Total No. of Questions : 8] *[Total No. of Printed Pages : 2*

Roll No .....

IT-6002 (CBGS)

B.E. VI Semester

Examination, May 2019

Choice Based Grading System (CBGS)

Software Engineering and Project Managements

Time : Three Hours

Maximum Marks : 70

Note: i) Attempt any five questions.
ii) All questions carry equal marks.

1. a) What is Agile development? Give brief overview of how development occurs in Agile philosophy. 7
   b) Explain the unified approach to software development. Discuss the merits and demerits of this approach. 7

2. a) What do you understand by a layered software design? What are the advantages of a layered design? 7
   b) Explain the waterfall model. Explain why it is more advantageous than adhoc methods. 7

3. a) What is function point? Explain its importance. What is function-oriented metrics? 7
   b) Explain how a software development effort is initiated and finally terminated in the spiral model. Also explain why the spiral life cycle model is considered to be a meta model. 7

4. a) What are the main activities carried out during requirements analysis and specification phase? What is the final outcome of the requirements analysis and specification phase? 7

b) Describe the design process in software development. What are the characteristics and criteria for design? 7

5. a) What is black box-testing? Is it necessary to perform this? Explain various test activities. 7

b) Explain the integration testing process and system testing processes and discuss their outcomes. 7

6. a) Discuss the impact of cohesion, coupling, fan-in, fan-out and factoring in design phase. 7

b) Briefly outline the five steps involved in constructing a Work Break Down structure of a software project. 7

7. a) List the various software quality attributes and explain briefly. http://www.rgpvonline.com 7

b) What is the role of project managers in conducting a successful SQA program? 7

8. a) Define the term Component? What are the benefits of CBSE (Component Based Software Engineering). 7

b) What is risk management? Explain briefly the technical risks in a software project. 7

******

36

Ans 1) A) Agile is the ability to create and respond to change. It is a way of dealing with, and ultimately succeeding in, an uncertain and turbulent environment.

Agile software development is an umbrella term for a set of frameworks and practices based on the values and principles expressed in the Manifesto for Agile Software

Development and the 12 Principles behind it. When you approach software development in a particular manner, it's generally good to live by these values and principles and use them to help figure out the right things to do given your particular context.
One thing that separates Agile from other approaches to software development is the focus on the people doing the work and how they work together. Solutions evolve through collaboration between self-organizing cross-functional teams utilizing the appropriate practices for their context.

There's a big focus in the Agile software development community on collaboration and the self-organizing team.

That doesn't mean that there aren't managers. It means that teams have the ability to figure out how they're going to approach things on their own.

It means that those teams are cross-functional. Those teams don't have to have specific roles involved so much as that when you get the team together, you make sure that you have all the right skill sets on the team.

There still is a place for managers. Managers make sure team members have, or obtain, the right skill sets. Managers provide the environment that allows the team to be successful. Managers mostly step back and let their team figure out how they are going to deliver products, but they step in when the teams try but are unable to resolve issues.

When most teams and organizations start doing Agile software development, they focus on the practices that help with collaboration and organizing the work, which is great. However, another key set of practices that are not as frequently followed but should be are specific technical practices that directly deal with developing software in a way that help your team deal with uncertainty.

The Agile Manifesto

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.

The following 12 Principles are based on the Agile Manifesto.

1

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

2

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

3

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

4

Business people and developers must work together daily throughout the project.

5

Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

6

The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

7

Working software is the primary measure of progress.

8

Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

9

Continuous attention to technical excellence and good design enhances agility.

10

Simplicity–the art of maximizing the amount of work not done–is essential.

11

The best architectures, requirements, and designs emerge from self-organizing teams.

12

At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Ans 1) B) UML as the name suggests has come this far through the process of unification and thus combines ideas of leading thinkers and gives access to their methodology of modeling in consistency with that of the others. Now we are planning to get down a level further and to represent these modeling methodologies on the same canvas.
To bring these entities under the same frame of reference we need first to look for a commonality that is not far stretched or bolted out of blue sky. Such point of joining between the static logical view and the dynamic interaction view is in terms of objects in the interaction view as also the object view that materialize as instances of classifiers (classes) in the logical view (class diagram).

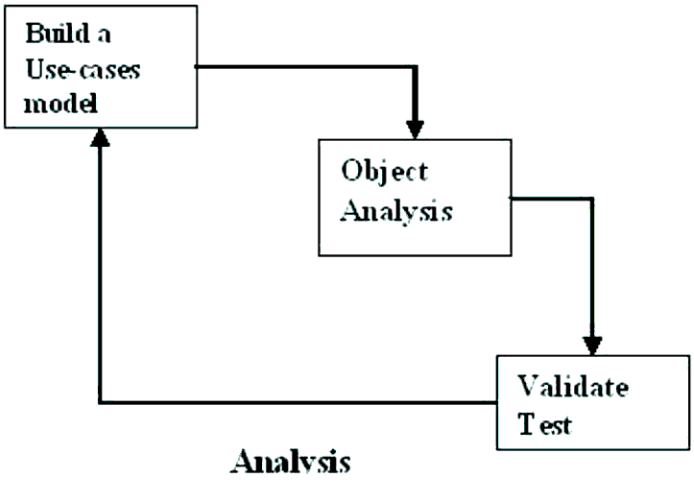
The unified approach to software development revolves around the following processes and concepts.

