation that meets the requirements.
- Lack of awareness of the importance of architectural design to software development.
- Lack of understanding of the role of software architect and poor communication among stakeholders.
- Lack of understanding of the design process, design experience and evaluation of design.

Role of Software Architect

A Software Architect provides a solution that the technical team can create and design for the entire application. A software architect should have expertise in the following areas −

Design Expertise

- Expert in software design, including diverse methods and approaches such as object-oriented design, event-driven design, etc.
- Lead the development team and coordinate the development efforts for the integrity of the design.

- Should be able to review design proposals and tradeoff among themselves.

Component-based architecture

Component-based architecture focuses on the decomposition of the design into individual functional or logical components that represent well-defined communication interfaces containing methods, events, and properties. It provides a higher level of abstraction and divides the problem into sub-problems, each associated with component partitions.

The primary objective of component-based architecture is to ensure **component reusability**. A component encapsulates functionality and behaviors of a software element into a reusable and self-deployable binary unit. There are many standard component frameworks such as COM/DCOM, JavaBean, EJB, CORBA, .NET, web services, and grid services. These technologies are widely used in local desktop GUI application design such as graphic JavaBean components, MS ActiveX components, and COM components which can be reused by simply drag and drop operation.

Component-oriented software design has many advantages over the traditional object-oriented approaches such as −

- Reduced time in market and the development cost by reusing existing components.
- Increased reliability with the reuse of the existing components.

What is a Component?

A component is a modular, portable, replaceable, and reusable set of well-defined functionality that encapsulates its implementation and exporting it as a higher-level interface.

A component is a software object, intended to interact with other components, encapsulating certain functionality or a set of functionalities. It has an obviously defined interface and conforms to a recommended behavior common to all components within an architecture.

A software component can be defined as a unit of composition with a contractually specified interface and explicit context dependencies only. That is, a software component can be deployed independently and is subject to composition by third parties.

Views of a Component

A component can have three different views − object-oriented view, conventional view, and process-related view.

Object-oriented view

A component is viewed as a set of one or more cooperating classes. Each problem domain class (analysis) and infrastructure class (design) are explained to identify all attributes and operations that apply to its implementation. It also involves defining the interfaces that enable classes to communicate and cooperate.

Conventional view

It is viewed as a functional element or a module of a program that integrates the processing logic, the internal data structures that are required to implement the processing logic and an interface that enables the component to be invoked and data to be passed to it.

Process-related view

In this view, instead of creating each component from scratch, the system is building from existing components maintained in a library. As the software architecture is formulated, components are selected from the library and used to populate the architecture.

- A user interface (UI) component includes grids, buttons referred as controls, and utility components expose a specific subset of functions used in other components.
- Other common types of components are those that are resource intensive, not frequently accessed, and must be activated using the just-in-time (JIT) approach.
- Many components are invisible which are distributed in enterprise business applications and internet web applications such as Enterprise JavaBean (EJB), .NET components, and CORBA components.

Characteristics of Components

- **Reusability** − Components are usually designed to be reused in different situations in different applications. However, some components may be designed for a specific task.
- **Replaceable** − Components may be freely substituted with other similar components.
- **Not context specific** − Components are designed to operate in different environments and contexts.
- **Extensible** − A component can be extended from existing components to provide new behavior.
- **Encapsulated** − A A component depicts the interfaces, which allow the caller to use its functionality, and do not expose details of the internal processes or any internal variables or state.
- **Independent** − Components are designed to have minimal dependencies on other components.

Principles of Component−Based Design

A component-level design can be represented by using some intermediary representation (e.g. graphical, tabular, or text-based) that can be translated into source code. The design of data structures, interfaces, and algorithms should conform to well-established guidelines to help us avoid the introduction of errors.

- The software system is decomposed into reusable, cohesive, and encapsulated component units.
- Each component has its own interface that specifies required ports and provided ports; each component hides its detailed implementation.
- A component should be extended without the need to make internal code or design modifications to the existing parts of the component.
- Depend on abstractions component do not depend on other concrete components, which increase difficulty in expendability.
- Connectors connected components, specifying and ruling the interaction among components. The interaction type is specified by the interfaces of the components.
- Components interaction can take the form of method invocations, asynchronous invocations, broadcasting, message driven interactions, data stream communications, and other protocol specific interactions.
- For a server class, specialized interfaces should be created to serve major categories of

clients. Only those operations that are relevant to a particular category of clients should be specified in the interface.

- A component can extend to other components and still offer its own extension points. It is the concept of plug-in based architecture. This allows a plugin to offer another plugin API.

7 Bring out the necessity of Real-time system design process with appropriate example?APR/MAY-12, MAY/JUNE-13, APRIL/MAY-15

Systems Design

It is a process of planning a new business system or replacing an existing system by defining its components or modules to satisfy the specific requirements. Before planning, you need to understand the old system thoroughly and determine how computers can best be used in order to operate efficiently.

System Design focuses on **how to accomplish the objective of the system**.

System Analysis and Design (SAD) mainly focuses on −

- Systems
- Processes
- Technology

What is a System?

The word System is derived from Greek word Systema, which means an organized relationship between any set of components to achieve some common cause or objective.

A system is "an orderly grouping of interdependent components linked together according to a plan to achieve a specific goal."

Constraints of a System

A system must have three basic constraints −

- A system must have some **structure and behavior** which is designed to achieve a predefined objective.
- **Interconnectivity** and **interdependence** must exist among the system components.
- The **objectives of the organization** have a **higher priority** than the objectives of its subsystems.

For example, traffic management system, payroll system, automatic library system, human resources information system.

Properties of a System

A system has the following properties −

Organization

Organization implies structure and order. It is the arrangement of components that helps to achieve predetermined objectives.

Interaction

It is defined by the manner in which the components operate with each other.

For example, in an organization, purchasing department must interact with production department and payroll with personnel department.

Interdependence

Interdependence means how the components of a system depend on one another. For proper functioning, the components are coordinated and linked together according to a specified plan. The output of one subsystem is the required by other subsystem as input.

Integration

Integration is concerned with how a system components are connected together. It means that the parts of the system work together within the system even if each part performs a unique function.

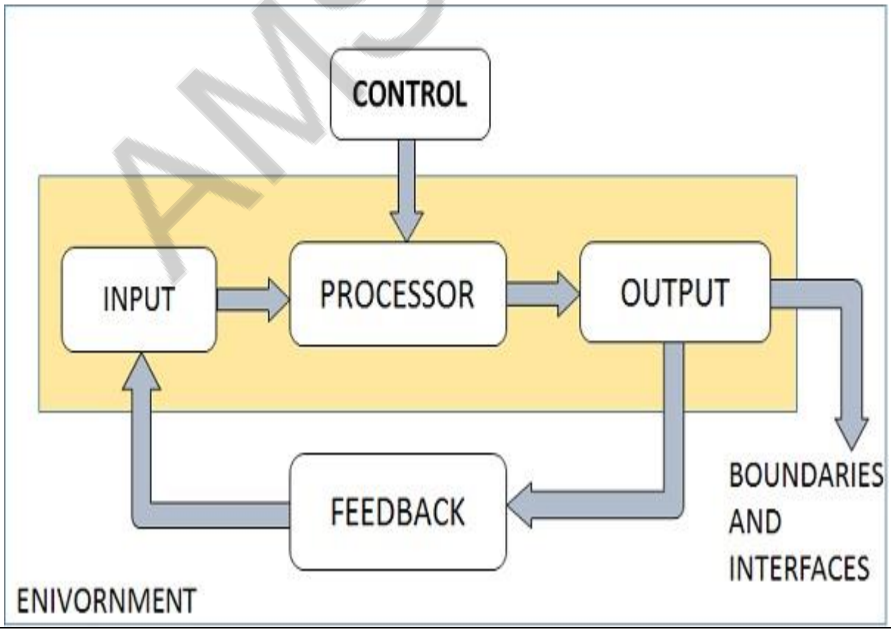
Central Objective

The objective of system must be central. It may be real or stated. It is not uncommon for an organization to state an objective and operate to achieve another.

The users must know the main objective of a computer application early in the analysis for a successful design and conversion.

Elements of a System

The following diagram shows the elements of a system −



Outputs and Inputs

- The main aim of a system is to produce an output which is useful for its user.
- Inputs are the information that enters into the system for processing.
- Output is the outcome of processing.

Processor(s)

- The processor is the element of a system that involves the actual transformation of input into output.
- It is the operational component of a system. Processors may modify the input either totally or partially, depending on the output specification.
- As the output specifications change, so does the processing. In some cases, input is also modified to enable the processor for handling the transformation.

Control

- The control element guides the system.
- It is the decision–making subsystem that controls the pattern of activities governing input, processing, and output.
- The behavior of a computer System is controlled by the Operating System and software. In order to keep system in balance, what and how much input is needed is determined by Output Specifications.

Feedback

- Feedback provides the control in a dynamic system.
- Positive feedback is routine in nature that encourages the performance of the system.
- Negative feedback is informational in nature that provides the controller with information for action.

Environment

- The environment is the “supersystem” within which an organization operates.
- It is the source of external elements that strike on the system.
- It determines how a system must function. For example, vendors and competitors of organization’s environment, may provide constraints that affect the actual performance of the business.

Boundaries and Interface

- A system should be defined by its boundaries. Boundaries are the limits that identify its components, processes, and interrelationship when it interfaces with another system.

- Each system has boundaries that determine its sphere of influence and control.
- The knowledge of the boundaries of a given system is crucial in determining the nature of its interface with other systems for successful design.

Types of Systems

The systems can be divided into the following types −

Physical or Abstract Systems

- Physical systems are tangible entities. We can touch and feel them.
- Physical System may be static or dynamic in nature. For example, desks and chairs are the physical parts of computer center which are static. A programmed computer is a dynamic system in which programs, data, and applications can change according to the user's needs.
- Abstract systems are non-physical entities or conceptual that may be formulas, representation or model of a real system.

Open or Closed Systems

- An open system must interact with its environment. It receives inputs from and delivers outputs to the outside of the system. For example, an information system which must adapt to the changing environmental conditions.
- A closed system does not interact with its environment. It is isolated from environmental influences. A completely closed system is rare in reality.

Adaptive and Non Adaptive System

- Adaptive System responds to the change in the environment in a way to improve their performance and to survive. For example, human beings, animals.
- Non Adaptive System is the system which does not respond to the environment. For example, machines.

Permanent or Temporary System

- Permanent System persists for long time. For example, business policies.
- Temporary System is made for specified time and after that they are demolished. For example, A DJ system is set up for a program and it is dissembled after the program.

Natural and Manufactured System

- Natural systems are created by the nature. For example, Solar system, seasonal system.
- Manufactured System is the man-made system. For example, Rockets, dams, trains.

Deterministic or Probabilistic System

- Deterministic system operates in a predictable manner and the interaction between system components is known with certainty. For example, two molecules of hydrogen and one molecule of oxygen makes water.

- Probabilistic System shows uncertain behavior. The exact output is not known. For example, Weather forecasting, mail delivery.

Social, Human-Machine, Machine System

- Social System is made up of people. For example, social clubs, societies.
- In Human-Machine System, both human and machines are involved to perform a particular task. For example, Computer programming.
- Machine System is where human interference is neglected. All the tasks are performed by the machine. For example, an autonomous robot.

Man–Made Information Systems

- It is an interconnected set of information resources to manage data for particular organization, under Direct Management Control (DMC).
- This system includes hardware, software, communication, data, and application for producing information according to the need of an organization.

  Man-made information systems are divided into three types −

- **Formal Information System** − It is based on the flow of information in the form of memos, instructions, etc., from top level to lower levels of management.
- **Informal Information System** − This is employee based system which solves the day to day work related problems.
- **Computer Based System** − This system is directly dependent on the computer for managing business applications. For example, automatic library system, railway reservation system, banking system, etc.

8 **What is structured design? Illustrate the structured design process from DFD to structured chart with a case study.NOV/DEC 2016,**

Structured Analysis is a development method that allows the analyst to understand the system and its activities in a logical way.

It is a systematic approach, which uses graphical tools that analyze and refine the objectives of an existing system and develop a new system specification which can be easily understandable by user.

It has following attributes −

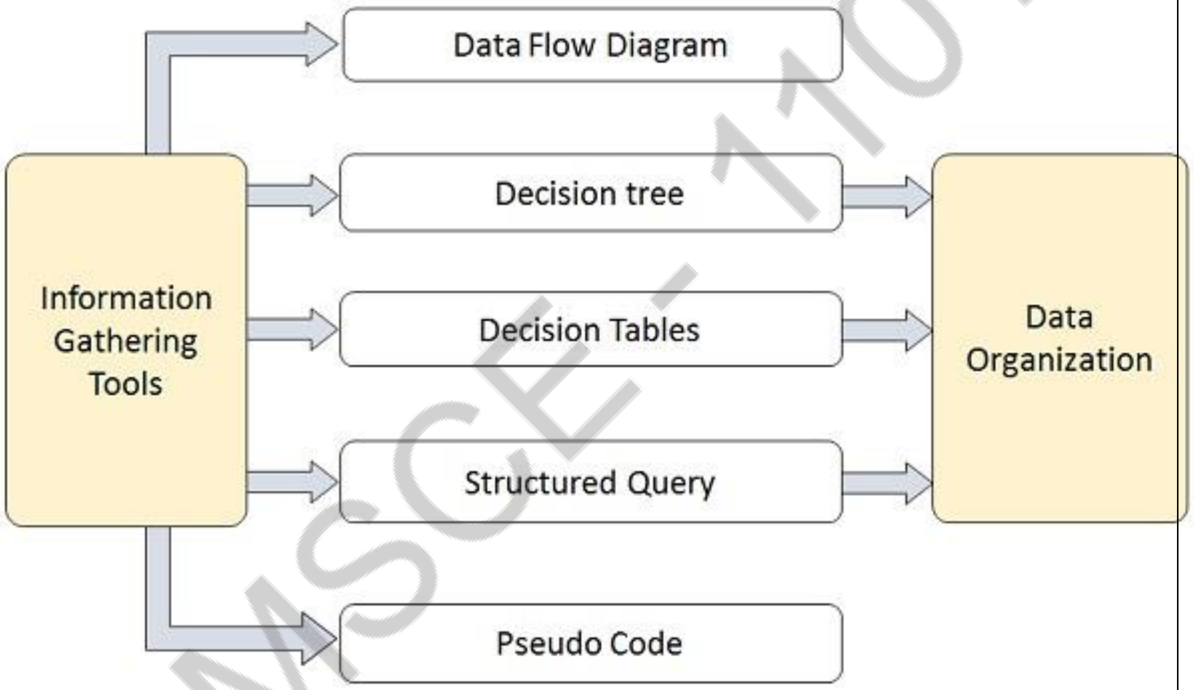
- It is graphic which specifies the presentation of application.
- It divides the processes so that it gives a clear picture of system flow.
- It is logical rather than physical i.e., the elements of system do not depend on vendor or hardware.

- It is an approach that works from high-level overviews to lower-level details.

Structured Analysis Tools

During Structured Analysis, various tools and techniques are used for system development. They are −

- Data Flow Diagrams
- Data Dictionary
- Decision Trees
- Decision Tables
- Structured English
- Pseudocode



Data Flow Diagrams (DFD) or Bubble Chart

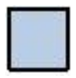
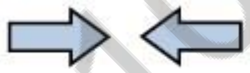
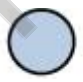
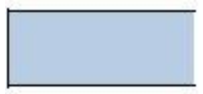
It is a technique developed by Larry Constantine to express the requirements of system in a graphical form.

- It shows the flow of data between various functions of system and specifies how the current system is implemented.
- It is an initial stage of design phase that functionally divides the requirement specifications down to the lowest level of detail.
- Its graphical nature makes it a good communication tool between user and analyst or analyst and system designer.
- It gives an overview of what data a system processes, what transformations are performed, what data are stored, what results are produced and where they flow.

Basic Elements of DFD

DFD is easy to understand and quite effective when the required design is not clear and the user wants a notational language for communication. However, it requires a large number of iterations for obtaining the most accurate and complete solution.

The following table shows the symbols used in designing a DFD and their significance −

Symbol Name	Symbol	
Square		
Arrow		
Circle		
Open Rectangle		

Types of DFD

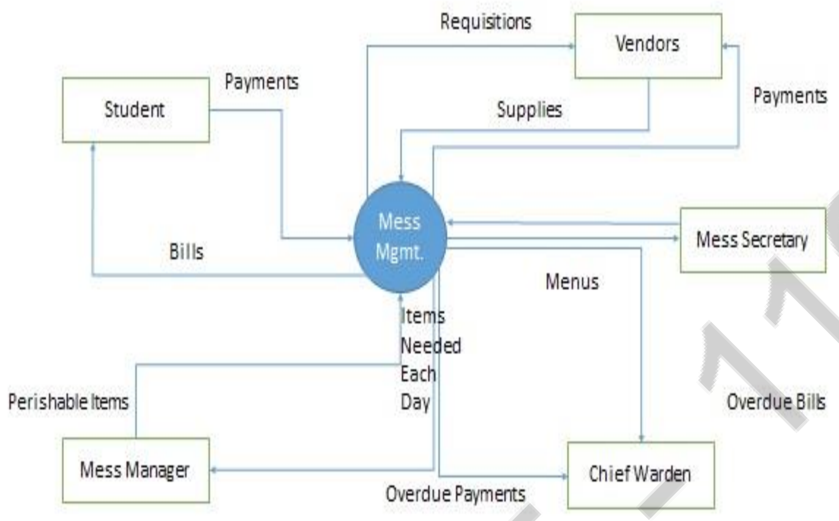
DFDs are of two types: Physical DFD and Logical DFD. The following table lists the points that differentiate a physical DFD from a logical DFD.

Physical DFD	
It is implementation dependent. It shows which functions are performed.	It is impleme processes.
It provides low level details of hardware, software, files, and people.	It explains e
It depicts how the current system operates and how a system will be implemented.	It shows hov

Context Diagram

A context diagram helps in understanding the entire system by one DFD which gives the overview of a system. It starts with mentioning major processes with little details and then goes onto giving more details of the processes with the top-down approach.

The context diagram of mess management is shown below.



Data Dictionary

A data dictionary is a structured repository of data elements in the system. It stores the descriptions of all DFD data elements that is, details and definitions of data flows, data stores, data stored in data stores, and the processes.

A data dictionary improves the communication between the analyst and the user. It plays an important role in building a database. Most DBMSs have a data dictionary as a standard feature. For example, refer the following table −

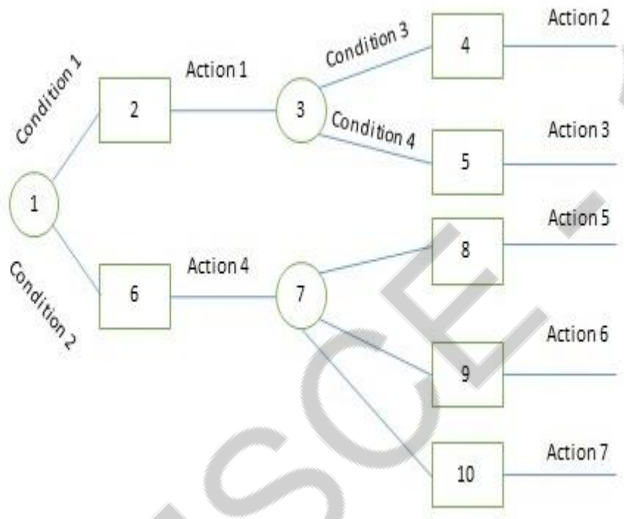
Sr.No.	**Data Name**	**Description**
1	ISBN	ISBN Number
2	TITLE	title
3	SUB	Book Subjects

4	ANAME	Author Name

Decision Trees

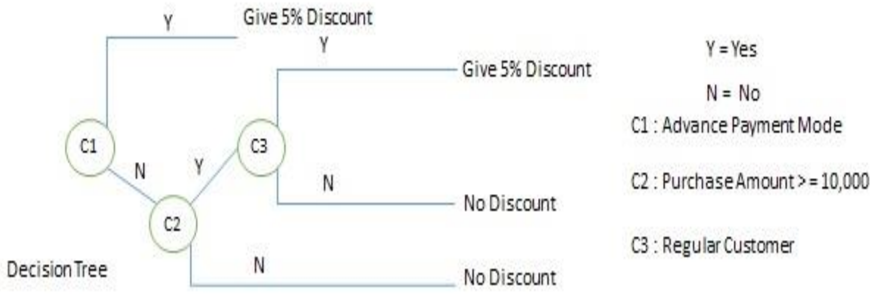
Decision trees are a method for defining complex relationships by describing decisions and avoiding the problems in communication. A decision tree is a diagram that shows alternative actions and conditions within horizontal tree framework. Thus, it depicts which conditions to consider first, second, and so on.