The processes are,

- Object oriented analysis
- Object oriented design
- Testing
- Developing and Prototyping

Object oriented analysis:

Analysis is the first stage of the development process. It is also the first step in producing high-performance software. Analysis is central to the performance tuning process.
The analysis part having these below steps –



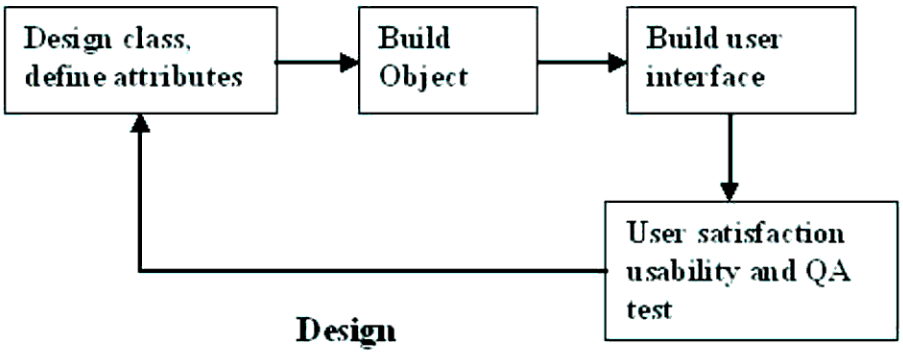
OOA process consists of the following steps:

- Identify the actors
- Develop a simple business process model using UML Activity diagram
- Develop the usecase
- Develop interaction diagrams
- Identify classes

Object-Oriented Design

Object – oriented design plays a major role in the performance process. While there are many factors that contribute to a good design, and all of them are important, there is one concept that is especially critical to creating high-performance systems. This concept is called encapsulation.
Making encapsulation part of your design from the start enables you to:

- Quickly evaluate different algorithms and data structures to see which is most efficient.
- Easily evolve your design to accommodate new or changed requirements.



OOD process consist of,

- Designing classes, their attributes, methods, associations, structures and protocols, apply design axioms
- Design the Access Layer
- Design the prototype user interface
- User satisfaction and usability tests based on the usage/use cases

- Iterate and refine the design

Iterative development and continuous testing:

Iterate and Reiterate until you are satisfied with the system. Testing uncovers the design weaknesses or provides additional information. Repeat the entire process, taking what you have learned and reworking you design more onto re-prototyping and retesting.

Unified Approach encourages integration testing from the day 1 of the project usage scenarios can become test scenarios. There fore use cases will drive the usability testing.

Modeling based on the Unified Modeling Language:

UML is becoming the universal language for modeling systems; it is intended to be used to express models of many different kinds and purposes. UML has become the standard notation for object oriented modeling systems.

The **Unified Approach** uses the UML to describe and model the analysis and design phases of system development.

Unified Approach Proposed Repository:

Best practice sharing eliminates duplication of problem solving. It must be applied to application development, if quality and productivity are to be added. The idea promoted here is to create a repository that allows the maximum reuse of previous experience and previously defined objects, patterns, frameworks etc in an easily accessible manner. It should be accessible to many people.

Merits:

- If your organization has done projects in the past, objects in the repositories from those projects might be useful.
- Creating additional applications will require no more than assembling components from the library.
- Applying lessons learned from the part will increase the quality of the product and reduce the cost and development time.

Demerits of RUP **Software Development**

- The team members need to be expert in their field to **develop** a **software** under this **methodology**.
- The **development** process is too complex and disorganized.
- On cutting edge projects which utilise new technology, the reuse of components will not be possible.

Ans 2) A) **SOFTWARE ENGINEERING: A LAYERED TECHNOLOGY**

- The bedrock that supports s/e is the quality focus.
- Process defines a framework for a set of key process areas (KPAs) that must be established for effective delivery of software engineering technology.

Key process areas include :

The key process areas establish the context in which technical methods are applied, work products (models, documents, data, reports, forms, etc.) are produced, milestones are established, quality is ensured, and change is properly managed.

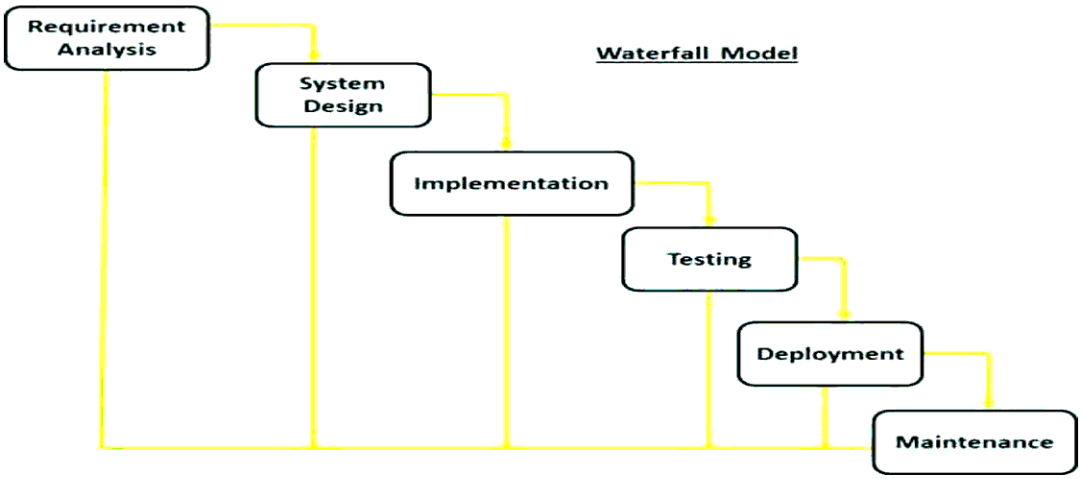
- Methods provide the technical how-to's for building software.
- Software engineering tools provide automated or semi- automated support for the process and the methods.

Eg. CASE tool – computer aided software engineering

Advantage

1) The **layered** structure makes it easier to perform user acceptance **testing** prior to deploying enhancements or bug fixes.
2) Ease of **development**: The **layers** of isolation **approach** allows for a smooth, agile **approach** to new **development**

Ans 2) B) **WaterFall Model**



Requirement Gathering and analysis: All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification doc.

Requirement produces pieces and parts along with their relationships to each other. They are basically the Noun Phrases of SRS.

Next it defines Actions that the system should perform. Eg Methods and Function names which are basically the Verb phrases of SRS.

Also **Feasibility study** is performed which is a part of Analysis phase for determining whether it practically Possible/Feasible to develop the software . They are of many types , Eg. :-

1) Economic Feasibility – Finding out whether the software firm will gain enough profit

2) Technical Feasibility :- Is the software firm is having enough Technical persons and the software technology is available with the firm or not.

3) Time Feasibility :- Whether the software can be developed in Time or not. Etc.

System Design –

1) Here the Architecture is developed also known as Blueprint of the system

2) Requirement document acts as an input to this phase

3) Here Requirements are mapped into Architecture.

4) System Design phase produces following documents:

a) Architecture

b) Implementation plan

c) Critical priority tasks

d) Performance Analysis

e) Test Plan

Architecture defines the Components, interfaces and behaviors of the system.

These Components can be built from Scratch or can be

Re-used from the existing component Library.

- Components performs specific Task.

- Interfaces are the boundary of the component through which it can communicate with other components or other components can communicate with this component.
- The **implementation plan** establishes the schedule and needed resources. It defines implementation details including programming languages, platforms, programming environments, debuggers, and many more.
- The **critical priority analysis** generates a list of **critical tasks**. It is absolutely necessary to successfully accomplish a critical task. The project will succeed or fail based on the outcome of these tasks. Some projects may have more than one critical task.
- **Performance Analysis**

Once given the typical scenarios from the requirement document, the system can be designed to meet performance objectives depending on the usage frequency of different scenarios.

- The **test plan** defines the testing necessary to establish quality for the system. If the system passes all tests in the test plan, then it is declared to be complete.
- **Implementation:** With inputs from system design, the system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality which is referred to as Unit Testing.
- **Integration and Testing:** All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.
- **Deployment of system:** Once the functional and non functional testing is done, the product is deployed in the customer environment or released into the market.
- **Maintenance:** There are some issues which come up in the client environment. To fix those issues patches are released. Also to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.

Advantages

- Simple and easy to understand and use
- Easy to manage due to the rigidity of the model
- Phases are processed and completed one at a time.
- Works well for smaller projects where requirements are very well understood.

Disadvantage

- There is no way to go back from next stage to previous stage.
- No working software is produced until late during the life cycle.

- High amounts of risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for large projects.
- Cannot accommodate changing requirements.

Ans 3) A) **Function-Oriented Metrics**

- Function-oriented software metrics use a measure of the functionality delivered by the application as a normalization value.
- Since 'functionality' cannot be measured directly, it must be derived indirectly using other direct measures.
- Function oriented metrics uses Function point which are derived using an empirical relationship based on Information domain & software complexity.

Five information domain characteristics were determined.

- Weighting Factors
- ↘ ↘ ↘

Measurement parameters	counts		simple	avg	complex		
No of user I/p's	☐	*	3	4	6	=	☐
No of user O/p's	☐	*	4	5	7	=	☐
No of user inquiries	☐	*	3	4	6	=	☐
No of Files	☐	*	7	10	15	=	☐
No of external interfaces	☐	*	5	7	10	=	☐
Count total	→						☐

Five Information Domain characteristic were determined :

1) Number of user inputs. Each user input that provides distinct application oriented data to the software is counted.

2) Number of user outputs. Each user output that provides distinct application oriented information to the user is counted. In this context output refers to reports, screens, error messages, etc.

3) Number of user inquiries. An inquiry is defined as an on-line input that results in the generation of some immediate software response in the form of an on-line output. Each distinct inquiry is counted.

4) Number of files. Each logical master file (i.e., a logical grouping of data that may be one part of a large database or a separate file) is counted.

5) Number of external interfaces. All machine readable interfaces that are used to transmit information to another system are counted.

To compute function points (FP), the following relationship is used:

$$FP = \text{count total} \ [0.65 + 0.01 \ \Sigma(Fi)]$$

where count total is the sum of all FP entries in the above Figure.

The *Fi (i = 1 to 14) are "complexity adjustment values" based on responses to the* following questions:

1. Does the system require reliable backup and recovery?

2. Are data communications required?

3. Are there distributed processing functions?

4. Is performance critical?

5. Will the system run in an existing, heavily utilized operational environment?

6. Does the system require on-line data entry?

7. Does the on-line data entry require the input transaction to be built over multiple screens or operations?

8. Are the master files updated on-line?

9. Are the inputs, outputs, files, or inquiries complex?

10. Is the internal processing complex?

11. Is the code designed to be reusable?

12. Are conversion and installation included in the design?

13. Is the system designed for multiple installations in different organizations?

14. Is the application designed to facilitate change and ease of use by the user?

Each of these questions is answered using a scale that ranges from 0 (not important or applicable) to 5 (absolutely essential).

Once function points have been calculated, they are used in a manner analogous to LOC as a way to normalize measures for software productivity, quality, and other attributes:

- Errors per FP.
- Defects per FP.
- $ per FP.
- Pages of documentation per FP.
- FP per person-month.