Decision trees depict the relationship of each condition and their permissible actions. A square node indicates an action and a circle indicates a condition. It forces analysts to consider the sequence of decisions and identifies the actual decision that must be made.



The major limitation of a decision tree is that it lacks information in its format to describe what other combinations of conditions you can take for testing. It is a single representation of the relationships between conditions and actions.

For example, refer the following decision tree −



Decision Tables

Decision tables are a method of describing the complex logical relationship in a precise manner which is easily understandable.

- It is useful in situations where the resulting actions depend on the occurrence of one or several combinations of independent conditions.
- It is a matrix containing row or columns for defining a problem and the actions.

Components of a Decision Table

- **Condition Stub** − It is in the upper left quadrant which lists all the condition to be checked.
- **Action Stub** − It is in the lower left quadrant which outlines all the action to be carried out to meet such condition.
- **Condition Entry** − It is in upper right quadrant which provides answers to questions asked in condition stub quadrant.
- **Action Entry** − It is in lower right quadrant which indicates the appropriate action resulting from the answers to the conditions in the condition entry quadrant.

The entries in decision table are given by Decision Rules which define the relationships between combinations of conditions and courses of action. In rules section,

- Y shows the existence of a condition.
- N represents the condition, which is not satisfied.
- A blank - against action states it is to be ignored.
- X (or a check mark will do) against action states it is to be carried out.

9 **(a) Describe golden rules for interface design <u>NOV/DEC 2016</u>**

User Interface Golden rules

The following rules are mentioned to be the golden rules for GUI design, described by Shneiderman and Plaisant in their book (Designing the User Interface).

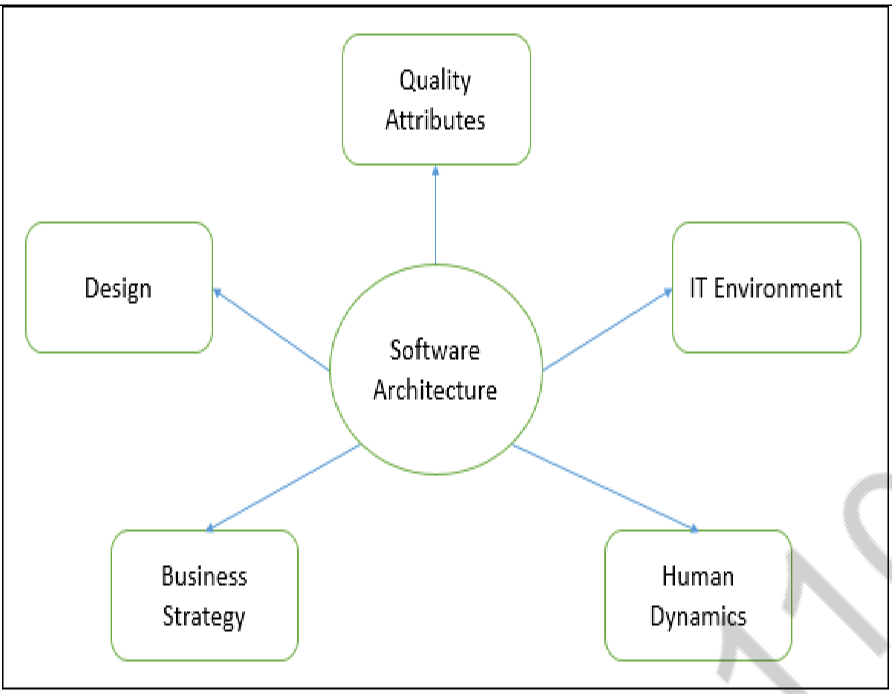
- **Strive for consistency -** Consistent sequences of actions should be required in similar situations. Identical terminology should be used in prompts, menus, and help screens. Consistent commands should be employed throughout.
- **Enable frequent users to use short-cuts -** The user's desire to reduce the number of interactions increases with the frequency of use. Abbreviations, function keys, hidden commands, and macro facilities are very helpful to an expert user.
- **Offer informative feedback -** For every operator action, there should be some system feedback. For frequent and minor actions, the response must be modest, while for infrequent and major actions, the response must be more substantial.
- **Design dialog to yield closure -** Sequences of actions should be organized into groups with a beginning, middle, and end. The informative feedback at the completion of a group of actions gives the operators the satisfaction of accomplishment, a sense of relief, the signal to drop contingency plans and options from their minds, and this indicates that the way ahead is clear to prepare for the next group of actions.
- **Offer simple error handling -** As much as possible, design the system so the user will not make a serious error. If an error is made, the system should be able to detect it and offer simple, comprehensible mechanisms for handling the error.
- **Permit easy reversal of actions -** This feature relieves anxiety, since the user knows that errors can be undone. Easy reversal of actions encourages exploration of unfamiliar options. The units of reversibility may be a single action, a data entry, or a complete group of actions.
- **Support internal locus of control -** Experienced operators strongly desire the sense that they are in charge of the system and that the system responds to their actions. Design the system to make users the initiators of actions rather than the responders.

(b) Reduce short-term memory load - The limitation of human information processing in short-term memory requires the displays to be kept simple, multiple page displays be consolidated, window-motion frequency be reduced, and sufficient training time be allotted for codes, mnemonics, and sequences of actions

Refer class notes

10 **What is software architecture ? Describe in detail different types of software architectural styles with illustrations. <u>APRIL/MAY 2017, APRIL/MAY 2018</u>**

The architecture of a system describes its major components, their relationships (structures), and how they interact with each other. Software architecture and design includes several contributory factors such as Business strategy, quality attributes, human dynamics, design, and IT environment.



We can segregate Software Architecture and Design into two distinct phases: Software Architecture and Software Design. In **Architecture**, nonfunctional decisions are cast and separated by the functional requirements. In Design, functional requirements are accomplished.

Software Architecture

Architecture serves as a **blueprint for a system**. It provides an abstraction to manage the system complexity and establish a communication and coordination mechanism among components.

- It defines a **structured solution** to meet all the technical and operational requirements, while optimizing the common quality attributes like performance and security.
- Further, it involves a set of significant decisions about the organization related to software development and each of these decisions can have a considerable impact on quality, maintainability, performance, and the overall success of the final product. These decisions comprise of −
    - Selection of structural elements and their interfaces by which the system is composed.
    - Behavior as specified in collaborations among those elements.
    - Composition of these structural and behavioral elements into large subsystem.
    - Architectural decisions align with business objectives.
    - Architectural styles guide the organization.

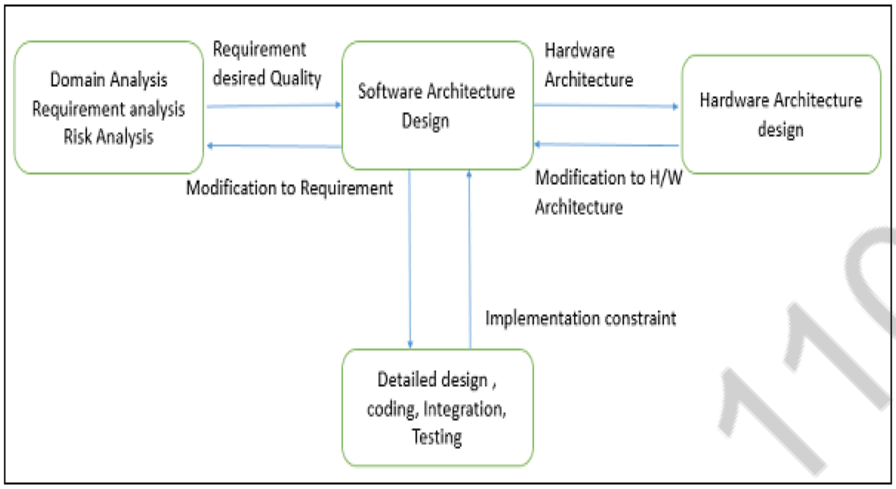
Software Design

Software design provides a **design plan** that describes the elements of a system, how they fit, and work together to fulfill the requirement of the system. The objectives of having a design plan are as follows −

- To negotiate system requirements, and to set expectations with customers, marketing, and management personnel.

- Act as a blueprint during the development process.
- Guide the implementation tasks, including detailed design, coding, integration, and testing.

It comes before the detailed design, coding, integration, and testing and after the domain analysis, requirements analysis, and risk analysis.



Domain Analysis
Requirement analysis
Risk Analysis
Requirement
desired Quality
Modification to Requirement
Software Architecture Design
Hardware Architecture
Modification to H/W Architecture
Hardware Architecture design
Implementation constraint
Detailed design , coding, Integration, Testing

Goals of Architecture

The primary goal of the architecture is to identify requirements that affect the structure of the application. A well-laid architecture reduces the business risks associated with building a technical solution and builds a bridge between business and technical requirements.

Some of the other goals are as follows −

- Expose the structure of the system, but hide its implementation details.
- Realize all the use-cases and scenarios.
- Try to address the requirements of various stakeholders.
- Handle both functional and quality requirements.
- Reduce the goal of ownership and improve the organization’s market position.
- Improve quality and functionality offered by the system.
- Improve external confidence in either the organization or system.

Limitations

Software architecture is still an emerging discipline within software engineering. It has the following limitations −

- Lack of tools and standardized ways to represent architecture.
- Lack of analysis methods to predict whether architecture will result in an implementation that meets the requirements.
- Lack of awareness of the importance of architectural design to software development.
- Lack of understanding of the role of software architect and poor communication among

stakeholders.

- Lack of understanding of the design process, design experience and evaluation of design.

11 What is the purpose of DFD ?What are the compoenets of DFD? Construct DFD for the following system..

An online shopping system for xyz provides many services and

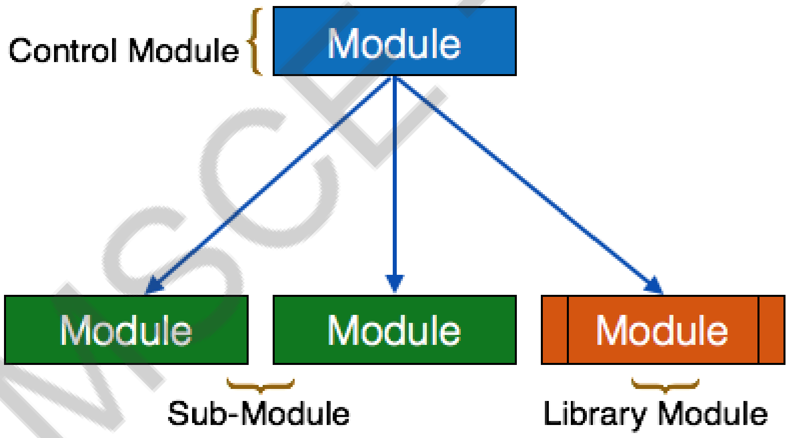
benefits to its members and staffs. <u>APRIL/MAY 2018</u>

structure chart is a chart derived from Data Flow Diagram. It represents the system in more detail than DFD. It breaks down the entire system into lowest functional modules, describes functions and sub-functions of each module of the system to a greater detail than DFD.

Structure chart represents hierarchical structure of modules. At each layer a specific task is performed.

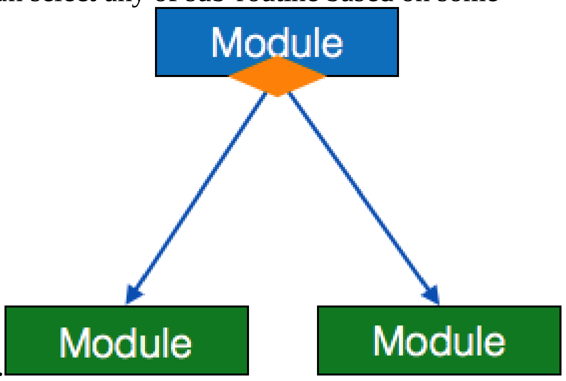
Here are the symbols used in construction of structure charts -

- **Module** - It represents process or subroutine or task. A control module branches to more than one sub-module. Library Modules are re-usable and invokable from any



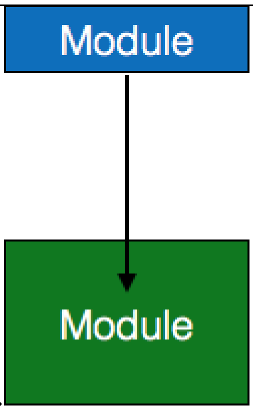
module.

- **Condition** - It is represented by small diamond at the base of module. It depicts that control module can select any of sub-routine based on some



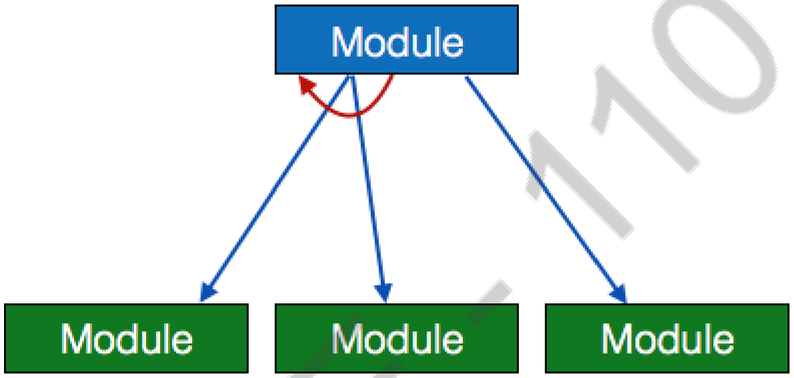
condition.

- **Jump** - An arrow is shown pointing inside the module to depict that the control will jump



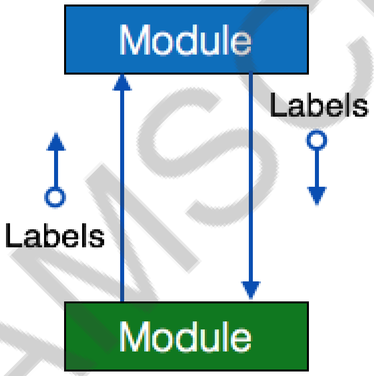
in the middle of the sub-module.

- **Loop -** A curved arrow represents loop in the module. All sub-modules covered by loop repeat execution of



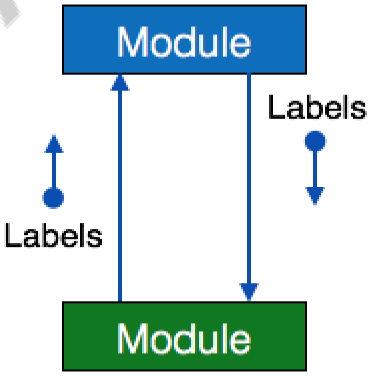
module.

- **Data flow -** A directed arrow with empty circle at the end represents data



flow.

- **Control flow -** A directed arrow with filled circle at the end represents control



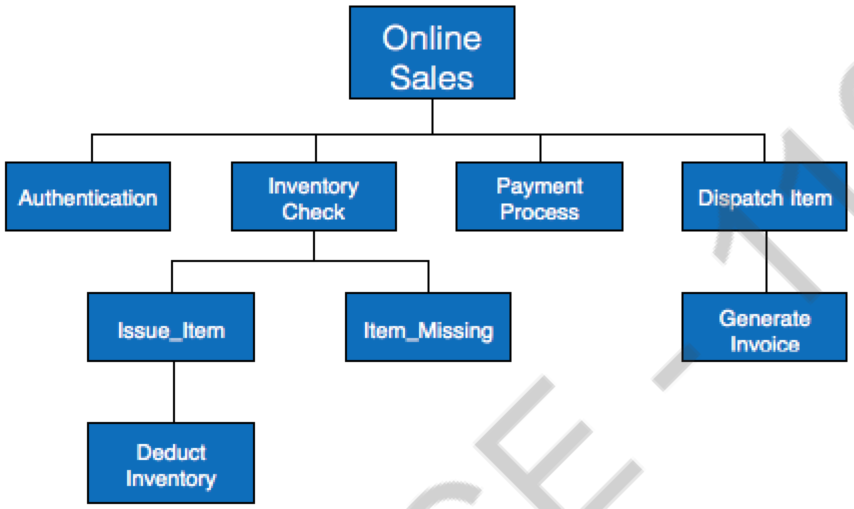
flow.

HIPO Diagram

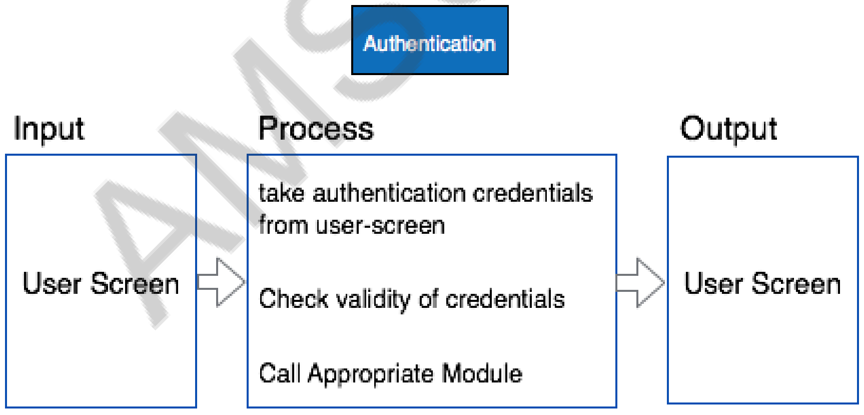
HIPO (Hierarchical Input Process Output) diagram is a combination of two organized method to analyze the system and provide the means of documentation. HIPO model was developed by IBM in year 1970.

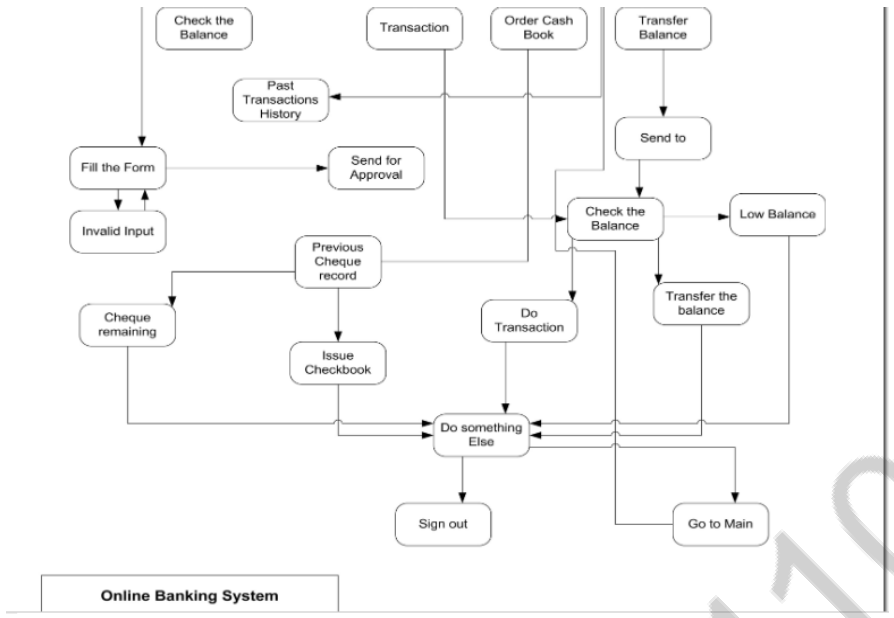
HIPO diagram represents the hierarchy of modules in the software system. Analyst uses HIPO diagram in order to obtain high-level view of system functions. It decomposes functions into sub-functions in a hierarchical manner. It depicts the functions performed by system.