Arguments in favor of Function point and its extensions :

1) It is programming language independent, making it ideal for applications using conventional and nonprocedural languages

2) it is based on data that are more likely to be known early in the evolution of a project

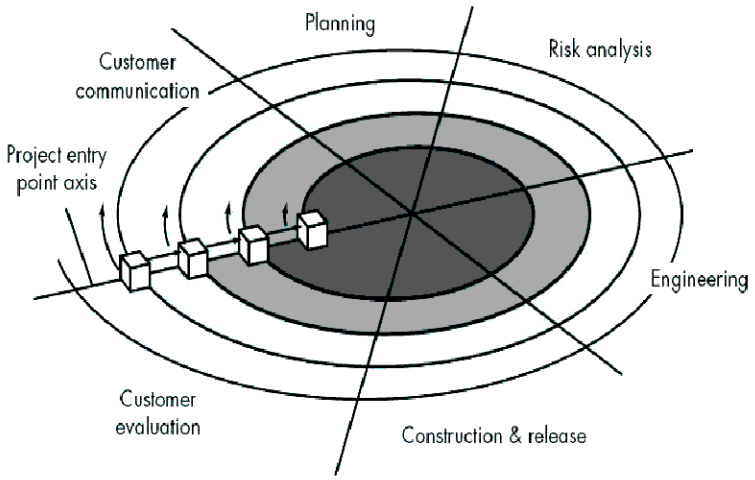
Arguments in oppose :

1) Computation is based on subjective rather then objective data

2) Counts of the information domain can be difficult to achieve.

3) FP has no direct physical meaning – its just a number

Ans 3) B) Spiral Model



Risk Analysis includes identifying, estimating, and monitoring technical feasibility and management risks, such as schedule slippage and cost overrun.

Engineering Design: Design phase starts with the conceptual design in the baseline spiral and involves architectural design, logical design of modules, physical product design and final design in the subsequent spirals

Construction : Construct phase refers to production of the actual software product at every spiral.

In the baseline spiral when the product is just thought of and the design is being developed a POC (Proof of Concept) is developed in this phase to get customer feedback.

Then in the subsequent spirals with higher clarity on requirements and design details a working model of the software called build is produced with a version number. These builds are sent to customer for feedback.

Customer Evaluation : After testing the build, at the end of first iteration, the customer evaluates the software and provides feedback.

Advantage:-

1) Changing requirements can be accommodated.

2) Requirements can be captured more accurately.

3) Users see the system early.

4) Development can be divided into smaller parts and more risky parts can be developed earlier which helps better risk management

Disadvantage:-

1) Management is more complex.

2) Not suitable for small or low risk projects and could be expensive for small projects.

3) Process is complex

4) Large number of intermediate stages requires excessive documentation.

The **Spiral model is called** as a **Meta Model** because it subsumes all the other SDLC **models**. For example, a single loop **spiral** actually represents the Iterative Waterfall **Model**. ...
The **spiral model** uses the approach of Prototyping **Model** by building a prototype at the start of each phase as a risk handling technique.

Ans 4) A)

Requirements Analysis

Objective

The objective of this phase is to define in more detail the system inputs, processes, outputs and interfaces. At the end of this phase the system's processes will be defined at the functional level, meaning the functions to be performed will be known, but not necessarily how they will be performed. Unless specifically constrained by the Project Charter, *Requirements Analysis* should **not** consider the computer programs, files and data streams.

Requirements Analysis will identify and consider the risks related to how the technology will be integrated into the standard operating procedures. Requirements Analysis will collect the functional and system requirements of the business process, the user requirements and the operational requirements (e.g., when operational what is necessary to keep the system up and running).

Entry Criteria

In order for Requirements Analysis to begin, there must be an approved Project Charter. The scope of the project will be understood and stated in the Project Charter. The roles and responsibilities for the various activities in the Development Life cycle will be known.

Roles & Responsibilities

Project Manager: The project manager is accountable for the success of this phase. The primary responsibility of the Project Manager during Requirements Analysis is to ensure the Business Analyst(s) has access to the proper: Subject Matter Experts, Business Process documentation, existing technology and potential technological solutions as well as current and desired artifacts. A major impediment to successful requirements analysis is lack of exposure to any of the previously listed items

Business Analyst – The BA must first develop a plan for how the requirements analysis activity will be accomplished. The BA must then document the business process descriptions and collect the requirements of the system from the Subject Matter Experts (SME's) in a manner which allows traceability to documents generated in previous activities and creates a framework for future activities. All identified requirements should fall within the constraints of the project scope and align with the customer's statement of needs. The BA will generate a requirements traceability matrix which becomes the basis for the *Design activity*.

The BA constructs a logical model that describes the data and processes needed to support the requested functionality. Processes are identified along with any of the associated data they use or create. The interactions of dependent processes is also defined.

Because of the variability in scope of the projects intended to fall within the confines of this life cycle, it is expected that the BA will need a flexible set of tools to properly elicit and document the business requirements. The BA will work closely with the SME's to ensure a logical model showing processes, data structures and business activities in an accurate, consistent and complete manner.

The BA must also consider the current technical architecture, application software and data that is used to support the business function to guarantee that no necessary functionality has been overlooked. The BA will also be aware of considerations surrounding persons with disabilities and any other legal considerations. Some consideration must also be paid to capacity and growth associated with the project.

Subject Matter Expert – The Subject Matter Expert understands the current business processes and any new requirements that are to be satisfied by the project. They must work closely with the BA to transfer both stated and tacit knowledge for inclusion in the Functional Requirements Documents

Designer – The Designer receives the Artifacts produced in the Requirements Analysis phase. Because of the variability in scope of the projects intended to fall within the confines of this life cycle, it is expected that the Designers will coordinate with the BA early on to ensure there is an agreed upon set of artifacts delivered.

Testing Lead – The Testing Lead is involved during this activity to ensure that the requirements identified by the BA and accepted by the customer are measurable and that IT has the resources to complete adequate testing. Having the Testing Lead involved at this step allows for proper scheduling and preparation for the various stages of testing that occur during the SDLC.