HIPO diagrams are good for documentation purpose. Their graphical representation makes it easier for designers and managers to get the pictorial idea of the system structure.



In contrast to IPO (Input Process Output) diagram, which depicts the flow of control and data in a module, HIPO does not provide any information about data flow or control flow.


Authentication
Input
Process
Output
User Screen
take authentication credentials from user-screen
Check validity of credentials
Call Appropriate Module
User Screen



12 Describe in detail about architectural styles?

Architectural styles define the components and connectors (‘what?’)

• Less domain specific

• Good architecture makes use of design patterns (on a more finegranular level)

• Usually domain independent

An architectural style is a named collection of architectural design decisions that

− are applicable in a given development context

− constrain architectural design decisions that are specific to a particular system within that context

− elicit beneficial qualities in each resulting system

• Reflect less domain specificity than architectural patterns

• Useful in determining everything from subroutine structure to top-level application structure

• Many styles exist and we will discuss them in detail in the next lecture

Benefits of Using Styles

Reuse

• Design: Well-understood solutions applied to new problems

• Code: Shared implementations of invariant aspects of a style

Understandability of system organization

• A phrase such as “client-server” conveys a lot of information

Interoperability

• Supported by style standardization

• Style-specificity

• Analyses: enabled by the constrained design space

• Visualizations: depictions matching engineers' mental models

Basic Properties of Styles

A vocabulary of design elements

• Component and connector types; data elements

− e.g., pipes, filters, objects, servers

"Architectural styles define the components and connectors"

• A software connector is an architectural building block tasked with effecting and regulating interactions among components (Taylor, Medvidovic, Dashofy)

• Procedure call connectors

• Shared memory connectors

• Message passing connectors

• Streaming connectors

• Distribution connectors

• Wrapper/adaptor connectors

A set of configuration rules

• Topological constraints that determine allowed compositions of elements

− e.g., a component may be connected to at most two other components

A semantic interpretation

• Compositions of design elements have well-defined meanings

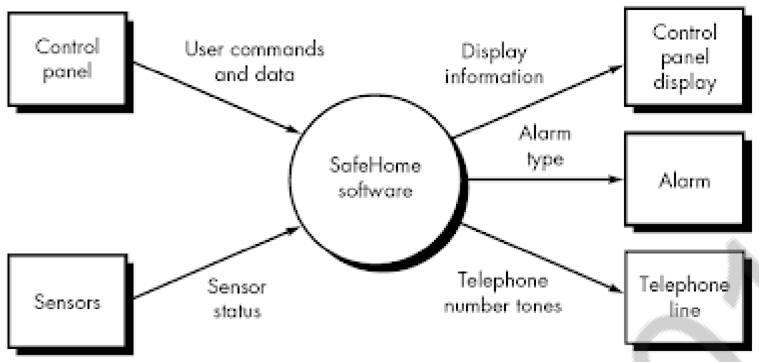
• Possible analyses of systems built in a style

13 Explain transform mapping with suitable example and design steps involved in it.(Nov/Dec 2012)

Transform mapping is a set of design steps that allows a DFD with transform flow characteristics to be mapped into a specific architectural style. In this section transform mapping is described by applying design steps to an example system—a portion of the SafeHome security software.

An Example

The SafeHome security system is representative of many computer-based products and systems in use today. The product monitors the real world and reacts to changes that it encounters. It also interacts with a user through a series of typed inputs and alphanumeric displays. The level 0 data flow diagram for SafeHome, is shown in figure

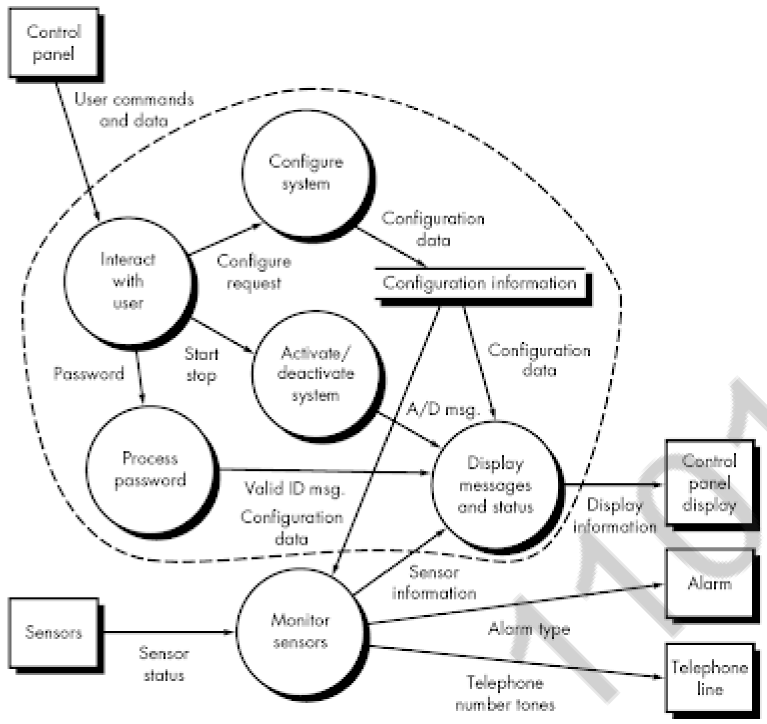


During requirements analysis, more detailed flow models would be created for SafeHome. In addition, control and process specifications, a data dictionary, and various behavioral models would also be created.

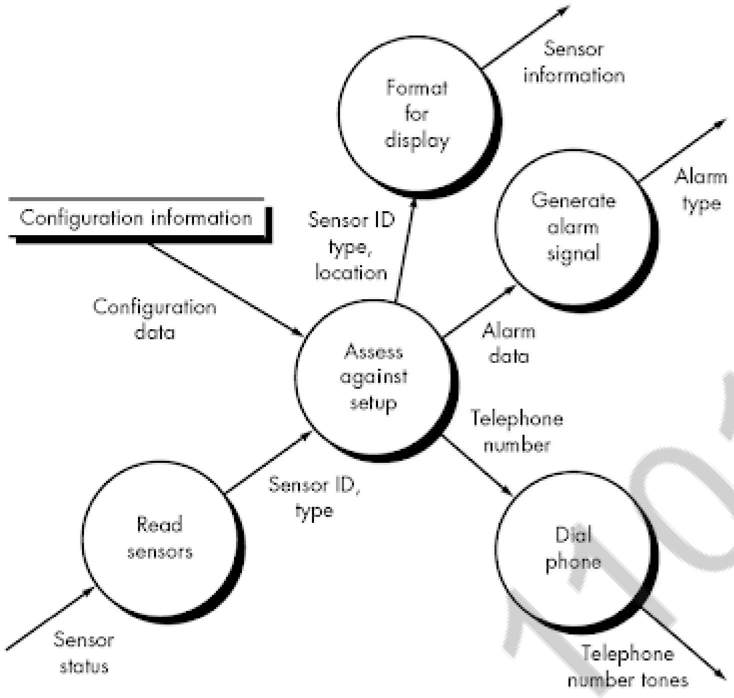
Design Steps

The preceding example will be used to illustrate each step in transform mapping. The steps begin with a re-evaluation of work done during requirements analysis and then move to the design of the software architecture.

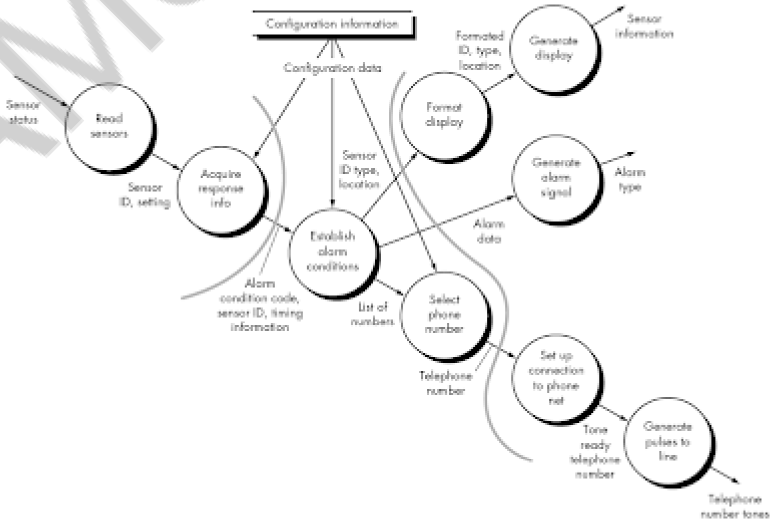
Step 1. Review the fundamental system model. The fundamental system model encompasses the level 0 DFD and supporting information. In actuality, the design step begins with an evaluation of both the System Specification and the Software Requirements Specification. Both documents describe information flow and structure at the software interface. Figure 1 and 2 depict level 0 and level 1 data flow for the SafeHome software.



Step 2. Review and refine data flow diagrams for the software. Information obtained from analysis models contained in the Software Requirements Specification is refined to produce greater detail. For example, the level 2 DFD for monitor sensors is examined, and a level 3 data flow diagram is derived . At level 3, each transform in the data flow diagram exhibits relatively high cohesion. That is, the process implied by a transform performs a single, distinct function that can be implemented as a module9 in the SafeHome software. Therefore, the DFD in figure contains sufficient detail for a "first cut" at the design of architecture for the monitor sensors subsystem, and we proceed without further refinement.



Step 3. Determine whether the DFD has transform or transaction flow characteristics. In general, information flow within a system can always be represented as transform. However, when an obvious transaction characteristic is encountered, a different design mapping is recommended. In this step, the designer selects global (softwarewide) flow characteristics based on the prevailing nature of the DFD. In addition, local regions of transform or transaction flow are isolated. These subflows can be used to refine program architecture derived from a global characteristic described previously. For now, we focus our attention only on the monitor sensors subsystem data flow depicted in figure.



Evaluating the DFD , we see data entering the software along one incoming path and exiting along three outgoing paths. No distinct transaction center is implied (although the transform establishes alarm conditions that could be perceived as such). Therefore, an overall transform characteristic will be assumed for information flow.

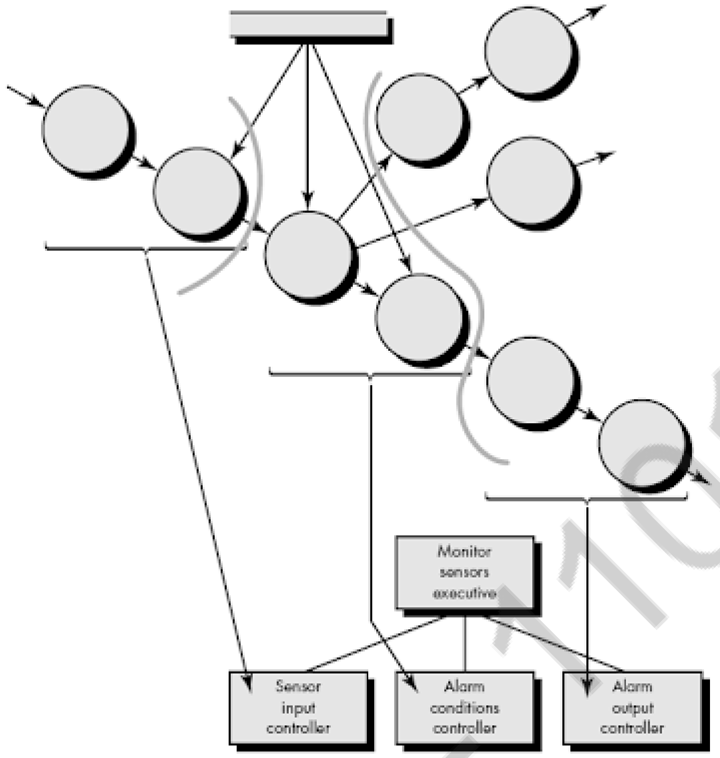
Step 4. Isolate the transform center by specifying incoming and outgoing flow boundaries. In the preceding section incoming flow was described as a path in which information is converted from external to internal form; outgoing flow converts from internal to external form. Incoming and outgoing flow boundaries are open to interpretation. That is, different designers may select slightly different points in the flow as boundary locations. In fact, alternative design solutions can be derived by varying the placement of flow boundaries. Although care should be taken when boundaries are selected, a variance of one bubble along a flow path will generally have little impact on the final program structure.

Flow boundaries for the example are illustrated as shaded curves running vertically through the flow in the above figure. The transforms (bubbles) that constitute the transform center lie within the two shaded boundaries that run from top to bottom in the figure. An argument can be made to readjust a boundary (e.g, an incoming flow boundary separating read sensors and acquire response info could be proposed). The emphasis in this design step should be on selecting reasonable boundaries, rather than lengthy iteration on placement of divisions.

Step 5. Perform "first-level factoring." Program structure represents a top-down distribution of control. Factoring results in a program structure in which top-level modules perform decision making and low-level modules perform most input, computation, and output work. Middle-level modules perform some control and do moderate amounts of work.

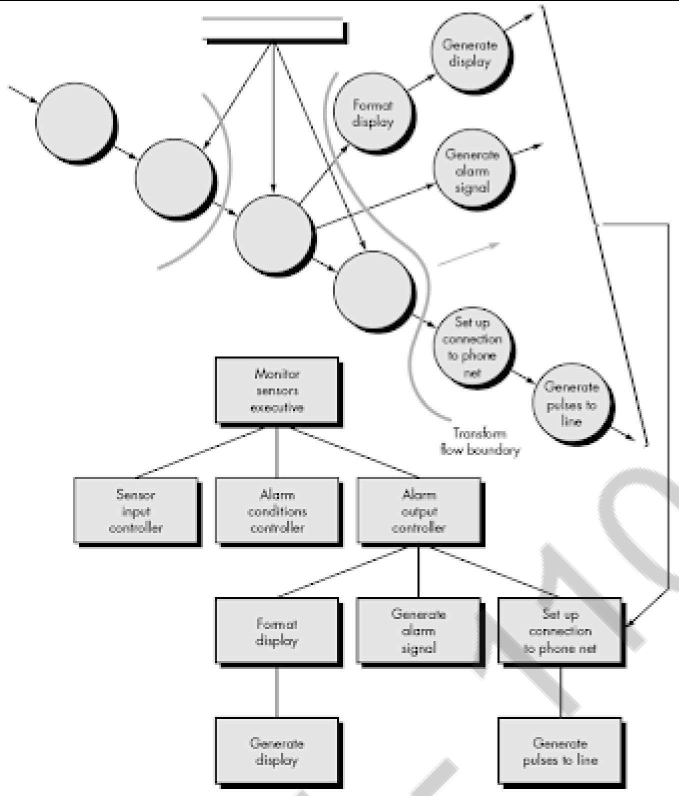
When transform flow is encountered, a DFD is mapped to a specific structure (a call and return architecture) that provides control for incoming, transform, and outgoing information processing. This first-level factoring for the monitor sensors subsystem is illustrated in figure below. A main controller (called monitor sensors executive) resides at the top of the program structure and coordinates the following subordinate control functions:

- An incoming information processing controller, called sensor input controller, coordinates receipt of all incoming data.
- A transform flow controller, called alarm conditions controller, supervises all operations on data in internalized form (e.g., a module that invokes various data transformation procedures).
- An outgoing information processing controller, called alarm output controller, coordinates production of output information.



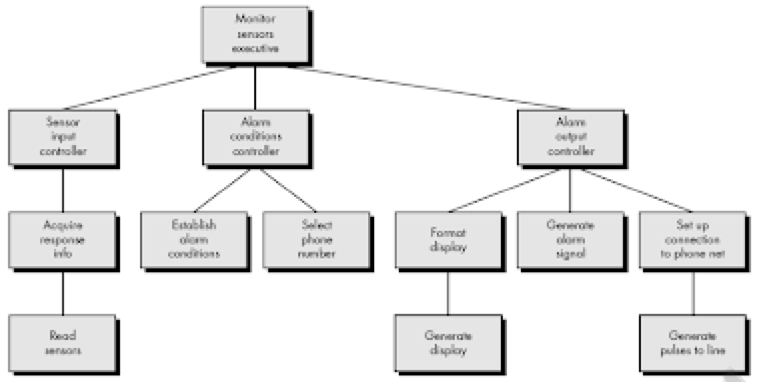
Although a three-pronged structure is implied by figure complex flows in large systems may dictate two or more control modules for each of the generic control functions described previously. The number of modules at the first level should be limited to the minimum that can accomplish control functions and still maintain good coupling and cohesion characteristics.

Step 6. Perform "second-level factoring." Second-level factoring is accomplished by mapping individual transforms (bubbles) of a DFD into appropriate modules within the architecture. Beginning at the transform center boundary and moving outward along incoming and then outgoing paths, transforms are mapped into subordinate levels of the software structure. The general approach to second-level factoring for the SafeHome data flow is illustrated in figure.



Although the figure illustrates a one-to-one mapping between DFD transforms and software modules, different mappings frequently occur. Two or even three bubbles can be combined and represented as one module (recalling potential problems with cohesion) or a single bubble may be expanded to two or more modules. Practical considerations and measures of design quality dictate the outcome of secondlevel factoring. Review and refinement may lead to changes in this structure, but it can serve as a "first-iteration" design.

Second-level factoring for incoming flow follows in the same manner. Factoring is again accomplished by moving outward from the transform center boundary on the incoming flow side. The transform center of monitor sensors subsystem software is mapped somewhat differently. Each of the data conversion or calculation transforms of the transform portion of the DFD is mapped into a module subordinate to the transform controller. A completed first-iteration architecture is shown in figure.



The modules mapped in the preceding manner and shown in figure represent an initial design of software architecture. Although modules are named in a manner that implies function, a brief processing narrative (adapted from the PSPEC created during analysis modeling) should be written for each. The narrative describes
• Information that passes into and out of the module (an interface description).
• Information that is retained by a module, such as data stored in a local data structure.
• A procedural narrative that indicates major decision points and tasks.
• A brief discussion of restrictions and special features (e.g., file I/O, hardwaredependent characteristics, special timing requirements).
The narrative serves as a first-generation Design Specification. However, further refinement and additions occur regularly during this period of design.

Step 7. Refine the first-iteration architecture using design heuristics for improved software quality. A first-iteration architecture can always be refined by applying concepts of module independence . Modules are exploded or imploded to produce sensible factoring, good cohesion, minimal coupling, and most important, a structure that can be implemented without difficulty, tested without confusion, and maintained without grief.

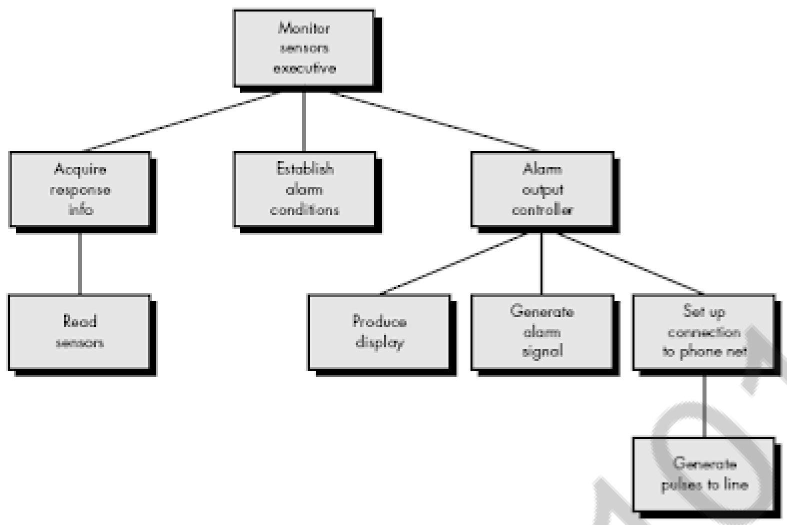
Refinements are dictated by the analysis and assessment methods described briefly , as well as practical considerations and common sense. There are times, for example, when the controller for incoming data flow is totally unnecessary, when some input processing is required in a module that is subordinate to the transform controller, when high coupling due to global data cannot be avoided, or when
optimal structural characteristics cannot be achieved. Software requirements coupled with human judgment is the final arbiter. Many modifications can be made to the first iteration architecture developed for the SafeHome monitor sensors subsystem. Among many possibilities,

1. The incoming controller can be removed because it is unnecessary when a single incoming flow path is to be managed.

2. The substructure generated from the transform flow can be imploded into the module establish alarm conditions (which will now include the processing implied by select phone number). The transform controller will not be needed and the small decrease in cohesion is tolerable.

3. The modules format display and generate display can be imploded (we assume that display formatting is quite simple) into a new module called produce display.

The refined software structure for the monitor sensors subsystem is shown in figure.



The objective of the preceding seven steps is to develop an architectural representation of software. That is, once structure is defined, we can evaluate and refine software architecture by viewing it as a whole. Modifications made at this time require little additional work, yet can have a profound impact on software quality.

The reader should pause for a moment and consider the difference between the design approach described and the process of "writing programs." If code is the only representation of software, the developer will have great difficulty evaluating or refining at a global or holistic level and will, in fact, have difficulty "seeing the forest for the trees."

14 Explain the design principles in detail

Design principles are widely applicable laws, guidelines, biases and design considerations which designers apply with discretion. Professionals from many disciplines—e.g., behavioral science, sociology, physics and ergonomics—provided the foundation for design principles via their accumulated knowledge and experience.

Types of Design Principles

Designers use principles such as **visibility**, **findability** and **learnability** to address basic human behaviors. We **use some design principles to guide actions**. **Perceived affordances** such as buttons are an example. That way, we **put users in control in seamless experiences**.

Usability kingpin Jakob Nielsen identified ten “commandments”:

Keep users informed of system status with constant feedback.
Set information in a logical, natural order.
Ensure users can easily undo/redo actions.
Maintain consistent standards so users know what to do next without having to learn new toolsets.
Prevent errors if possible; wherever you can’t do this, *warn* users before they commit to actions.

Don't make users remember information – keep options, etc. *visible*.
Make systems flexible so novices and experts can *choose* to do more or less on them.
Design with aesthetics and minimalism in mind – don't clutter with unnecessary items.
Provide plain-language error messages to pinpoint problems and likely solutions.
Offer easy-to-search troubleshooting resources, if needed.

Empathy expert Whitney Hess adds:

1. **Don't interrupt or give users obstacles –** make obvious pathways which offer an easy ride.

2. **Offer few options** – don't hinder users with nice-to-haves; give them needed alternatives instead.

3. **Reduce distractions** – let users perform tasks consecutively, not simultaneously.

4. **Cluster related objects together.**

5. **Have an easy-to-scan visual hierarchy that reflects users' needs**, with commonly used items handily available.

6. **Make things easy to find.**

7. **Show users where they've come from** and where they're headed with signposts/cues.

8. **Provide context** – show how everything interconnects.

9. **Avoid jargon**.

10. **Make designs efficient and streamlined.**

11. **Use defaults wisely** – when you offer predetermined, well-considered options, you help minimize users' decisions and increase efficiency.

12. **Don't delay users** – ensure quick interface responses.

13. **Focus on emotion** – pleasure of use is as vital as ease of use; arouse users' passion to increase engagement.

14. **Use "less is more"** – make everything count in the design. If functional and aesthetic elements don't add to the user experience, forget them.

15. **Be consistent with navigational mechanisms**, organizational structure, etc., to make a stable, reliable and predictable design.

16. **Create a good first impression**.

17. **Be trustworthy and credible** – identify yourself through your design to assure users and eliminate uncertainty.

. **UNIT – 4 PART –A**

S.NO	QUESTIONS
1	**What are the characteristics of good tester? NOV/DEC- 10,MAY/JUN-13** All tests should be traceable to customer requirements. Tests should be planned long before testing begins. The Pareto principle applies to software testing.
2	**Define software testing?** Software testing is a critical element of software quality assurance and represents the ultimate review of specification, design, and coding.
3	**What are the objectives of testing?** i. Testing is a process of executing a program with the intend of finding an error. ii. A good test case is one that has high probability of finding an undiscovered error. iii. A successful test is one that uncovers as an-yet undiscovered error.
4	**What is integration testing?and What are the approaches of integration testing?APR/MAY-11** In this testing the individual software modules are combined and tested as a group. It occurs after unit testing & before system testing. 1. The non-incremental testing. 2. Incremental testing.
5	**What is regression testing? APR/MAY-15 , NOV/DEC- 11,NOV/DEC 2013,** It tends to verify the software application after a change has been made. It seeks to uncover software errors by partially retesting a modified program.

6 **Distinguish between stress and load testing**

Stress testing is subjecting a system to an unreasonable load while denying it the resources (e.g., RAM, disc, mips, interrupts, etc.) needed to process that load.

Load testing is subjecting a system to a statistically representative (usually) load. The two main reasons for using such loads is in support of software reliability testing and in performance testing. The term "load testing" by itself is too vague and imprecise to warrant use.

7 **Define black box testing? APR/MAY-12,MAY/JUN-13**

A black-box tests are used to demonstrate that software functions are operational, that input is properly accepted and output is correctly produced, and that the integrity of external information.

8 **What is boundary condition testing? APR/MAY-12**

It is tested using boundary value analysis. (check BVA – 16 mark question)

9 **How is software testing results related to the reliability of software? NOV/DEC-12**

Applying fault avoidance, fault tolerance and fault detection for the project helps to achieve reliability of software.

10 **What is big-bang approach? NOV/DEC-12**

Big bang approach talks about testing as the last phase of development. All the defects are found in the last phase and cost of rework can be huge.

11 **Why does software fail after it has passed from acceptance testing?** **<u>APR/MAY 2016</u>**

Each acceptance test represents some expected result from the system. Customers are responsible for verifying the correctness of the acceptance tests and reviewing test scores to decide which failed tests are of highest priority. Acceptance tests are also used as regression tests prior to a production release. A user story is not considered complete until it has passed its acceptance tests. This means that new acceptance tests must be created for each iteration or the development team will
report zero progress.

12 **What are the objectives of testing?**

xii. Testing is a process of executing a program with the intend of finding an error.

xiii. A good test case is one that has high probability of finding an undiscovered error.

xiv. A successful test is one that uncovers as an-yet undiscovered
error.

13 **What are the testing principles the software engineer must apply while performing the software testing? MAY/JUNE 2016, APRIL/MAY 2018**

i. All tests should be traceable to customer requirements.

ii. Tests should be planned long before testing begins.

iii. The pareto principle can be applied to software testing-80%
of all
errors uncovered during testing will likely be traceable to 20% of all program modules.

iv. Testing should begin "in the small" and progress toward testing "in the large".

v. Exhaustive testing is not possible.

vi. To be most effective, an independent third party should conduct testing.

14 What are the two levels of testing?

i. Component testing Individual components are tested. Tests are derived from developer‟s experience.

ii. System Testing The group of components are integrated to create a system or sub-system is done.These tests are based on the system specification.

15 What are the various testing activities?

iii. Test planning

iv. Test case design

v. Test execution

vi. Data collection

vii. Effective evaluation

16 What is equivalence partitioning?

Equivalence partitioning is a black box technique that divides the input domain into classes of data. From this data test cases can be derived. Equivalence class represents a set of valid or invalid states for
input conditions.

17 What methods are used for breaking very long expression and statements? NOV/DEC2016

Refactoring is done to break long expression and ststements.

16 What are the various testing strategies for conventional software?

i. Unit testing ii. Integration testing. iii. Validation testing. iv. System testing.

18 How can refactoring be made more effective? APR/MAY 2016

Refactoring improves nonfunctional attributes of the software. Advantages include improved code readability and reduced complexity; these can improve source-codemaintainability and create a more
expressive internal architecture or object model to improve extensibility

19 How will you test a simple loop <u>NOV/DEC 2015</u>

- A simple loop is tested in the following way:
- Skip the entire loop.
- Make 1 pass through the loop.
- Make 2 passes through the loop.
- Make x passes through the loop where x<y, n is the maximum number of passes through the loop.
- Make "y","y-1","y+1" passes through the loop where "y" is the maximum number of allowable passes through the loop.

20 What are the conditions exists after performing validation testing?

After performing the validation testing there exists two conditions.

- The function or performance characteristics are according to the specifications and are accepted.
- The requirement specifications are derived and the deficiency list is created. The deficiencies then can be resolved by establishing
  the proper communication with the customer.

21 Distinguish between alpha and beta testing. <u>MAY/JUNE 2016</u>

- Alpha and beta testing are the types of acceptance testing.
- Alpha test: The alpha testing is attesting in which the version of complete software is tested by the customer under the supervision of developer. This testing is performed at developer's site.
- Beta test: The beta testing is a testing in which the version of the software is tested by the customer without the developer being
  present. This testing is performed at customer's site.

22 **What are the various types of system testing?**

to check whether it meets the check whether it meets the specific requirements ofthe business needs.

particular phase

3. Recovery testing – is intended to check the system‟ s ability to recover from failures.

4. Security testing – verifies that system protection mechanism prevent improper penetration or data alteration.

5. Stress testing – Determines breakpoint of a system to establish maximum service level.

6. Performance testing – evaluates the run time performance of the software, especially real-time software.

23 **Define debugging** and **What are the common approaches in debugging?**

Debugging is defined as the process of removal of defect. It occurs as a consequence of successful testing

Brute force method: The memory dumps and run-time tracks are examined and program with write statements is loaded to obtain clues to error causes.

Back tracking method: The source code is examined by looking backwards from symptom to potential causes of errors.

Cause elimination method: This method uses binary partitioning to reduce the number of locations where errors can exists.

24 **Distinguish between verification and validation. <u>NOV/DEC2016,</u> <u>NOV/DEC 2017,</u> <u>APRIL/MAY 2018</u>**

Verification	**Validation**
Evaluates the intermediary products	Evaluates the final product to

Checks whether the product is built as per the specified requirement and design specification.	It determines whether the software is fit for use and satisfy the business need.
right"?	right
tware	software
Involves all the static testing techniques inspection and walkthrough	Includes all the dynamic testing like smoke, regression, functional, systems and UAT

25 **What is meant by structural testing?**

In structural testing derivation of test cases is according to program structure. Hence knowledge of the program is used to identify additional test cases.

26 **What is the need for regression testing? <u>APR/MAY 2015</u>**

The purpose of regression testing is to confirm that a recent program or code change has not adversely affected existing features. Regression testing is nothing but full or partial selection of already executed test cases which are re-executed to ensure existing functionalities work fine.

27 **Write about drivers and stubs. NOV/DEC 2017**

Drivers and stub software need to be developed to test incompatible software.

The "driver" is a program that accepts the test data and prints the relevant results.

The "stub" is a subprogram that uses the module interfaces and performs the minimal data manipulation if required.

28 **What is cyclomatic complexity?**

Cyclomatic complexity is software metric that gives the quantitative Measure of logical complexity of the program.

29 **How to compute the cyclomatic complexity?**

The cyclomatic complexity can be computed by any one of the following ways. 1. The numbers of regions of the flow graph correspond to the cyclomatic complexity.

2. Cyclomatic complexity (G), for the flow graph G, is defined as: V(G)=E-N+2, E -- number of flow graph edges, N -- number of flow graph nodes

3. V(G) = P+1 Where P is the number of predicate nodes contained in the flow graph.

30 **List out the applications of GUI? April /May 2015**

GUI-Graphical User Interface- is a type of interface that allows users to interact with electronic devices through graphical icons and visual indicators such as secondary notation, as opposed to text-based interfaces, typed command labels or text navigation

In addition to computers, GUIs can be found in hand-held devices such as MP3 players, portable media players, gaming devices and smaller household,smartphones office and industry equipment.

Eg:Ticket booking, Inventory tool, Billing Machine, Windows OS

31 **What is flow graph notation and how it is important. April /May 2015**

A control flow graph (CFG) in computer science is a representation, Using graph notation, of all paths that might be traversed through aprogram during its execution.

32 **What is smoke testing ? APRIL /MAY 2017**

Smoke Testing, also known as "Build Verification **Testing**", is a type of software **testing** that comprises of a non-exhaustive set of **tests** that aim at ensuring that the most important functions work. The results of this **testing** is used to decide if a build is stable enough to proceed with
further **testing**.

33 **List testing strategies that address verification. Which types of testing address validation ? APRIL/MAY 2017**

Verification involves all the static testing techniques. Examples includes reviews, inspection and walkthrough
Validation includes all the dynamic testing techniques. Example includes all types of testing like smoke, regression, functional, systems and UAT

33 **What are the types of static testing tools?**

There are three types of static testing tools.

- **Code based testing tools** : These tools take source code as input and generate test cases.
- **Specialized testing tools :** Using this language the detailed test specification can be written for each test case.
- **Requirement-based testing tools**: These tools help in designing the as per user requirements.

34 **What is done in test design step?**

The details of the layout, tooling and standards required for test development are designed in this stage.

35 **Distinguish between verification and validation**?

Verification refers to the set of activities that ensure that software correctly implements a specific function. Validation refers to a different set of activities that ensure that the software that has been built is

traceable to the customer requirements.

36 **Write about drivers and stubs?**

Drivers and stub software need to be developed to test incompatible software. The “driver” is a program that accepts the test data and prints the relevant results. The “stub” is a subprogram that uses the module

interfaces and performs the minimal data manipulation if required.

37 **Define debugging**.

Debugging is defined as the process of removal of defect. It occurs as a consequence of successful testing.

38 **Define the terms:**

a) **Graph Matrices.**

b) **Connection Matrices.**

Graph Matrices:

 To develop software tool the data structure used is graph Matrix.

 Square Matrix

 Size equals number of nodes on the Flow graph

Connection Matrices:

 It Link Weight = 1= > Connection Exists

 It Link Weight=1=>Connection Does not Exists.

39

What errors are commonly found during Unit Testing?

Errors commonly found during Unit Testing are:

- Misunderstood or incorrect arithmetic precedence
- Mixed Mode Operations
- Incorrect Initializations
- Precision Accuracy
- Incorrect Symbolic representation of expression.

40

What problems may be encountered when Top-Down Integration is chosen?

Following problems may be encountered when Top Down Integration is chosen:

- Develop stubs that perform limited functions that simulate the actual module.

Integrate the software from the bottom of the hierarchy upward

41

What are the Steps in Bottom-Up Integration?

Steps in Bottom-Up Integration are:

- Low level components are combined into clusters perform specific software sub function.
- Driver is written to coordinate test case input and output.
- Cluster is tested.

42 **What is Flow Graph Notation?**

Flow Graph Notation means Simple notation for representing Control Flow. It is drawn only when Logical Structure of component is complex.

43 **What is acceptance testing**

Acceptance testing : This type of testing involves testing of the system with customer data if the system behaves as per customer need then it is accepted.

44 **What are the various testing strategies for conventional software?**

The various testing strategies are:

(i) Unit testing (ii) Integration testing

(iii) Validation testing (iv) System testing.

45 **List some of the testing done during SDLC.**

White box testing, black box testing, integration testing, system testing, installation testing. Regression testing, Acceptance testing.

46 **What is functionality testing?**

It is a black box testing which exercises the basic functionality of the product from an external; perspective.

47 What are the steps carried out in installation testing?

Ans. The steps carried out in installation testing are:

• Packaging • Documenting

• Installing • Verifying

48 **What are the objective of Formal Technical Reviews. Ans.** The Objective of Formal Technical Reviews are:

- Uncover errors in function, logic and implementation for representation of software.
- Software represented according to predefined standard.
- Verify software under review meets requirements
- Achieve software developed in Uniform Manner.
- Make projects more manageable.

49 **Explain Integrated testing team model?**

Ans. There in one project manage who manages both the development and the testing functions

50 **What are the common approaches in debugging? Ans.** The common approaches tin debugging are:

- **Brute force method:** The memory dumps and run- time tracks are examined and program with write statements in loaded to obtain clues to error causes.
- **Back tracking method:** The source code is examined by looking

backwards from symptom to potential causes or errors.

- **Causes eliminations method:** This method uses binary partitioning to reduce the number of location where errors can exists.

PART –B

S.NO	QUESTIONS
1	**What is black box & white-box testing? Explain how basis path testing helps to derive test cases to test every statement of a program.NOV/DEC-12, APRIL/MAY 2015, NOV/DEC 2017, APRIL/MAY 2017**

Criteria	Black Box Testing	White Box Testing
Definition	Black Box Testing is a software testing method in which the internal structure/ design/ implementation of the item being tested is NOT known to the tester	White Box Testing is a software testing method in which the internal structure/ design/ implementation of the item being tested is known to the tester.
Levels Applicable To	Mainly applicable to higher levels of testing:Acceptance Testing System Testing	Mainly applicable to lower levels of testing:Unit Testing Integration Testing
Responsibility	Generally, independent Software Testers	Generally, Software Developers
Programming Knowledge	Not Required	Required
Implementation Knowledge	Not Required	Required
Basis for Test Cases	Requirement Specifications	Detail Design

2 **Define: Regression testing. Distinguish: top-down and bottom-up integration. How is testing different from debugging? JustifyNOV/DEC-10, APRIL/MAY 2018**

Regression testing (rarely *non-regression testing*[1]) is re-running functional and non-functional tests to ensure that previously developed and tested software still performs after a change.[2] If not, that would be called a *regression*. Changes that may require regression testing include bug fixes, software enhancements, configuration changes, and even substitution of electronic components.[3] As regression test suites tend to grow with each found defect, test automation is frequently involved. Sometimes a change impact analysis is performed to determine an appropriate subset of test

BASIS FOR COMPARISON	TOP-DOWN INTEGRATION TESTING	BOTTOM-UP INTEGRATION TESTING
Basic	Uses stubs as the momentary replacements for the invoked modules and simulates the behaviour of the separated lower-level modules.	Use test drivers to initiate and pass the required data to the lower-level of modules.
Beneficial	If the significant defect occurs toward the top of the program.	If the crucial flaws encounters towards the bottom of the program.
Approach	Main function is written at first then the subroutines are called from it.	Modules are created first then are integrated with the main function.
Implemented on	Structure/procedure-oriented programming languages.	Object-oriented programming languages.

Risk analysation	Collaborating the impact of internal operational failures.	Models are used to analyze the individual process.
Complexity	Simple	Complex and highly data intensive.
Works on	Big to small components.	Small to big components.

Testing and ***Debugging*** are significant activities during software development and maintenance. ***Testing*** aims at finding a problem while ***Debugging*** aims at solving the problem. Only after the testing team reports the ***defect***, ***debugging*** can take place. With debugging, the developer identifies the problem in the system/application/code. Once the developer has fixed the bug, tester re-tests to ensure that the error/bug no longer exists. The figure given below demonstrates the fact very well

3 Write a note on equivalence partitioning & boundary value analysis of black box testingAPR/MAY-16 , NOV/DEC-15

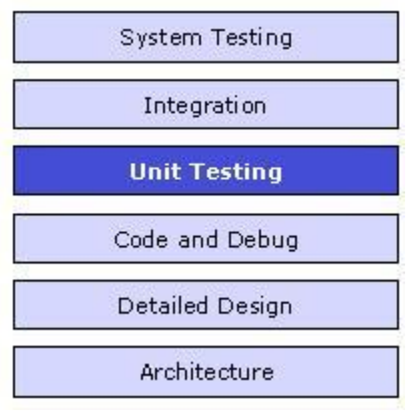
Equivalence partitioning (EP) is a specification-based or black-box technique. It can be applied at any level of testing and is often a good technique to use first.