Artifacts

SRS – Software Requirement specification*): This is a document that defines in more detail the system inputs, processes, outputs and interfaces.*

- *Entity-relationship diagrams (optional)*
- *process hierarchy diagrams (optional)*
- *process dependency diagrams (optional)*
- *logical model (optional)*
- *activity diagrams (optional)*
- *business algorithms (optional)*
- *entity life cycle diagrams (optional)*
- *entity state change matrices (optional)*
- *Mockups (optional)*
- *Data Flow Diagrams: Intended to represent the flow of information around a system (optional)*
- Requirements Traceability Matrix (included at end of FRD)

Review *(Revision)*

The completion of Requirements Analysis is signified by a presentation of the SRS to the Customer and the Designers. At this point the Project Manager should also review the SRS for the time line for the remaining life

cycle phases, review of resource availability and a risk assessment organized to align with the remaining steps of the life cycle.

Approval

When the customer signs off on the Business Requirements the solution designers can begin their system design work. Depending on the technique of development there might be more than one visit to the Requirements Analysis activity. It is important that the Business Analyst, Designers and Customer agree and understand the expected iterations of Requirements analysis.

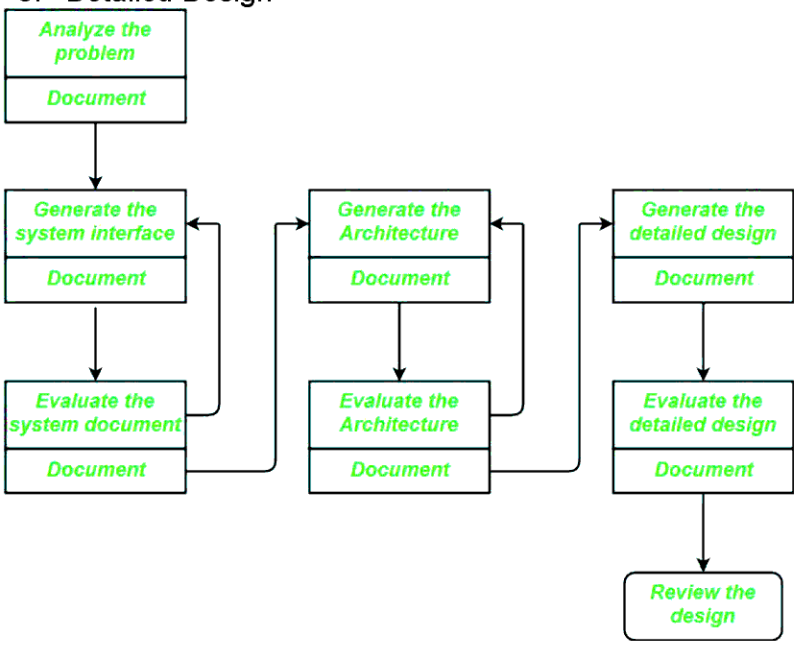
Exit Criteria

Requirements Analysis is complete when the customer signs off on the Software Requirements Specification Document.

Ans 4) B) Software Design Process

The design phase of software development deals with transforming the customer requirements as described in the SRS documents into a form implementable using a programming language. The software design process can be divided into the following three levels of phases of design:

1. Interface Design
2. Architectural Design
3. Detailed Design



Interface Design:
Interface design is the specification of the interaction between a system and its environment. this phase proceeds at a high level of abstraction with respect to the inner workings of the system i.e, during interface design, the internal of the systems are completely ignored and the system is treated as a black box. Attention is focussed on the dialogue between the target system and the users, devices, and other systems with which it interacts. The design problem statement produced during the problem analysis step should identify the people, other systems, and devices which are collectively called *agents*.

Interface design should include the following details:

- Precise description of events in the environment, or messages from agents to which the system must respond.
- Precise description of the events or messages that the system must produce.
- Specification on the data, and the formats of the data coming into and going out of the system.
- Specification of the ordering and timing relationships between incoming events or messages, and outgoing events or outputs.

Architectural Design:

Architectural design is the specification of the major components of a system, their responsibilities, properties, interfaces, and the relationships and interactions between them. In architectural design, the overall structure of the system is chosen, but the internal details of major components are ignored.

Issues in architectural design includes:

- Gross decomposition of the systems into major components.
- Allocation of functional responsibilities to components.
- Component Interfaces
- Component scaling and performance properties, resource consumption properties, reliability properties, and so forth.
- Communication and interaction between components.

The architectural design adds important details ignored during the interface design. Design of the internals of the major components is ignored until the last phase of the design.

Detailed Design:

Design is the specification of the internal elements of all major system components, their properties, relationships, processing, and often their algorithms and the data structures.

The detailed design may include:

- Decomposition of major system components into program units.
- Allocation of functional responsibilities to units.
- User interfaces
- Unit states and state changes
- Data and control interaction between units
- Data packaging and implementation, including issues of scope and visibility of program elements
- Algorithms and data structures

Ans 5) A)

BLACK BOX TESTING is defined as a testing technique in which functionality of the Application Under Test (AUT) is tested without looking at the internal code structure, implementation details and knowledge of internal paths of the software. This type of testing is based entirely on software requirements and specifications. In BlackBox Testing we just focus on inputs and output of the software system without bothering about internal knowledge of the software program.

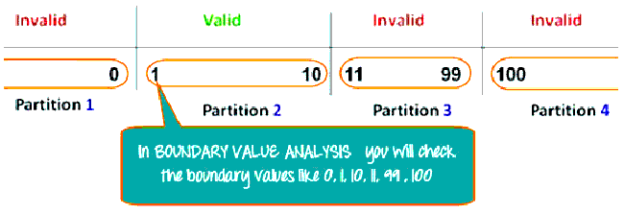
The above Black-Box can be any software system you want to test. For Example, an operating system like Windows, a website like Google, a database like Oracle or even your own custom

application. Under Black Box Testing, you can test these applications by just focusing on the inputs and outputs without knowing their internal code implementation.