- The idea behind this technique is to divide (i.e. to partition) a set of test conditions into groups or sets that can be considered the same (i.e. the system should handle them equivalently), hence 'equivalence partitioning'. **Equivalence partitions** are also known as equivalence classes – the two terms mean exactly the same thing.
- In equivalence-partitioning technique we need to test only one condition from each partition. This is because we are assuming that all the conditions in one partition will be treated in the same way by the software. If one condition in a partition works, we assume all of the conditions in that partition will work, and so there is little point in testing any of these others. Similarly, if one of the conditions in a partition does not work, then we assume that none of the conditions in that partition will work so again there is little point in testing any more in that partition.

4 What is unit testing? Why is it important? Explain the unit test consideration and test procedure.APR/MAY- 11,MAY/JUN-13 NOV/DEC2015

A **unit test** is the smallest testable part of an application like functions, classes, procedures, interfaces. Unit testing is a method by which individual units of source code are tested to determine if they are fit for use.

- **Unit tests are basically written and executed by software developers** to make sure that code meets its design and requirements and behaves as expected.
- The goal of unit testing is to segregate each part of the program and test that the individual parts are working correctly.
- This means that for any function or procedure when a set of inputs are given then it should return the proper values. It should handle the failures gracefully during the course of execution when any invalid input is given.
- A unit test provides a written contract that the piece of code must assure. Hence it has several benefits.
- Unit testing is basically done before integration as shown in the image below.



Method Used for unit testing: White Box Testing method is used for executing the unit test.

When Unit testing should be done?

Unit testing should be done before Integration testing.

By whom unit testing should be done?

Unit testing should be done by the developers.

Advantages of Unit testing:

1. Issues are found at early stage. Since unit testing are carried out by developers where they test their individual code before the integration. Hence the issues can be found very early and can be resolved then and there without impacting the other piece of codes.

2. Unit testing helps in maintaining and changing the code. This is possible by making the codes less interdependent so that unit testing can be executed. Hence chances of impact of changes to any other code gets reduced.

3. Since the bugs are found early in unit testing hence it also helps in reducing the cost of bug fixes. Just imagine the cost of bug found during the later stages of development like during system testing or during acceptance testing.

4. Unit testing helps in simplifying the debugging process. If suppose a test fails then only latest

changes made in code needs to be debugged.

5 Explain Integration & debugging activities? MAY/JUN-15

Integration testing is the process of testing the interface between two software units or module. It's focus on determining the correctness of the interface. The purpose of the integration testing is to expose faults in the interaction between integrated units. Once all the modules have been unit tested, integration testing is performed.

Integration test approaches –
There are four types of integration testing approaches. Those approaches are the following:
1. Big-Bang Integration Testing –
It is the simplest integration testing approach, where all the modules are combining and verifying the functionality after the completion of individual module testing. In simple words, all the modules of the system are simply put together and tested. This approach is practicable only for very small systems. If once an error is found during the integration testing, it is very difficult to localize the error as the error may potentially belong to any of the modules being integrated. So, debugging errors reported during big bang integration testing are very expensive to fix.

Advantages
- It is convenient for small systems.

Disadvantages:
- There will be quite a lot of delay because you would have to wait for all the modules to be integrated.
- High risk critical modules are not isolated and tested on priority since all modules are tested at once.

2. Bottom-Up Integration Testing –
In bottom-up testing, each module at lower levels is tested with higher modules until all modules are tested. The primary purpose of this integration testing is, each subsystem is to test the interfaces among various modules making up the subsystem. This integration testing uses test drivers to drive and pass appropriate data to the lower level modules.
Advantages:
- In bottom-up testing, no stubs are required.
- A principle advantage of this integration testing is that several disjoint subsystems can be tested simultaneously.

Disadvantages:
- Driver modules must be produced.
- In this testing, the complexity that occurs when the system is made up of a large number of small subsystem.

3. Top-Down Integration Testing –
Top-down integration testing technique used in order to simulate the behaviour of the lower-level modules that are not yet integrated.In this integration testing, testing takes place from top to bottom. First high-level modules are tested and then low-level modules and finally integrating the low-level modules to a high level to ensure the system is working as intended.
Advantages:
- Separately debugged module.
- Few or no drivers needed.

- It is more stable and accurate at the aggregate level.

Disadvantages:

- Needs many Stubs.
- Modules at lower level are tested inadequately.

4. Mixed Integration Testing –

A mixed integration testing is also called sandwiched integration testing. A mixed integration testing follows a combination of top down and bottom-up testing approaches. In top-down approach, testing can start only after the top-level module have been coded and unit tested. In bottom-up approach, testing can start only after the bottom level modules are ready. This sandwich or mixed approach overcomes this shortcoming of the top-down and bottom-up approaches. A mixed integration testing is also called sandwiched integration testing.

Advantages:

- Mixed approach is useful for very large projects having several sub projects.
- This Sandwich approach overcomes this shortcoming of the top-down and bottom-up approaches.

Disadvantages:

- For mixed integration testing, require very high cost because one part has Top-down approach while another part has bottom-up approach.
- This integration testing cannot be used for smaller system with huge interdependence between different modules.

DEBUGGING

It is a systematic process of spotting and fixing the number of bugs, or defects, in a piece of software so that the software is behaving as expected. Debugging is harder for complex systems in particular when various subsystems are tightly coupled as changes in one system or interface may cause bugs to emerge in another.

Debugging is a developer activity and effective debugging is very important before testing begins to increase the quality of the system. Debugging will not give confidence that the system meets its requirements completely but testing gives confidence.

6 Explain software testing types?<u>APR/MAY-16, NOV/DEC 2015</u>

Press-Pg-no- 384

7 Write elaborately on unit testing and regression testing. How do you develop test suites.<u>APRIL/MAY-15, APRIL/MAY 2018</u>

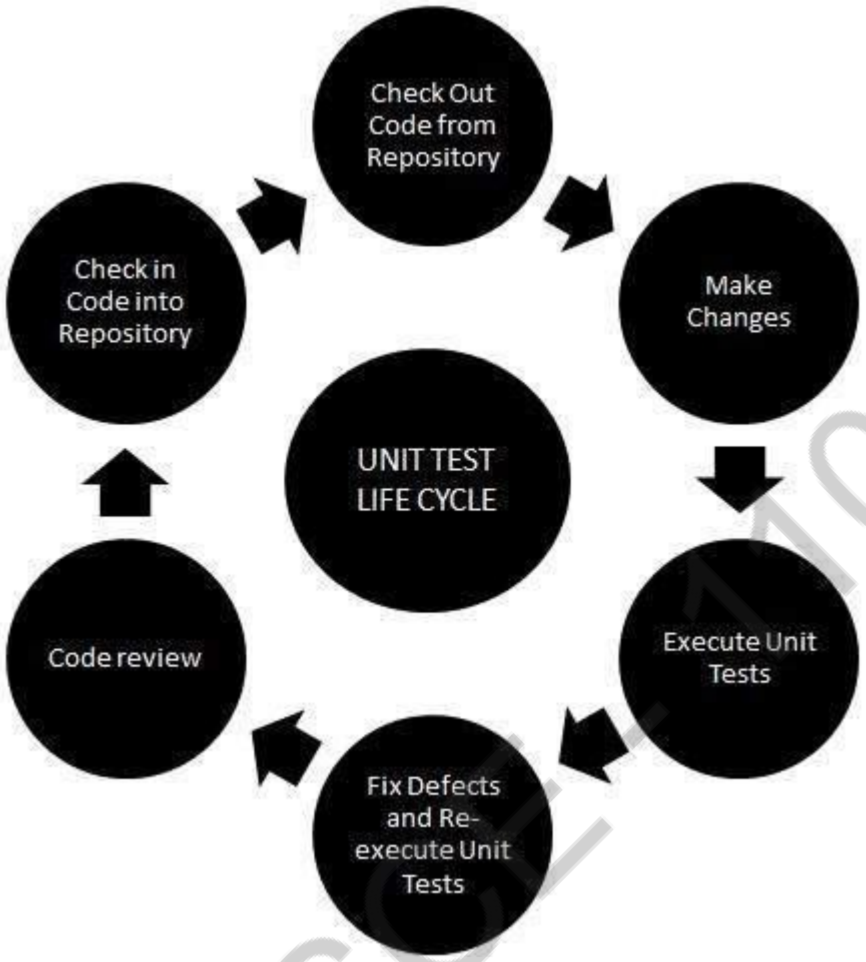
Unit testing, a testing technique using which individual modules are tested to determine if there are any issues by the developer himself. It is concerned with functional correctness of the standalone modules.

The main aim is to isolate each unit of the system to identify, analyze and fix the defects.

Unit Testing - Advantages:

- Reduces Defects in the Newly developed features or reduces bugs when changing the existing functionality.
- Reduces Cost of Testing as defects are captured in very early phase.
- Improves design and allows better refactoring of code.
- Unit Tests, when integrated with build gives the quality of the build as well.

Unit Testing LifeCyle:


Check Out Code from Repository
Make Changes
Execute Unit Tests
Fix Defects and Re-execute Unit Tests
Code review
Check in Code into Repository
UNIT TEST LIFE CYCLE

Unit Testing Techniques:

- **Black Box Testing -** Using which the user interface, input and output are tested.
- **White Box Testing -** used to test each one of those functions behaviour is tested.
- **Gray Box Testing -** Used to execute tests, risks and assessment methods.

Regression testing a black box testing technique that consists of re-executing those tests that are impacted by the code changes. These tests should be executed as often as possible throughout the software development life cycle.

Types of Regression Tests:

- **Final Regression Tests: -** A "final regression testing" is performed to validate the build that hasn't changed for a period of time. This build is deployed or shipped to customers.
- **Regression Tests: -** A normal regression testing is performed to verify if the build has NOT broken any other parts of the application by the recent code changes for defect fixing or for enhancement.

Selecting Regression Tests:

- Requires knowledge about the system and how it affects by the existing functionalities.

- Tests are selected based on the area of frequent defects.
- Tests are selected to include the area, which has undergone code changes many a times.
- Tests are selected based on the criticality of the features.

Regression Testing Steps:

Regression tests are the ideal cases of automation which results in better **R**eturn **O**n **I**nvestment (ROI).

- Select the Tests for Regression.
- Choose the apt tool and automate the Regression Tests
- Verify applications with Checkpoints
- Manage Regression Tests/update when required
- Schedule the tests
- Integrate with the builds
- Analyze the results

8 i.What is cyclomatic complexity? How to compute cyclomatic complexity

<u>APRIL/MAY-15, NOV/DEC 2017</u>

Cyclomatic complexity is a source code complexity measurement that is being correlated to a number of coding errors. It is calculated by developing a Control Flow Graph of the code that measures the number of linearly-independent paths through a program module.

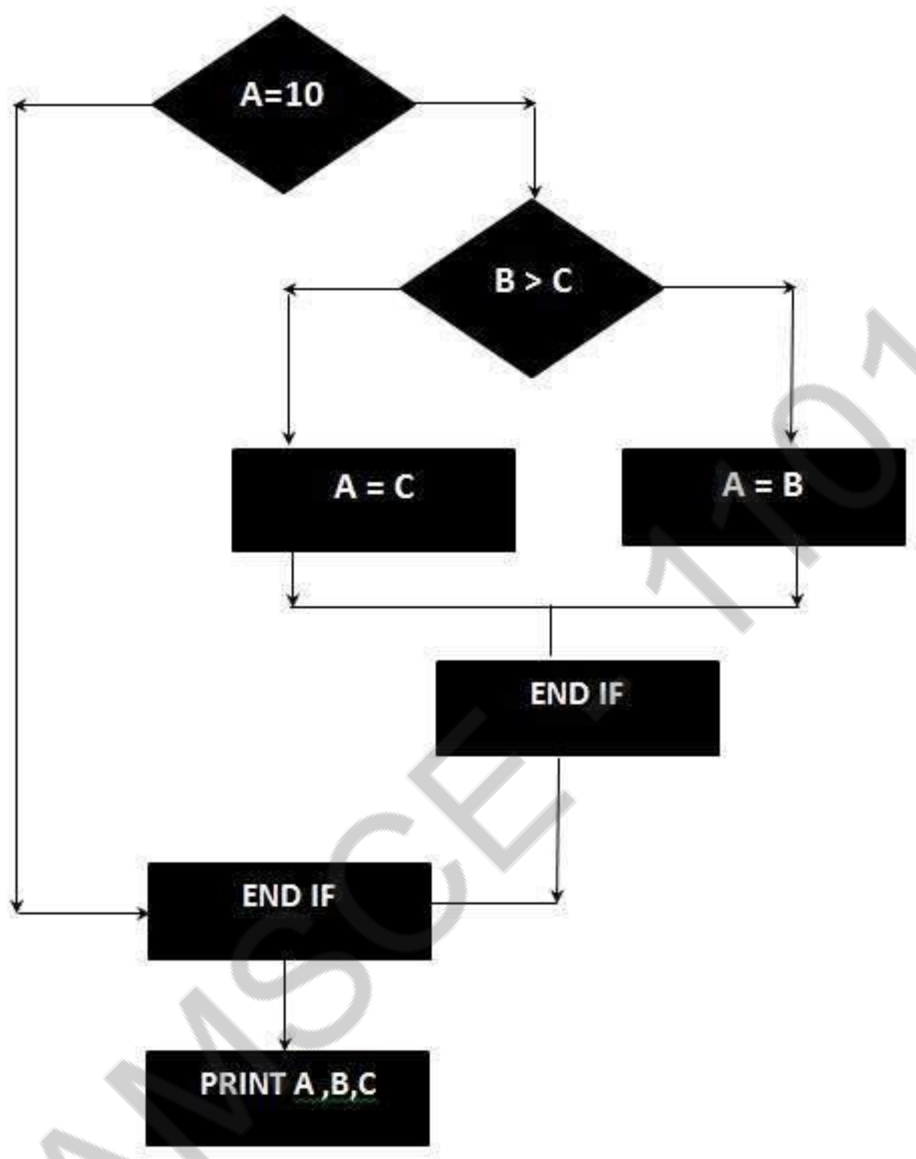
Lower the Program's cyclomatic complexity, lower the risk to modify and easier to understand. It can be represented using the below formula:

```
Cyclomatic complexity = E - N + 2*P
where,
  E = number of edges in the flow graph.
  N = number of nodes in the flow graph.
  P = number of nodes that have exit points
```

Example :

```
IF A = 10 THEN
 IF B > C THEN
   A = B
 ELSE
   A = C
 ENDIF
ENDIF
Print A
Print B
Print C
```

FlowGraph:



The Cyclomatic complexity is calculated using the above control flow diagram that shows seven nodes(shapes) and eight edges (lines), hence the cyclomatic complexity is 8 - 7 + 2 = 3

9 Explain integration testing in detail.MAY/JUN-13, APRIL/MAY 2017, APRIL/MAY 2018

Upon completion of unit testing, the units or modules are to be integrated which gives raise to integration testing. The purpose of integration testing is to verify the functional, performance, and reliability between the modules that are integrated.

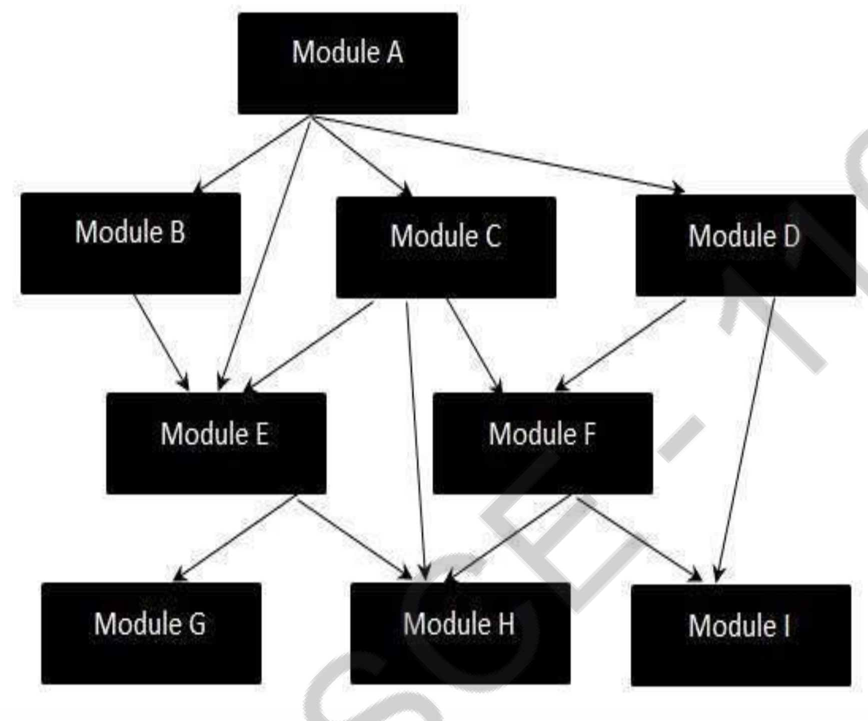
Integration Strategies:

- **Big-Bang Integration**

Big Bang Integration Testing is an integration testing strategy wherein all units are linked at once, resulting in a complete system. When this type of testing strategy is adopted, it is difficult to isolate any errors found, because attention is not paid to verifying the interfaces across individual units.

Big Bang Integration - WorkFlow Diagram

Big Bang Testing is represented by the following workflow diagram:



Disadvantages of Big-Bang Testing

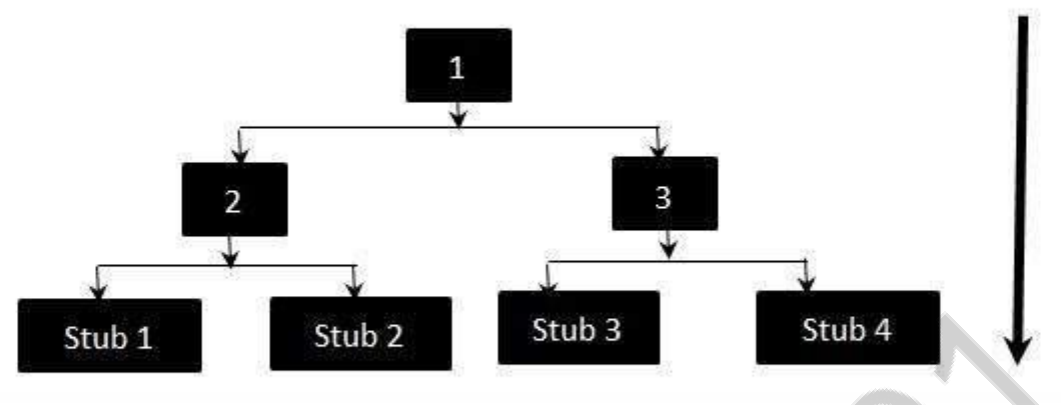
- Defects present at the interfaces of components are identified at very late stage as all components are integrated in one shot.
- It is very difficult to isolate the defects found.
- There is high probability of missing some critical defects, which might pop up in the production environment.
- It is very difficult to cover all the cases for integration testing without missing even a single scenario.

Top Down Integration

Top-down integration testing is an integration testing technique used in order to simulate the behaviour of the lower-level modules that are not yet integrated. Stubs are the modules that act as temporary replacement for a called module and give the same output as that of the actual product.

The replacement for the 'called' modules is known as 'Stubs' and is also used when the software needs to interact with an external system.

Stub - Flow Diagram:



The above diagrams clearly states that Modules 1, 2 and 3 are available for integration, whereas, below modules are still under development that cannot be integrated at this point of time. Hence, Stubs are used to test the modules. The order of Integration will be:

1,2
1,3
2,Stub 1
2,Stub 2
3,Stub 3
3,Stub 4

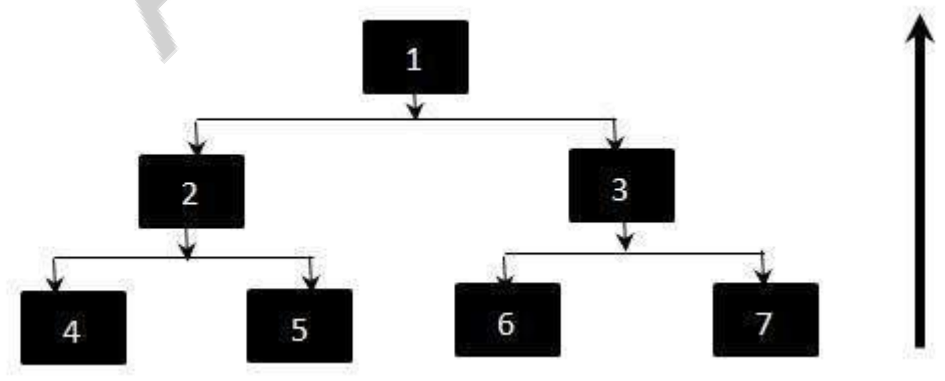
Testing Approach:

+ Firstly, the integration between the modules 1,2 and 3
+ Test the integration between the module 2 and stub 1,stub 2
+ Test the integration between the module 3 and stub 3,stub 4

Bottom Up Integration

Each component at lower hierarchy is tested individually and then the components that rely upon these components are tested.

Bottom Up Integration - Flow Diagram



The order of Integration by Bottom-down approach will be:

4,2
5,2
6,3
7,3
2,1
3,1

Testing Approach :

+ Firstly, Test 4,5,6,7 individually using drivers.
+ Test 2 such that it calls 4 and 5 separately. If an error occurs we know that the problem is in one of the modules.
+ Test 1 such that it calls 3 and If an error occurs we know that the problem is in 3 or in

Hybrid Integration

Integration Testing is a phase in software testing in which standalone modules are combined and tested as a single entity. During that phase, the interface and the communication between each one of those modules are tested. There are two popular approaches for Integration testing which is Top down Integration Testing and Bottom up Integration Testing.

In Hybrid Integration Testing, we exploit the advantages of Top-down and Bottom-up approaches. As the name suggests, we make use of both the Integration techniques.

Top down Integration
Bottom Up Integration

Hybrid Integration Testing - Features

- It is viewed as three layers; viz - The Main Target Layer, a layer above the target layer and a layer below the target layer.
- Testing is mainly focussed for the middle level target layer and is selected on the basis of system characteristics and the structure of the code.
- Hybrid Integration testing can be adopted if the customer wants to work on a working version of the application as soon as possible aimed at producing a basic working system in the earlier stages of the development cycle.

10 What is black box testing? Explain the different types of black box testing strategies with example?<u>NOV/DEC 2016</u>

Black-box testing is a method of software testing that examines the functionality of an application based on the specifications. It is also known as Specifications based testing. Independent Testing Team usually performs this type of testing during the software testing life cycle.

This method of test can be applied to each and every level of software testing such as unit, integration, system and acceptance testing.

Behavioural Testing Techniques:

There are different techniques involved in Black Box testing.

- **Equivalence Class**

Equivalence Partitioning also called as equivalence class partitioning. It is abbreviated as ECP. It is a software testing technique that divides the input test data of the application under test into each partition at least once of equivalent data from which test cases can be derived.

An advantage of this approach is it reduces the time required for performing testing of a software due to less number of test cases.

Example:

The Below example best describes the equivalence class Partitioning:

```
Assume that the application accepts an integer in the range 100 to 999
Valid Equivalence Class partition: 100 to 999 inclusive.
Non-valid Equivalence Class partitions: less than 100, more than 999, decimal numbers and
alphabets/non-numeric characters.
```

- **Boundary Value Analysis**

Boundary value analysis is a type of black box or specification based testing technique in which tests are performed using the boundary values.

Example:

An exam has a pass boundary at 50 percent, merit at 75 percent and distinction at 85 percent. The Valid Boundary values for this scenario will be as follows:

```
49, 50 - for pass
74, 75 - for merit
84, 85 - for distinction
```

Boundary values are validated against both the valid boundaries and invalid boundaries.

The Invalid Boundary Cases for the above example can be given as follows:

```
0 - for lower limit boundary value
101 - for upper limit boundary value
```

- **Domain Tests**

Domain testing is a software testing technique in which selecting a small number of test cases from a nearly infinite group of test cases. For testing few applications, Domain specific knowledge plays a very crucial role.

Domain testing is a type of functional testing and tests the application by feeding interesting inputs and evaluating its outputs.

Domain - Equivalence Class Testing

Equivalence class carries its own significance when performing domain testing. Different ways of equivalence class are:

- Intuitive equivalence
- Specified equivalence
- Subjective equivalence
- Risk-based equivalence:

- **Orthogonal Arrays**

Orthogonal array testing is a systematic and statistical way of a black box testing technique used when number of inputs to the application under test is small but too complex for an exhaustive testing.

Orthogonal Array Testing Characteristics:

- OAT, is a systematic and statistical approach to pairwise interactions.
- Executing a well-defined and a precise test is likely to uncover most of the defects.
- 100% Orthogonal Array Testing implies 100% pairwise testing.

Example:

If we have 3 parameters, each can have 3 values then the possible Number of tests using conventional method is 3^3 = 27
While the same using OAT, it boils down to 9 `test cases.`

- **Decision Tables**

- State Models
- Exploratory Testing
- All-pairs testing

11 1. **(a) Consider the pseudo code for simple subtraction given below:**

<u>NOV/DEC 2016, APRIL/MAY 2018</u>

```
(1) program 'simple subtraction'
(2) input (x,y)
(3) output (x)
(4) output (y)
(5) if x> y then DO
(6) x-y = z
(7) else y –x = z
(8) endif
(9) output (z)
(10) output "end program"
```

Perform basis path testing and generate test cases.

(b) What is refactoring? When is it needed? Explain with ex?

Refer class notes.

12 **Explain in detail about system testing**

System Testing is the testing of a complete and fully integrated software product. Usually, software is only one element of a larger computer-based system. Ultimately, software is interfaced with other software/hardware systems. System Testing is actually a series of different tests whose sole purpose is to exercise the full computer-based system.

Two Category of Software Testing

- Black Box Testing
- White Box Testing

System test falls under the **black box testing** category of software testing.

White box testing is the testing of the internal workings or code of a software application. In contrast, black box or System Testing is the opposite. System test involves the external workings of the software from the user's perspective.

13 **Explain about the software testing strategies**

Software testing is an investigation conducted to provide stakeholders with information about the quality of the software product or service under test.[1] Software testing can also provide an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation. Test techniques include the process of executing a program or application with the intent of finding software bugs (errors or other defects), and verifying that the software product is fit for use.

Software testing involves the execution of a software component or system component to evaluate one or more properties of interest. In general, these properties indicate the extent to which the component or system under test:

- meets the requirements that guided its design and development,
- responds correctly to all kinds of inputs,
- performs its functions within an acceptable time,
- it is sufficiently usable,
- can be installed and run in its intended environments, and
- achieves the general result its stakeholders desire.

As the number of possible tests for even simple software components is practically infinite, all software testing uses some strategy to select tests that are feasible for the available time and resources. As a result, software testing typically (but not exclusively) attempts to execute a program or application with the intent of finding software bugs (errors or other defects). The job of testing is an iterative process as when one bug is fixed, it can illuminate other, deeper bugs, or can even create new ones.

Software testing can provide objective, independent information about the quality of software and risk of its failure to users or sponsors.[1]

Software testing can be conducted as soon as executable software (even if partially complete) exists. The overall approach to software development often determines when and how testing is conducted. For example, in a phased process, most testing occurs after system requirements have been defined and then implemented in testable programs. In contrast, under an agile approach, requirements, programming, and testing are often done concurrently.

14 Discuss in detail about test strategies for conventional software(May/June 2011)

Refer class notes

15 Explain in detail about basic path testing.(May/Jun 2014)

Path Testing is a structural testing method based on the source code or algorithm and NOT based on the specifications. It can be applied at different levels of granularity.

Path Testing Assumptions:

- The Specifications are Accurate
- The Data is defined and accessed properly
- There are no defects that exist in the system other than those that affect control flow

Path Testing Techniques:

- **Control Flow Graph (CFG) -** The Program is converted into Flow graphs by representing the code into nodes, regions and edges.
- **Decision to Decision path (D-D) -** The CFG can be broken into various Decision to Decision paths and then collapsed into individual nodes.

- **Independent (basis) paths -** Independent path is a path through a DD-path graph which cannot be reproduced from other paths by other methods.

UNIT – 5

PART –A

S.NO	QUESTIONS
1	**What are the processes of risk management? NOV/DEC-10, NOV/DEC- 12, NOV/DEC 2013,NOV/DEC2015** Risk identification Risk projection (estimation) Risk mitigation, monitoring, and management
2	**State the need for software configuration review. NOV/DEC-11** The intent of the review is to ensure that all elements of the software configuration have been properly developed, cataloged & have necessary detail to bolster the supportpfase of the software lifecycle.

3 **List any five CASE tools classified by function in the taxonomy of CASE tools** **NOV/DEC-11**

1. project planning tools
2. metrics & management tools
3. prototyping tools
4. Re- engineering tools
5. documentation tools.

4 **Define error, fault and failure.** **NOV/DEC-10**

Error – it is a state that can lead to a system behavior that is unexpected by the

System user.

Fault- it is a characteristic of a software system that can lead to system error.

Failure – it is an event that occurs at some point in time when the system does not

Deliver a service as per user's expectation.

5 **What is project planning?** **APR/MAY-12, APR/MAY-15**

The various types of plan is developed to support main software project plan which is concerned with schedule & budget. Types of project plan

Quality plan, Validation plan, Configuration mgmt plan, Maintenance plan, Staff development plan.

6 **List the various types of software errors?** **APR/MAY-11, NOV/DEC-12**

Reports detailing bugs in a program are commonly known as bug reports, defect reports, fault reports, problem reports, trouble reports, change requests.

7 **Differentiate between size oriented and function oriented metrics? <u>MAY/JUN-13 MAY/JUNE 2016,NOV/DEC 2015</u>**

Size oriented metrics – it considers the size of the software that has been produced. The software organization maintains simple records in tabular form. Table entries are LOC, effort, defects, and project name. Function oriented metrics – it measures the functionality delivered by software. Function point based on software information domain and
Complexity.

8 **Define measure.(APRIL/MAY-2008)**

Measure is defined as a quantitative indication of the extent, amount, dimension, or size of some attribute of a product or process.

9 **How is productivity and cost related to function points? <u>NOV/DEC2016</u>**

Software Productivity = Function Points / Inputs (persons/mnth) Cost = $ / Function Points (FP)

10 **What are the types of metrics? <u>MAY/JUNE 2016</u>**

Direct metrics – It refers to immediately measurable attributes. Example – Lines of code,execution speed.

Indirect metrics – It refers to the aspects that are not immediately quantifiable or measurable.

Example – functionality of a program.

11 What are the advantages and disadvantages of size measure? Advantages:

- Artifact of software development which is easily counted.
- Many existing methods use LOC as a key input.
- A large body of literature and data based on LOC already exists

Disadvantages:

This method is dependent upon the programming language.

- This method is well designed but shorter program may get suffered.
- It does not accommodate non procedural languages.
- In early stage of development it is difficult to estimate LOC.

12 Write short note on the various estimation techniques.

Algorithmic cost modeling – the cost estimation is based on the size of the software.

Expert judgement – The experts from software development and the application domain use their exoerience to predict software costs.

Estimation by analogy – The cost of a project is computed by comparing the project to a similar project in the same application domain and then cost can be computed.

Parkinson's law – The cost is determined by available resources rather than by objective assessment.

Pricing to win – The project costs whatever the customer ready to spend it.

13 What is COCOMO model?

COnstructiveCOstMOdel is a cost model, which gives the estimate of number of man- months it will take to develop the software product.

14 **Give the procedure of the Delphi method.**

1. The co-ordinator presents a specification and estimation form to each expert.
2. Co-ordinator calls a group meeting in which the experts discuss estimation issues with the coordinator and each other.
3. Experts fill out forms anonymously.
4. Co-ordinator prepares and distributes a summary of the estimates.
5. The Co-ordinator then calls a group meeting.In this meeting the experts mainly discuss the points where their estimates vary widely.
6. The experts again fill out forms anonymously.
7. Again co-ordinator edits and summarizes the forms,repeating steps5 and 6 until the co-ordinator is satisfied with the overallprediction synthesized from experts.

15 **What are the metrics computed during error tracking activity?**

Errors per requirement specification page. Errors per component-design level

Errors per component-code level DRE-requirement analysis

DRE-architectural analysis

DRE-component level design

DRE-coding.

16 **What is risk management? <u>NOV/DEC2016</u>**

Risk management is the identification, assessment, and prioritization of risks followed by coordinated and economical application of resources to minimize, monitor, and control the probability and/or impact of unfortunate eventsor to maximize the realization of opportunities. Risk management's objective is to assure uncertainty does not deflect the endeavor from the business goals.

17 **What is software maintenance?**

Software maintenance is an activity in which program is modified after it has been put into use.

18 **Will exhaustive testing guarantee that the program is 100% correct? <u>APR/MAY 2016</u>**

No, even exhaustive testing will not guarantee that the program is 100 percent correct. There are too many variables to consider.

19 **What are the types of software maintenance?**

- Corrective maintenance – Means the maintenance for correcting the software faults.
- Adaptive maintenance – Means maintenance for adapting the change in environment.
- Perfective maintenance – Means modifying or enhancing the system to meet the new requirements.
- Preventive maintenance – Means changes made to improve future maintainability

20 **How the CASE tools are classified?**

CASE tools can be classified by

a. By function or use

b. By user type(e.g. manager,tester),or

c. By stage in software engineering process (e.g.requirements,test).

21 **Dinguish between direct & indirect measures of metrics.**

Direct metrics is directly measurable attribute(lines of code execution speed,size of memory.

Indirect metrics: these are the aspects that are not immediately measurable.(functionality,reliabblity,maintainability)

22 **List down few process and product metrics. <u>MAY/JUNE 2016</u>**

1.size metrics-It is used for measuring the size of the software.(local based metric,FP based metric)

2.complexity metric- A software module can be described by a control flow graph.(cyclomatic complexity, McCabe complexity)

3.quality metric- (Defects,reliabilitymetric,Maintainability)

23 **Define software measure.**

It is a numeric value for a attribute of a software product or process.

Types:

1.Direct measure 2.indirect measure

24 **List out the different approaches to size of the software**.

1. LOC-computing the line of code
2. FP-computing function point of the program.

25 **An organic software occupies 15000 LOC.how many programmers are needed to complete?(NOV/DEC-12)**

System=organic Lines of

coding=15k LOC

$E=a_b(KLOC)b_b$

$=2.4(15)^{1.05}$

$=41$ persons per month $D=c_b(e)d_b$

$=2.5(41)^{0.38}$

$=10$ months $P=41/10$

$P=4$ persons.

4 persons are needed.

26 **What is error tracking?(APRIL/MAY-14)**

It is a process of finding out and correcting the errors that may occur during the software development process at various stages such as software design,coding or documenting.

27 **What are the types of static testing tools?**

There are three types of static testing tools.

- Code based testing tools – These tools take source code as input and generate test cases.
- Specialized testing tools – Using this language the detailed test specification can be written for each test case.
- Requirement-based testing tools – These tools help in designing the test cases as per user

requirements.

28 **What are the productivity measures and list its type. APRIL/MAY 2017**

Productivity is an overall measure of the ability to produce a good or service. More specifically, productivity is the measure of how specified resources are managed to accomplish timely objectives as stated in terms of quantity and quality. Productivity may also be defined as an index that measures output (goods and services) relative to the input (labor, materials, energy, etc., used to produce the output). there are two major ways to increase productivity: increase the numerator (output) or decrease the denominator (input).

29 **Define ZIPF's law.**

The probability of occurrence of words or other items starts high and tapers off. Thus, a few occur very often while many others occur rarely. Formal Definition: Pn ~ 1/na, where Pn is the frequency of occurrence of the nth ranked item and a is close to 1.

30 **List out the principles of project scheduling. NOV/DEC2017**

Software project scheduling is an activity that distributes estimated effort across the planed project duration by allocating the effort to specific software engineering tasks.

First, a macroscopic schedule is developed. a detailed schedule is redefined for each entry in the macroscopic schedule.

A schedule evolves over time.

Basic principles guide software project scheduling:

- Compartmentalization

- Interdependency

- Time allocation

- Effort allocation

- Effort validation

- Defined responsibilities

- Defined outcomes

- Defined milestones

31 Write a note on Risk information sheet. NOV/DEC 2017

A risk information sheet is a means of capturing information about a risk. Risk information sheets are used to document new risks as they are identified. They are also used to modify information as risks are managed. It is a form that can be submitted to the appropriate person or included in a database with other project risks. In the absence of a database, this becomes a primary means of
documenting and retaining information about a risk.

32 List two customer related and technology related risks. APRIL/MAY 2017

customer related risks

•Customer relationship management may be fragmented.
•New methods with which to improve customer service and reduce related costs are not utilized.
•Lack of knowledge on the part of one section of an enterprise regarding interactions with a customer on the part of another can lead to customer frustration and embarrassment.
•Inability to respond to market demands caused by lack of integration among order-entry systems or, even worse, due to infrastructure.
• Lack of visibility of the order status along the whole supply chain.

Technology related risk :

- Architecture risk
- Artificial intelligence risk
- Audit risk
- Availability

33 What is EVA ? APRIL/MAY 2018

Earned Value Analysis (**EVA**) is an industry standard method of measuring a project's progress at any given point in time, forecasting its completion date and final cost, and analyzing variances in the schedule
and budget as the project proceeds.

34 Identify The Types Of Maintenance for each of the following<u>APRIL/MAY</u> <u>2018</u>

Correcting the Software Faults . Adapting the

change in environment

There are four **types of maintenance**, namely, **corrective**, adaptive, perfective, and **preventive**. ...
Correctivemaintenance dealswiththe **repair** of **faults** or **defects** found in day-today system functions. ...
In the event of a system **failure** due to an error, actions are taken to restore the operation of the **software** system.

35

What is cost schedule?

Cost schedule shows the planned cumulative expenditure cost by the use of resource overtime

36

What is RMMM?

Ans. RMMM stands for Risk Mitigation, Monitoring and Management Plan. It is also called Risk Aversion.

37

What Is Risk mitigation?

Ans. Mitigation is a possible means if minimizing or even avoiding the Impact of risk.

38

What are the factors that lead to Risk? Ans. The

factors that lead to Risk are:

- Estimation errors.
- Planning assumptions.
- Business risks.

39

What are the test points?

Test points allow data to be inspected or modified at various points in the system

40

What is refactoring?

A small change to a database schema which improves its design

41

Explain the common risk tools and techniques.

Ans. There are at least six different ways of identifying the potential risks. These are:

• Examining organizational history

• Preparing checklists

• Information buying

• Framework based risk categorization

• Simulation

• Decision trees.

42 **What is called support risk?**

Ans. Support risk is the degree of uncertainty fiat the resultant software will be easy to correct, adapt and enhance

43 **What Is Risk?**

Ans. Risks are events that are usually beyond the planner’s control.

44 **What are the Dimensions of Risk quantification? Ans.**

Probability and the impact of Risk.

45 **What is meant by Delphi method?**

The Delphi technique is an estimation technique intended to active a common agreement for estimation efforts.