Techniques of Black Box Testing

The following are the techniques employed while using Black box testing for a software application.

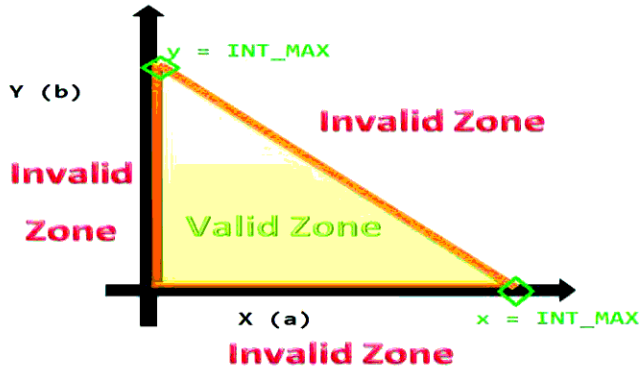
BVA or Boundary Value Analysis:



It is one among the useful and critical Black box testing technique that helps in equivalence partitioning. BVA helps in testing any software having a boundary or extreme values.

This technique is capable of identifying the flaws of the limits of the input values rather than focusing on the range of input value. Boundary Value Analysis also deals with edge or extreme output values.

Equivalence Class Partitioning:

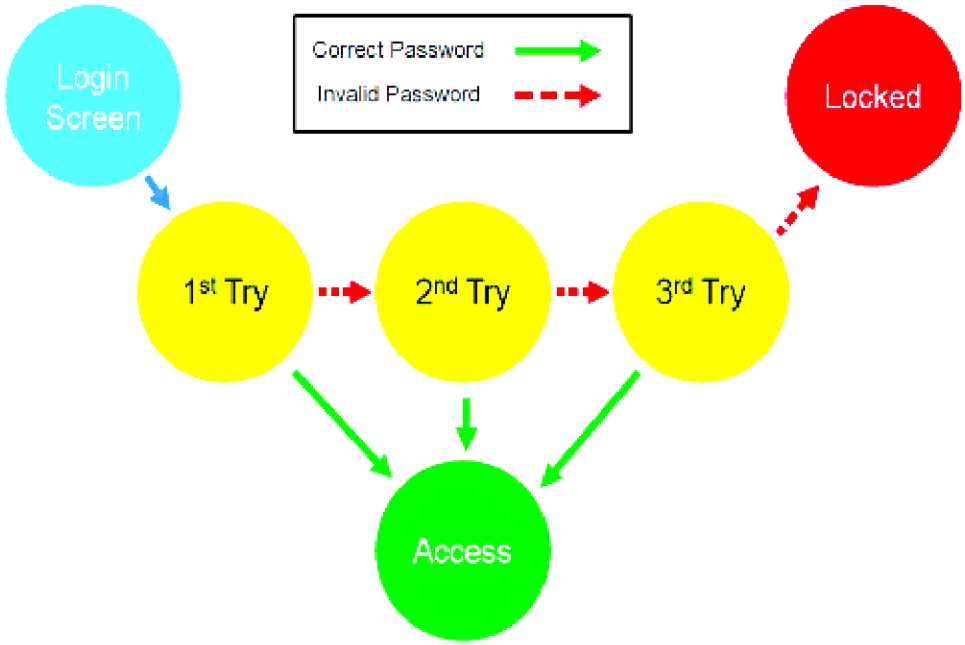


This technique of Black box testing is widely used to write test cases. It can be useful in reducing a broad set of possible inputs to smaller but effective ones.

It is performed through the division of inputs as classes, and each class is given a value.

It is applied when the need for exhaustive testing arises and for resisting the redundancy of inputs.

State Transition Testing



This technique usually considers the state, outputs, and inputs of a system during a specific period.

Based on the type of software that is tested, it checks for the behavioral changes of a system in a particular state or another state while maintaining the same inputs.

The test cases for this technique are created by checking the sequence of transitions and state or events among the inputs.

The whole set of test cases will have the traversal of the expected output values and all states.

Decision Table Testing:

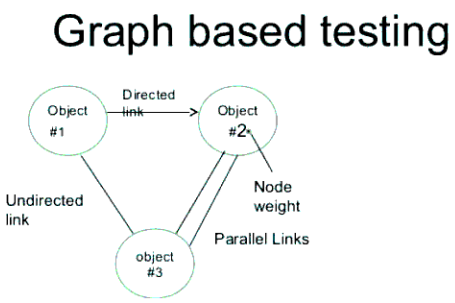
	Rule 1	Rule 2	Rule 3	Rule 4
Condition				
End of month	No	Yes	Yes	Yes
Salary Transferred	N/A	No	Yes	Yes
Provident fund	N/A	N/A	No	Yes
Action				
Income tax	No	No	Yes	Yes
Provident fund	No	No	No	Yes

In some instances, the inputs combinations can become very complicated for tracking several possibilities.

Such complex situations rely on decision tables, as it offers the testers an organized view about the inputs combination and the expected output.

This technique is identical to the graph-based testing technique; the major difference is using tables instead of diagrams or graphs.

Graph-Based Testing:



This technique of Black box testing involves a graph drawing that depicts the link between the causes (inputs) and the effects (output), which trigger the effects.

This testing utilizes different combinations of output and inputs. It is a helpful technique to understand the software's functional performance, as it visualizes the flow of inputs and outputs in a lively fashion.

Error Guessing Technique:

This testing technique is capable of guessing the erroneous output and inputs to help the tester fix it easily. It is solely based on judgment and perception of the earlier end user experience.