46 **What is meant by CASE tools?**

The computer aided software engineering tools automatic the project management activities, manage all the work products. The CASE tools assist to perform various activities such as analysis, design, coding and testing.

47 **What are the three phases of Risk management? Ans.** The three phases of risk management are:

Risk identification, Risk Quantification, and Risk mitigation.

48 **What are the factors that lead to Risk? Ans.** The factors that lead to Risk are:

- Estimation errors.
- Planning assumptions.
- Business risks.

49 **What is meant by software project scheduling?**

Software project scheduling is an activity that distributes estimated effort across the planned project duration by allocating the effort to specified software engineering tasks.

50 **What are the various steps under risk analysis? Ans.** The various steps under risk analysis are:

- Risk Estimation.
- Risk identification.
- Risk evaluation.

PART –B

S.NO **QUESTIONS**

1 **(f) Elaborate on the series of tasks of a software configuration management process.**

(g) Describe function point analysis with a neat example **NOV/DEC 2013**

Software configuration management, SCM is an activity which is used at every level and every part of the process of software Engineering. Every improvement takes the shape of better control. This is a discipline which controls betters and according to client need in software Engineering. With the help of this many types are changes which play an important role in software Engineering and development process.

In the simple way if we define the term configuration of management, this is the tool which makes better control, easy maintenance during the whole process of software development. With the help of software configuration management we can easily find out what modification and controlling required by the developer. SCM have the capacity to control all those effects which comes in software projects. The main objectives of SCM is increase the production by reduce the errors.

When a software development process start then SCM take change by identification, control, alteration, audit and etc. after that the output of total process provided to our customer. We can clarify the action of SCM as:

1. **Software configuration identification -** Normally software is used in various kinds of programs and documentation and data related to each program is called configuration identification. With the help of C.I we can make a guide line which will be helpful in software development process, several time the requirement of guideline for check the document and design of software. Document related to SCM are the useful item, with the help of this we can make better control and take a basic unit for configuration.

2. **Software configuration control -** This is the process of deciding with the help of this we make coordination between the changes which is necessary and apply them as per mentioned in guideline. Configuration control board gives the permission for any kind of change or modification which is necessary for the project. Many times CCB take advice of those members which are the part of software development process.

3. **Accounting status of Software configuration -** The process of maintaining record of all data which is necessary for the software is called accounting status of software. It has all the data related to the old software to new software that what changes are done or required for the fulfillment of the customer need.

4. **Auditing of software configuration -** Auditing of software configuration is may be defined as an art with the help of this we can understand that the required actions or changes are done by the developer or not. Some of the item involved in the process of verifying or auditing.

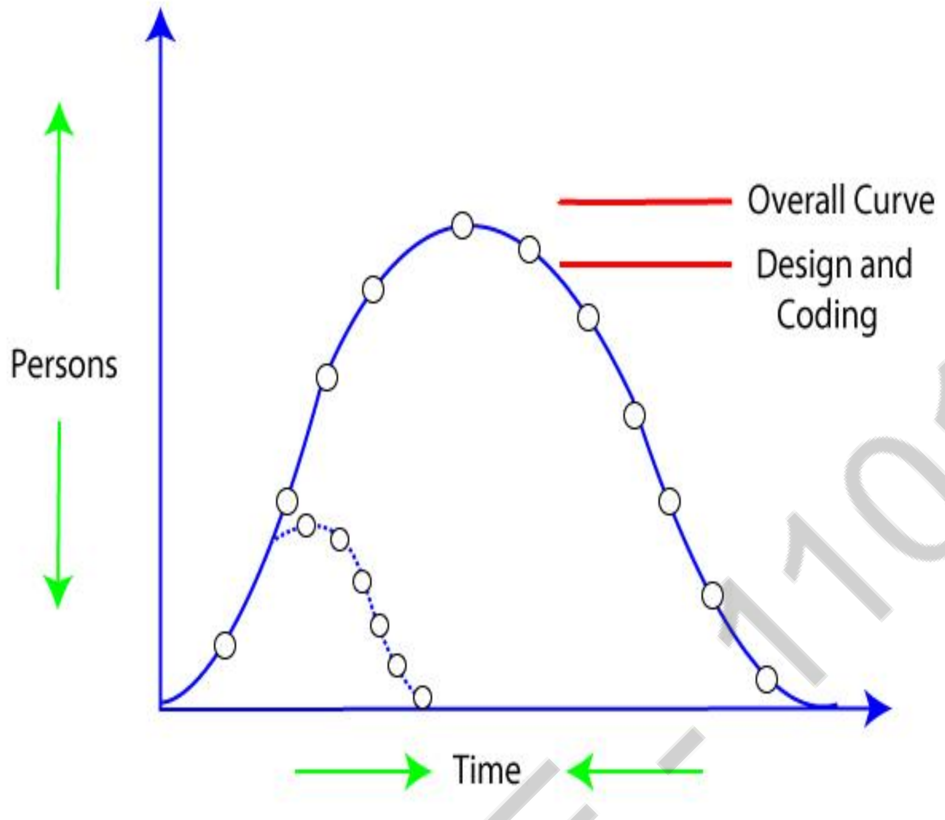
   - Function is properly performed by the software.
   - The process of documentation, data is completed or not.

Benefits

- With the help of SCM we can easily control all changes which are done in development process.
- It gives the surety to check that changes are done on required area.
- It is helpful to generate the new software with old components.
- SCM has the capacity to explain everything about the process of software development.

2 Explain make/buy decision & discuss Putnam resource allocation model & derive time & effort equation?APRIL/MAY2016

The Lawrence Putnam model describes the time and effort requires finishing a software project of a specified size. Putnam makes a use of a so-called The Norden/Rayleigh Curve to estimate project effort, schedule & defect rate as shown in fig:



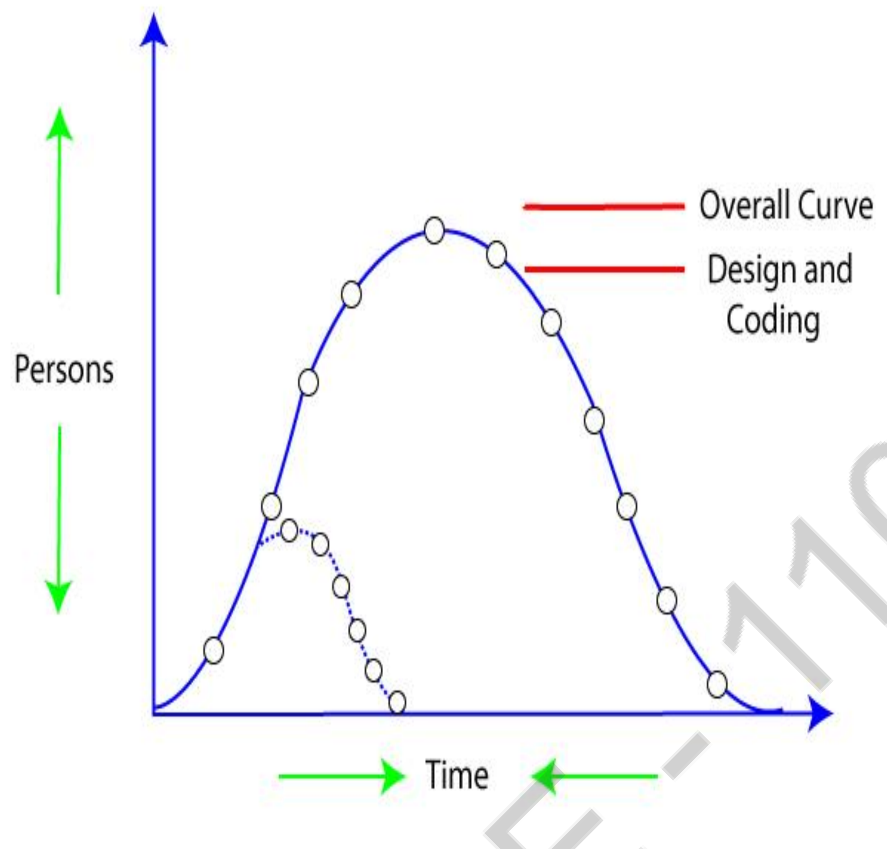
The Rayleigh manpower loading Curve

Putnam noticed that software staffing profiles followed the well known Rayleigh distribution. Putnam used his observation about productivity levels to derive the software equation:

$$\mathbf{L = C_k K^{1/3} t_d^{4/3}}$$

The various terms of this expression are as follows:

The Lawrence Putnam model describes the time and effort requires finishing a software project of a specified size. Putnam makes a use of a so-called The Norden/Rayleigh Curve to estimate project effort, schedule & defect rate as shown in fig:



The Rayleigh manpower loading Curve

Putnam noticed that software staffing profiles followed the well known Rayleigh distribution. Putnam used his observation about productivity levels to derive the software equation:

$$\mathbf{L = C_k K^{1/3} t_d^{4/3}}$$

The various terms of this expression are as follows:

K is the total effort expended (in PM) in product development, and L is the product estimate in **KLOC** .

$\mathbf{t_d}$ correlate to the time of system and integration testing. Therefore, $\mathbf{t_d}$ can be relatively considered as the time required for developing the product.

$\mathbf{C_k}$ Is the state of technology constant and reflects requirements that impede the development of the program.

Typical values of $\mathbf{C_k}$ = 2 for poor development environment

C_k= 8 for good software development environment

C_k = 11 for an excellent environment (in addition to following software engineering principles, automated tools and techniques are used).

The exact value of **C_k** for a specific task can be computed from the historical data of the organization developing it.

Putnam proposed that optimal staff develop on a project should follow the Rayleigh curve. Only a small number of engineers are required at the beginning of a plan to carry out planning and specification tasks. As the project progresses and more detailed work are necessary, the number of engineers reaches a peak. After implementation and unit testing, the number of project staff falls.

Effect of a Schedule change on Cost

Putnam derived the following expression:

$$L = C_k K^{1/3} t_d^{4/3}$$

Where, **K** is the total effort expended (in PM) in the product development

L is the product size in KLOC

t_d corresponds to the time of system and integration testing

C_k Is the state of technology constant and reflects constraints that impede the progress of the program

Now by using the above expression, it is obtained that,

$$K = L^3 / C_k^3 t_d^4$$

Or $$K = C / t_d^4$$

For the same product size, C =L^3 / C_k^3 is a constant.

Or $$\frac{K_1}{K_2} = t_{d2}^4 / t_{d1}^4$$

Or $$K \propto 1/t_d^4$$

Or, $$\text{cost} \propto 1/t_d$$

(As project development effort is equally proportional to project development cost)

From the above expression, it can be easily observed that when the schedule of a project is compressed, the required development effort as well as project development cost increases in proportion to the fourth power of the degree of compression. It means that a relatively small compression in delivery schedule can result in a substantial penalty of human effort as well as development cost.

For example, if the estimated development time is 1 year, then to develop the product in 6 months, the total effort required to develop the product (and hence the project cost) increases 16 times.

3 **Explain the various CASE tools for project management and how they are useful in achieving the objectives APRIL/MAY- 15**

Project management is one of the critical processes of any project. This is due to the fact that project management is the core process that connects all other project activities and processes together.

When it comes to the activities of project management, there are plenty. However, these plenty of project management activities can be categorized into five main processes.

Let's have a look at the five main project management processes in detail.

1 - Project Initiation

Project initiation is the starting point of any project. In this process, all the activities related to winning a project takes place. Usually, the main activity of this phase is the pre-sale.

During the pre-sale period, the service provider proves the eligibility and ability of completing the project to the client and eventually wins the business. Then, it is the detailed requirements gathering which comes next.

During the requirements gathering activity, all the client requirements are gathered and analysed for implementation. In this activity, negotiations may take place to change certain requirements or remove certain requirements altogether.

Usually, project initiation process ends with requirements sign-off.

2 - Project Planning

Project planning is one of the main project management processes. If the project management team gets this step wrong, there could be heavy negative consequences during the next phases of the project.

Therefore, the project management team will have to pay detailed attention to this process of the project.

In this process, the project plan is derived in order to address the project requirements such as, requirements scope, budget and timelines. Once the project plan is derived, then the project schedule is developed.

Depending on the budget and the schedule, the resources are then allocated to the project. This phase is the most important phase when it comes to project cost and effort.

3 - Project Execution

After all paperwork is done, in this phase, the project management executes the project in order to achieve project objectives.

When it comes to execution, each member of the team carries out their own assignments within the given deadline for each activity. The detailed project schedule will be used for tracking the project progress.

During the project execution, there are many reporting activities to be done. The senior management of the company will require daily or weekly status updates on the project progress.

In addition to that, the client may also want to track the progress of the project. During the project execution, it is a must to track the effort and cost of the project in order to determine whether the project is progressing in the right direction or not.

In addition to reporting, there are multiple deliveries to be made during the project execution. Usually, project deliveries are not onetime deliveries made at the end of the project. Instead, the deliveries are scattered through out the project execution period and delivered upon agreed timelines.

4 - Control and Validation

During the project life cycle, the project activities should be thoroughly controlled and validated. The controlling can be mainly done by adhering to the initial protocols such as project plan, quality assurance test plan and communication plan for the project.

Sometimes, there can be instances that are not covered by such protocols. In such cases, the project manager should use adequate and necessary measurements in order to control such situations.

Validation is a supporting activity that runs from first day to the last day of a project. Each and every activity and delivery should have its own validation criteria in order to verify the successful outcome or the successful completion.

When it comes to project deliveries and requirements, a separate team called 'quality assurance team' will assist the project team for validation and verification functions.

5 - Closeout and Evaluation

Once all the project requirements are achieved, it is time to hand over the implemented system and closeout the project. If the project deliveries are in par with the acceptance criteria defined by the client, the project will be duly accepted and paid by the customer.

Once the project closeout takes place, it is time to evaluate the entire project. In this evaluation, the mistakes made by the project team will be identified and will take necessary steps to avoid them in the future projects.

During the project evaluation process, the service provider may notice that they haven't gained the expected margins for the project and may have exceeded the timelines planned at the beginning.

In such cases, the project is not a 100% success to the service provider. Therefore, such instances should be studied carefully and should take necessary actions to avoid in the future.

Conclusion

Project management is a responsible process. The project management process connects all other project activities together and creates the harmony in the project.

Therefore, the project management team should have a detailed understanding on all the project management processes and the tools that they can make use for each project management process.

4 Brief about calculating Earned value measures<u>APR/MAY-12.APRIL/MAY 2018</u>

Earned value analysis is the <u>project management</u> tool that is used to measure project progress. It compares the actual work completed at any time to the original budget and schedule. It forecasts the final budget and schedule and analyzes the path to get there. It gives you the essential early warning signal that things are going awry.

There are two variables which the <u>earned value method</u> focuses on.

- Schedule (time)
- Cost

There are 8 steps to performing earned value analysis effectively. It may seem like alot at first glance, but for small projects this takes five minutes once you learn how to do it:

1. Determine the percent complete of each task.
2. Determine Planned Value (PV).
3. Determine Earned Value (EV).
4. Obtain Actual Cost (AC).
5. Calculate Schedule Variance (SV).
6. Calculate Cost Variance (CV).
7. Calculate Other Status Indicators (SPI, CPI, EAC, ETC, and TCPI)
8. Compile Results

The first four steps represent an information gathering phase. The remaining steps are calculations which give the project manager a glimpse into the current status of the project from a <u>budget</u> and <u>schedule</u> perspective.

Before you get started, it is important to define appropriate project status points in which this calculation is performed. Weekly status meetings work very well for any size project, but whatever time frame is used the important thing is to make sure these calculations are performed at that time.

Determine Percent Complete

To start the process, the percentage complete of each task needs to be determined.

Small tasks (80 hours or less) are often best done on a 0, 50, or 100% complete basis (not started, in progress, or complete). This brings the workload down to reasonable levels and prevents abuse when project team members exaggerate, for example they might tell you a task is 80% complete when it is really 50% complete.

For repetitive tasks you can also use progressive measures such as number of fence posts installed.

Determine Planned Value (PV)

Planned Value, also known as ***Budgeted Cost of Work Scheduled (BCWS)***, is defined as the amount of the task that is supposed to have been completed. It is in monetary terms as a portion of the task budget. For example let's say that:

- The task budget is $5,000,
- The task start date is January 1, and
- The task finish date is January 10.

If it's January 6 today, the task is supposed to be 60% complete. Therefore, *PV = $5,000 x 60% = $3,000.*

Determine Earned Value (EV)

Earned Value, also known as ***Budgeted Cost of Work Performed (BCWP)***, is the amount of the task that is actually complete. It is, again, in monetary terms as a portion of the task budget. For example, let's use the same example task.

- The task budget is $5,000, (same as above)
- The task start date is January 1, and (same as above)
- The task finish date is January 10. (same as above)

Let's say the actual percent complete of the task (step 1) is 40%. Therefore, *EV = $5,000 x 40% = $2,000.*

Obtain Actual Cost (AC)

The ***Actual Cost***, also known as ***Actual Cost of Work Performed (ACWP)***, as you might guess, is the actual cost of the work. Generally employee hours need to be converted into a cost, and all project costs need to be added up, such as the following items:

- Labor
- Materials
- Equipment
- Fixed cost items, like subcontractors

Since most projects have these well defined via accounting or project management software, we will not go into great detail here. For the purposes of our example project let's say the actual cost of the example task is $1,500.

At this point the information gathering phase is complete. The following calculations represent the application of the earned value analysis to keep your project on schedule and budget.

Calculate Schedule Variance (SV)

The Schedule Variance represents the schedule status of the project.

SV = EV – PV

In our above example the schedule variance is: *SV = $2,000 – $3,000 = -$1,000.*

A negative schedule variance means the task is behind schedule. A positive schedule variance means it is ahead of schedule. The amount can be compared to worker charge out rates or similar metrics to get an idea of how difficult it would be to recover.

Calculate Cost Variance (CV)

The Cost Variance represents the cost status of the project.

CV = EV – AC

In our above example the cost variance is: *CV = $2,000 – $1,500 = $500.*

A negative cost variance means the task is over budget. A positive cost variance means it is under budget.

Calculate Other Status Indicators

Although the SV and CV are the minimum requirement and work well for small projects, there are other variables that are derived from them which you might want to calculate:

- **Schedule Performance Index (SPI):** The schedule variance expressed in percentage terms, for example, SPI = 0.8 means the project 20% behind schedule.

SPI = EV / PV

- **Cost Performance Index (CPI):** The cost variance expressed in percentage terms, for example, CPI = 0.9 means the project is 10% over budget.

CPI = EV / AC

- **Estimate at Completion (EAC):** The expected budget at the end of the project given the variances that have already taken place. There are various ways to extrapolate this value but assuming that the past variances are likely to persist:

EAC = AC + BAC – EV

- **Estimate to Complete (ETC):** The expected cost to finish the rest of the project.

ETC = EAC – AC

- **To Complete Performance Index (TCPI):** The required CPI necessary to finish the project right on budget. For example, TCPI = 1.25 means you need to find 25% efficiencies to finish on budget.

TCPI = (BAC – EV) / (BAC – AC)

Compile the Results

Each metric is calculated for each individual task in the project. Therefore they need to be added up into overall project variances to get the overall progress indicator for the project. This represents the total variance of the project and can be reported to management, clients, and stakeholders.

The results are as instantaneous as the input data, that is, if you input the percent complete as of right now the status reported will be as of right now as well. It's amazing how a small variance does not cause anyone concern until they see it as a number, and it can be corrected before it becomes more serious.

Interpreting the Results

The first two calculations (SV and CV) give you the basic indicator of project progress. A negative value indicates an undesirable situation.

- *If the schedule variance (SV) is negative, you are behind schedule.*
- *If the cost variance (CV) is negative, you are over budget.*

The amount of the variance can be compared to the project's budget to see how concerning it is. For example, a variance of $1,000 on a $100,000 project is not that concerning but a $10,000 variance might need some attention. The variances can also be compared to employee charge out rates or something similar, for example a $1,000 variance might require a person who's earning $50/hour to work 20 hours to recover.

In our example the schedule variance was -$1,000 and the cost variance was $500. This means that the project is behind schedule, but it is being performed efficiently and is cost-positive. If an worker charging $75/hr was performing the majority of this work, they are about 13 hours behind schedule (although they will finish under budget). Thus, we know that this task requires a couple days of work over and above the regular schedule to get it back on track.

Graphing the results over multiple status points is a very helpful exercise. Good project control often means that the instantaneous project status snapshot is not as important as the trend the indicators are making over time. For example, if the SV has been increasing, then maybe the project will finish on time even though it's behind schedule today.

It is a well understood concept that if projects fall behind early they will tend to continue falling further behind throughout their entire life. Earned value analysis will alert you if you are even one hour behind and allow you to take the necessary remedial action. The value of this in producing successful projects is almost without equal.

5 Define Risk. Explain the needs and activities or risk management?APR/MAY-15 , NOV/DEC2015 ,NOV/DEC 2017

Risk is inevitable in a business organization when undertaking projects. However, the project manager needs to ensure that risks are kept to a minimal. Risks can be mainly divided between two types, negative impact risk and positive impact risk.

Not all the time would project managers be facing negative impact risks as there are positive impact risks too. Once the risk has been identified, project managers need to come up with a mitigation plan or any other solution to counter attack the risk.

Project Risk Management

Managers can plan their strategy based on four steps of risk management which prevails in an organization. Following are the steps to manage risks effectively in an organization:

- Risk Identification
- Risk Quantification
- Risk Response
- Risk Monitoring and Control

Let's go through each of the step in project risk management:

Risk Identification

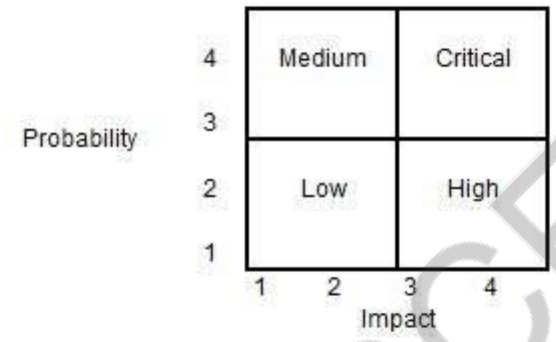
Managers face many difficulties when it comes to identifying and naming the risks that occur when undertaking projects. These risks could be resolved through structured or unstructured brainstorming or strategies. It's important to understand that risks pertaining to the project can only be handled by the project manager and other stakeholders of the project.

Risks, such as operational or business risks will be handled by the relevant teams. The risks that often impact a project are supplier risk, resource risk and budget risk. Supplier risk would refer to risks that can occur in case the supplier is not meeting the timeline to supply the resources required.

Resource risk occurs when the human resource used in the project is not enough or not skilled enough. Budget risk would refer to risks that can occur if the costs are more than what was budgeted.

Risk Quantification

Risks can be evaluated based on quantity. Project managers need to analyze the likely chances of a risk occurring with the help of a matrix.



Using the matrix, the project manager can categorize the risk into four categories as Low, Medium, High and Critical. The probability of occurrence and the impact on the project are the two parameters used for placing the risk in the matrix categories. As an example, if a risk occurrence is low (probability = 2) and it has the highest impact (impact = 4), the risk can be categorized as 'High'.

Risk Response

When it comes to risk management, it depends on the project manager to choose strategies that will reduce the risk to minimal. Project managers can choose between the four risk response strategies, which are outlined below.

- Risks can be avoided
- Pass on the risk
- Take corrective measures to reduce the impact of risks
- Acknowledge the risk

Risk Monitoring and Control

Risks can be monitored on a continuous basis to check if any change is made. New risks can be identified through the constant monitoring and assessing mechanisms.

Risk Management Process

Following are the considerations when it comes to risk management process:

- Each person involved in the process of planning needs to identify and understand the risks pertaining to the project.
- Once the team members have given their list of risks, the risks should be consolidated to a single list in order to remove the duplications.
- Assessing the probability and impact of the risks involved with the help of a matrix.
- Split the team into subgroups where each group will identify the triggers that lead to project risks.
- The teams need to come up with a contingency plan whereby to strategically eliminate the risks involved or identified.
- Plan the risk management process. Each person involved in the project is assigned a risk in which he/she looks out for any triggers and then finds a suitable solution for it.

Risk Register

Often project managers will compile a document, which outlines the risks involved and the strategies in place. This document is vital as it provides a huge deal of information.

Risk register will often consists of diagrams to aid the reader as to the types of risks that are dealt by the organization and the course of action taken. The risk register should be freely accessible for all the members of the project team.

Project Risk; an Opportunity or a Threat?

As mentioned above, risks contain two sides. It can be either viewed as a negative element or a positive element. Negative risks can be detrimental factors that can haphazard situations for a project.

Therefore, these should be curbed once identified. On the other hand, positive risks can bring about acknowledgements from both the customer and the management. All the risks need to be addressed by the project manager.

Conclusion

An organization will not be able to fully eliminate or eradicate risks. Every project engagement will have its own set of risks to be dealt with. A certain degree of risk will be involved when undertaking a project.

The risk management process should not be compromised at any point, if ignored can lead to detrimental effects. The entire management team of the organization should be aware of the project risk management methodologies and techniques.

Enhanced education and frequent risk assessments are the best way to minimize the damage from risks.

6 Explain about all COCOMO models?<u>NOV/DEC 2015, APRIL/MAY2016, APRIL/MAY 2017, APRIL/MAY 2018</u>

Cocomo (Constructive Cost Model) is a regression model based on LOC, i.e **number of Lines of Code**. It is a procedural cost estimate model for software projects and often used as a process of reliably predicting the various parameters associated with making a project such as size, effort, cost, time and quality. It was proposed by Barry Boehm in 1970 and is based on the study of 63 projects, which make it one of the best-documented models.
The key parameters which define the quality of any software products, which are also an outcome of the Cocomo are primarily Effort & Schedule:

- **Effort:** Amount of labor that will be required to complete a task. It is measured in person-months units.
- **Schedule:** Simply means the amount of time required for the completion of the job, which is, of course, proportional to the effort put. It is measured in the units of time such as weeks, months.

Different models of Cocomo have been proposed to predict the cost estimation at different levels, based on the amount of accuracy and correctness required. All of these models can be applied to a variety of projects, whose characteristics determine the value of constant to be used in subsequent calculations. These characteristics pertaining to different system types are mentioned below.

Boehm's definition of organic, semidetached, and embedded systems:

1. **Organic –** A software project is said to be an organic type if the team size required is adequately small, the problem is well understood and has been solved in the past and also the team members have a nominal experience regarding the problem.
2. **Semi-detached –** A software project is said to be a Semi-detached type if the vital characteristics such as team-size, experience, knowledge of the various programming environment lie in between that of organic and Embedded. The projects classified as Semi-Detached are comparatively less familiar and difficult to develop compared to the organic ones and require more experience and better guidance and creativity. Eg: Compilers or different Embedded Systems can be considered of Semi-Detached type.
3. **Embedded –** A software project with requiring the highest level of complexity, creativity, and experience requirement fall under this category. Such software requires a larger team size than the other two models and also the developers need to be sufficiently experienced and creative to develop such complex models.

All the above system types utilize different values of the constants used in Effort Calculations.

Types of Models: COCOMO consists of a hierarchy of three increasingly detailed and accurate forms. Any of the three forms can be adopted according to our requirements. These are types of COCOMO model:

1. Basic COCOMO Model
2. Intermediate COCOMO Model
3. Detailed COCOMO Model

The first level, **Basic COCOMO** can be used for quick and slightly rough calculations of Software Costs. Its accuracy is somewhat restricted due to the absence of sufficient factor considerations.
Intermediate COCOMO takes these Cost Drivers into account and **Detailed COCOMO** additionally accounts for the influence of individual project phases, i.e in case of Detailed it accounts for both these cost drivers and also calculations are performed phase wise henceforth producing a more accurate result. These two models are further discussed below.
Estimation of Effort: Calculations –

4. **Basic Model –**

5.

The above formula is used for the cost estimation of for the basic COCOMO model, and also is used in the subsequent models. The constant values a and b for the Basic Model for the different categories of system:

SOFTWARE PROJECTS	A	B
Organic	2.4	1.05
Semi Detached	3.0	1.12
Embedded	3.6	1.20

The effort is measured in Person-Months and as evident from the formula is dependent on Kilo-Lines of code. These formulas are used as such in the Basic Model calculations, as not much consideration of different factors such as reliability, expertise is taken into account, henceforth the estimate is rough.

6. **Intermediate Model –**

The basic Cocomo model assumes that the effort is only a function of the number of lines of code and some constants evaluated according to the different software system. However, in reality, no system's effort and schedule can be solely calculated on the basis of Lines of Code. For that, various other factors such as reliability, experience, Capability. These factors are known as Cost Drivers and the Intermediate Model utilizes 15 such drivers for cost estimation.

Classification of Cost Drivers and their attributes:

1. **Product attributes –**

- Required software reliability extent
- Size of the application database
- The complexity of the product

(ii) Hardware attributes –

- Run-time performance constraints
- Memory constraints
- The volatility of the virtual machine environment

- Required turnabout time

(iii) Personnel attributes –

- Analyst capability
- Software engineering capability
- Applications experience
- Virtual machine experience
- Programming language experience

(iv) Project attributes –

- Use of software tools
- Application of software engineering methods
- Required development schedule

;

COST DRIVERS	VERY LOW	LOW	NOMINAL	HIGH	VERY HIGH
Product Attributes					
Required Software Reliability	0.75	0.88	1.00	1.15	1.40
Size of Application Database		0.94	1.00	1.08	1.16
Complexity of The Product	0.70	0.85	1.00	1.15	1.30
Hardware Attributes					
Runtime Performance Constraints			1.00	1.11	1.30
Memory Constraints			1.00	1.06	1.21
Volatility of the virtual		0.87	1.00	1.15	1.30

machine environment					
Required turnabout time		0.94	1.00	1.07	1.15
Personnel attributes					
Analyst capability	1.46	1.19	1.00	0.86	0.71
Applications experience	1.29	1.13	1.00	0.91	0.82
Software engineer capability	1.42	1.17	1.00	0.86	0.70
Virtual machine experience	1.21	1.10	1.00	0.90	
Programming language experience	1.14	1.07	1.00	0.95	
Project Attributes					
Application of software engineering methods	1.24	1.10	1.00	0.91	0.82
Use of software tools	1.24	1.10	1.00	0.91	0.83
Required development schedule	1.23	1.08	1.00	1.04	1.10

The project manager is to rate these 15 different parameters for a particular project on a scale of one to three. Then, depending on these ratings, appropriate cost driver values are taken from the above table. These 15 values are then multiplied to calculate the EAF (Effort Adjustment Factor). The Intermediate COCOMO formula now takes the form:

The values of a and b in case of the intermediate model are as follows:

SOFTWARE PROJECTS	A	B
Organic	3.2	1.05
Semi Detached	3.0	1.12
Embeddedc	2.8	1.20

2. **Detailed Model –**

Detailed COCOMO incorporates all characteristics of the intermediate version with an assessment of the cost driver's impact on each step of the software engineering process. The detailed model uses different effort multipliers for each cost driver attribute. In detailed cocomo, the whole software is divided into different modules and then we apply COCOMO in different modules to estimate effort and then sum the effort.

The Six phases of detailed COCOMO are:

1. Planning and requirements
2. System design
3. Detailed design
4. Module code and test
5. Integration and test
6. Cost Constructive model

The effort is calculated as a function of program size and a set of cost drivers are given according to each phase of the software lifecycle.

7 Write about software maintenance, PERT - CPM for scheduling , RMMP <u>NOV/DEC-12</u>

Project Scheduling or project management is used to schedule, manage and control projects which can be analyzed into various semi-independent activities or tasks. Example: Building a New Home When building a home individual subcontractors are hired to:

— Grade and prepare the land
— Build the foundation
— Frame up the home
— Insulate the home
— Wire (Electricity, Cable, Telephone lines) the home — Drywall

— Paint (inside)
— Put vinyl siding on home
— Install Carpet
— Landscape
— Lay Concrete

PERT – Program Evaluation and Review Technique
– Developed by U.S. Navy for Polaris missile project
– Developed to handle uncertain activity times CPM
– Critical Path Method
– Developed by Du Pont & Remington Rand
– Developed for industrial projects for which activity times are known

There are project management software packages that can perform both.
PERT and CPM have been used to plan, schedule, and control a wide variety of projects:
– R&D of new products and processes
– Construction of buildings and highways
– Maintenance of large and complex equipment
– Design and installation of management systems
– Organizing transportation projects
– Deployment and/or relocation of forces
– Design of computer systems

- PERT/CPM is used to plan the scheduling and optimal staffing of individual activities that make up a project.
- Projects may have as many as several thousand activities and may have to be broken up into simpler sub-projects.
- Usually some activities depend on the completion of other activities before they can be started.
- So we need to start with the Prerequisites Task Set giving the order of precedencies, along with durations for each task, or activity

8 Describe steps involved in project scheduling process, project timeline chart and task network. **<u>MAY/JUN-15, APRIL/MAY 2018</u>**

Press-Pg-no- 708

9 **(b) Suppose you have a budgeted cost of a project as Rs. 9,00,000. The project is to be completed in 9 months. After a month you have completed 10 percent of project at a total expense of Rs. 1,00,000. The planned completion should have been 15 percent. You need to determine whether the project is on-time and on budget? Use Earned value analysis approach and interpret<u>NOV/DEC 2016</u>**

(c) Consider the following function point components and their complexity. If the total degree of influence is 52, find the estimated function points.

Function type	**Estimated count**	**complexity**
FED	**2**	**7**
GHD		**4**
10		
HJI	**22**	**4**
BU	**16**	**5**
BJ	**24**	**4**

Refer class notes

10 **Describe in detail COCOMO model for software cost estimation. Use it to estimate the effort required to build software for a simple ATM that produce 12 screens, 10 reports and has 80 software components. Assume average complexity and average developer maturity. Use application composition model with object points.<u>NOV/DEC 2016, NOV/DEC 2017</u>**

Refer class notes

11 **Explain the process of function point analysis?explain function point analysis with sample cases for componentfor different complexity <u>APRIL/MAY 2018</u>**

Refer class notes

12 **Discuss on the various software cost estimation techniques. (April/MayApr/May 2008)**

Refer class notes

13 **Explain the process of Delphi method ? advantages and disadvantages (Nov/Dec 2013)**

Refer class notes

14 **Explain about Risk management (May/Jun 2014)**

Risk management is the identification, evaluation, and prioritization of risks (defined in ISO 31000 as *the effect of uncertainty on objectives*) followed by coordinated and economical application of resources to minimize, monitor, and control the probability or impact of unfortunate events[1] or to maximize the realization of opportunities.

Risks can come from various sources including uncertainty in financial markets, threats from project failures (at any phase in design, development, production, or sustaining of life-cycles), legal liabilities, credit risk, accidents, natural causes and disasters, deliberate attack from an adversary, or events of uncertain or unpredictable root-cause. There are two types of events i.e. negative events can be classified as risks while positive events are classified as opportunities. Risk management standards have been developed by various institutions, including the Project Management Institute, the National Institute of Standards and Technology, actuarial societies, and ISO standards.[2][3] Methods, definitions and goals vary widely according to whether the risk management method is in the context of project management, security, engineering, industrial processes, financial portfolios, actuarial assessments, or public health and safety.

Strategies to manage threats (uncertainties with negative consequences) typically include avoiding the threat, reducing the negative effect or probability of the threat, transferring all or part of the threat to another party, and even retaining some or all of the potential or actual consequences of a particular threat, and the opposites for opportunities (uncertain future states with benefits).

Certain risk management standards have been criticized for having no measurable improvement on risk, whereas the confidence in estimates and decisions seems to increase.[1] For example, one study found that one in six IT projects were "black swans" with gigantic overruns (cost overruns averaged 200%, and schedule overruns 7

15 Give detail explanation about Scheduling and Tracking

Project Scheduling helps to establish a roadmap for project managers together with estimation methods and risk analysis. Project scheduling and Tracking begins with the identification of process models, identification of software tasks and activities, estimation of effort and work and ends with creation of network of tasks and making sure it gets done on time. This network is adapted on encountering of changes and risks.

At the project level, the Project Manager does project tracking and scheduling based on information received from Software Engineers. At an individual level the Software Engineer does it. It is important, as in a complex system many tasks may occur in parallel and have interdependencies that are understandable only with a schedule. A detailed schedule is a useful tool to assess progress on a moderate or large project.

The basic steps followed are, once the tasks dictated by the software process model is refined based on the functionality of the system , effort and duration are allocated for each task and an activity network is created that allows the project to meets its deadlines. The work product of this activity is the project schedule and in order that it is accurate it is required to check all tasks are covered in the activity network, effort and timing are appropriately allocated, interdependencies are correctly indicated, resources are allocated tasks in a right manner and closely spaced milestones are defined to track the project easily.

One of the major challenges in software project management is the difficulty to adhere to schedules. The common reasons for a late delivery of software project are an unrealistic deadline, changing customer requirements, honest underestimate of effort or resources, overlooked risks, unforeseen technical difficulties or human difficulties, miscommunication and failure by project manager to recognize the delay early and take appropriate measures.

Software project scheduling is an activity that distributes estimated effort across the duration of project cycle by allocating effort to each specific task that is associated with all process. The basic principles that guides software project scheduling is compartmentalization of the project into a number of manageable tasks, correct allocation of time, correct effort validation ,defining responsibility for each task to a team member, defining outcomes or work product for each task and defining milestones for a task or group of tasks as appropriate.

A Task set is a collection of software tasks, milestones and deliveries that must be completed for the project to be successfully accomplished. Task sets are defined for being applicable to different type of project and degree of rigor. The types of projects commonly encountered are Concept Development projects, New applications, Development projects, Application enhancement projects, Application maintenance projects and Re-engineering projects. The degree of rigor with which the software process is applied may be casual, structured, strict or quick reaction (used for emergency situation).

For the project manager to develop a systematic approach for selecting degree of rigor for the type of project project adaptation criteria are defined and a task set selector value is computed based on the characteristics of the project.

Program evaluation and review technique (PERT) and critical path method (CPM) are two of the commonly used project scheduling methods. These techniques are driven by information such as estimates of effort, decomposition of the product function, the selection of process model, task set and decomposition of tasks. The interdependencies among tasks are defined using a task network.

A task network or activity network is a graphic representation of the task flows for a project. According to basic PERT, expected task duration is calculated as the weighted average of the most pessimistic, the most optimistic and most probable time estimates. The expected duration of any path on the network is found by summing the expected durations of tasks.

PERT gives appropriate results when there is a single dominant path in the network. The time needed to complete the project is defined by the longest path in the network which is called critical path. CPM allows software planner to determine the critical path and establish most likely time estimates.

While creating schedule a timeline chart also called as Gantt chart can be generated. This can be developed for entire project or separately for each function or individual.

The information necessary for generation of this is Work Breakdown Structure (WBS – Making a complex project manageable by breaking it into individual components in a hierarchical structure that defines independent tasks), effort, duration and start date and details of assignment of tasks to resources. Along with this most software project scheduling tools produce project tables which is a tabular listing o project tasks, their planned and actual start, end dates and other information. This with timeline chart is valuable to project managers to track the progress.

Tracking the project schedule can be done by conducting periodic project status meeting, evaluating result of reviews conducted at all stages of development life cycle, determining the completion of defined project milestones, comparison of actual and planned dates, using earned value analysis technique for performing quantitative analysis of program.

Error tracking methods can also be used for assessing the status of current project. This is done by collecting error related measures and metrics from past project and using this as baseline for comparison against real time data.