Ans 5) B)

INTEGRATION TESTING

If all the modules are unit tested then why will the problem arise when they are combined ???

Problem comes in many ways – data may be lost in interface or some imprecise data may be magnified to unacceptable range & so on.

Integration testing :- Testing the interaction between the modules and interaction with other systems externally is called integration testing.

Three approaches are possible :-

Big Bang approach

Top down integration

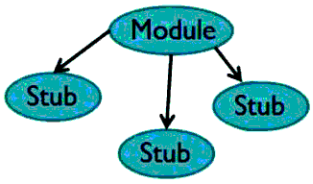
Bottom up integration

1) **Big Bang approach :-** It takes all the components at once & single round of integration is done.

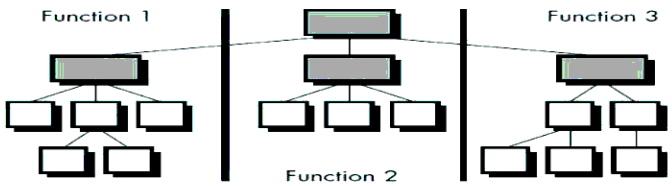
   Adv :- Repetition of testing the components is not required.

   Disadvantage :- For larger project the amount of errors which results are very large & complex in nature.

2) **Top down integration :-** It starts with the main program ie. Root of the tree.
   a) Lower level units are called stubs which are throw away units.
   b) It only contains interfaces with very little code so that main program can run
   c) Once the main program logic is tested completely , stubs are replaced with actual components one after another.



   d) Stubs can be developed in depth first or breadth first manner.
      i) In **depth first** we are testing the complete single functionality at one time then going to develop another one. So partial delivery to the customer is possible.

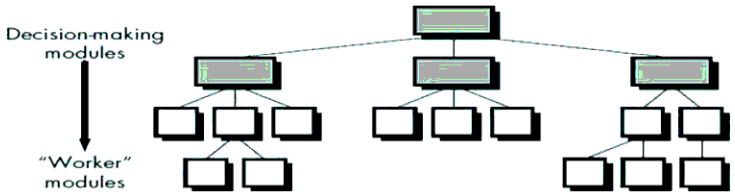


Depth First

ii) In **Breadth first** we test top most level then next level & so on.

It works on the concept that majority of the decision making modules are present in the upper layers so once those are successfully tested then we can be sure that we don't require major changes & even if they fail then since we are testing it early in the phase as compared to depth first , we will get more time for changes.

In top down approach throw away code (stubs) which has to be developed is much more as there are more leaves as compared to upper level modules.

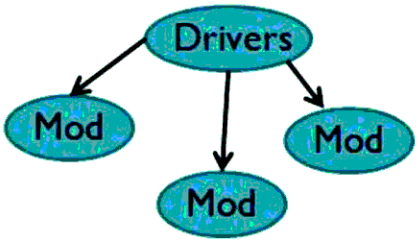


Breadth First

1) **Bottom up integration :-** Here we move from the bottom most level.

i) upper level, throw away units- Drivers are developed.

ii) Throw away units will be less as compared to Top down approach.



ANS 6) A)

- Design Process:- It is an iterative process through which requirements are translated intoa 'blueprint' for constructing the s/w.
- Initially the design is represented at a high level of abstraction & as iteration occurs subsequent refinements leads to design representations at much lower levels of abstraction.
- **Cohesion:-** degree to which component or class is related to itself.

a) It is 'single mindedness' of a component.

b) It implies that component or class encapsulates only attributes & operations that refer to itself.

Types:

1) Functional :- this level of cohesion occurs when a component performs a targeted computation and then returns a result.

2) Layer :- This type of cohesion occurs when a higher layer accesses the services of a lower layer but lower layers do not access higher layers.

3) Communicational :- All operations that access the same data are defined within one class.

4) Coincidentally cohesive :- the modules in which the set of tasks are loosely related to each other.

5)Logically cohesive :- A module that holds those tasks that are logically related to each other.

6) Temporal cohesion :- The module in which the tasks need to be executed in some specific time span.

7) Procedural cohesion :- When processing elements of a module are related with one another and must be executed in some specific order then such module is said to be procedural cohesive.

- **We always try that cohesion should be as high as possible**

Coupling :- is a qualitative measure of the degree to which classes are connected to one another. As classes (and components) become more interdependent, coupling increases.

- We always try that coupling should be as low as possible.

Types

1) Content coupling :- occurs when one component modifies data that is internal to another component. This violates data hiding principle.

2) Common coupling :- occurs when a no of components make use of a Global variable. Although it is sometimes necessary but it may lead to uncontrolled error propagation and unforeseen side effects when changes are made.

3) Control coupling :- occurs when operation A( ) passes control flag to operation B( ) which then directs logical flow within B

4) Stamp coupling :- occurs when class B is declared as one of the argument for the operation of class A. This complicates the situation.

5) Data coupling :- occurs when operations pass long strings of data arguments. The bandwidth of communication increases & complexity of the interface increases.

6) Routine call coupling :- occurs when one operation invokes other. It is very common and is also necessary.

7) Inclusion or import coupling :- occurs when compnent A, imports or includes a package or the content of component B.

8) External coupling :- occurs when a component communicates or collaborates with infrastructure components. (eg. OS functions, database, telecommunication functions)

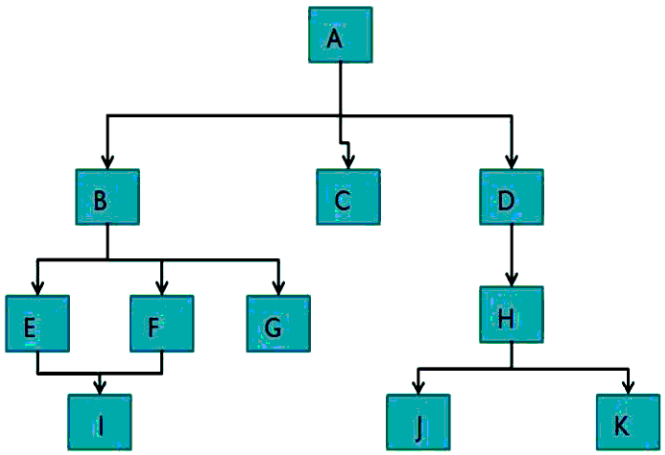
Refactoring :- is the process of changing a s/w system in such a way that it does not alter the external behavior of the code/design yet improves its internal structure.

Here the existing design is examined for redundancy, unused design elements, inefficient or unnecessary algorithms, poorly constructed or inappropriate data structures or any other design failures that can be corrected to yield a better design.

Eg.

1st design iteration yields a component that exhibits low cohesion (it performs two functionality that have limited relationship with each other)

So, in 2nd iteration we can refactor it into two separate components possessing high cohesion.



Fan out :- is a measure of the no of modules that are directly controlled by this module.

Eg. Fan out of "A" is 3.

Fan in :- indicates how many modules directly control this given module.

Eg. Fan in of "I" is 2.

ANS 6) B) Creating a Work Breakdown Structure

By definition, **projects are comprehensive tasks far removed from any routine**, which always entail a risk and are also squeezed into a tight schedule and cost corset. As a project manager, you receive a project order that contains the title, goals, start and end dates and the specified costs of the project. Nothing more. And often not even that much, but just the instruction: „Do it!"

For a project manager, **the first big challenge is to get an overview of the overall task**. What exactly is to be done in order to achieve the goals? When exactly must what be done to meet the deadlines? Where are the priorities? And which experts do I need as support? Answering these questions is supported by **the project management method „Work Breakdown Structure".**

The Work Breakdown Structure (WBS) is the central planning, communication and controlling instrument in projects.

The Work Breakdown Structure in the Start Phase

In the start phase, the Work Breakdown Structure divides each project into planable and controllable subtasks (=work packages) and thus contains both the contents on a meta level, the time delimitation of the individual work packages by start and end dates, and the responsibilities for the work packages.

The Work Breakdown Structure in the Implementation Phase

In the implementation phase and in project controlling, the work breakdown structure is used as the basis for **recording the progress of the work packages step by step** and for identifying problems, pending decisions and any delays in deadlines.

Structuring of the project into work packages

By **creating a Work Breakdown Structure (WBS)**, any project, no matter how complex, can be made tangible, structured and processed in a manageable way. By **breaking down the overall project** and distributing the work packages within the team, the project manager is relieved and can concentrate on managing the project. The Work Breakdown Structure creates the necessary **structure and orientation** as well as clarity about the respective responsibilities for the technical experts involved.

Overview in the Project Using the Work Breakdown Structure

As a central document, the Work Breakdown Structure also ensures a **uniform view of the project and its objectives** by all participants, as well as a common language. Each employee receives an overview of his or her own tasks on the one hand and the tasks of the others on the other, and is thus aware of **the dependencies between the individual work packages**. The project team members are aware of who has to deliver which results and when.

Developing a Work Breakdown Structure

Ideally, **the Work Breakdown Structure is drawn up** in the form of a workshop with the participation of a well-chosen group of participants **during the project start phase**. In complex projects with many participants, a small number of relevant employees are selected. In order to avoid lengthy and inefficient discussions, the number of participants should not exceed eight to ten.

5 steps to the Work Breakdown Structure in detail

1) List of all tasks

The first step in creating a Work Breakdown Structure is **a complete list of all tasks** to be performed within the project in the form of work packages. This should not be done by one person alone (e.g. the project manager) in a quiet room, but in a team. In practice, the brainstorming or mind mapping method is suitable for this purpose.

2) The tasks clusters

The defined tasks are **clustered according to subject areas or time schedule**. The best method for sorting depends on the project content and must be defined on a case-by-case basis.

3) Define work packages

Following clustering, **the identified tasks are summarised in work packages**. From the outset, it should be clear which granularity you want to use in order not to get lost in details that have not been lost in this meta-plan. The Work Breakdown Structure can be detailed, but if so, **the same level of detail should be found in all project phases**.

4) Assignment of responsibilities to the work packages And Define start and end dates of work packages

If the work packages are defined in the form of headings and in their place in the hierarchy, it's time to get down to business: Who does what? **The assignment of responsibilities** to the work packages takes place in the team with the technical experts. Each person in charge must make **a commitment to his or her task**. And above all, they must have the necessary time and know-how. Otherwise, the nomination of another employee must be considered.

Once the responsibilities have been determined, **the work packages are timed by defining the start and end dates**. It is important to consider where the priorities lie and which work packages are interdependent. Which activities must take place one after the other, which can be parallelized and which are perhaps not so important and can therefore be postponed?

5) Documentation of the created Work Breakdown Structure and assignment of unique work package numbers

The last step is the **documentation of the Work Breakdown Structure** created. In the course of this, each subtask also receives a coding – the **work package number**. This ensures that there is a fixed place in the Work Breakdown Structure and that the work packages are clearly identified.

The completed Work Breakdown Structure

The finished Work Breakdown Structure is then stored in a central location and distributed to the project team members. A **cross-check with the project goals** is helpful for **quality assurance of the content** of the project structure plan. Are all goals achieved when all work packages have been completed? If so, the **claim to completeness** is given. A subsequent inclusion of work packages is of course always possible, but care should be taken right from the start not to overlook anything.

ANS 7) A)

5 main measures of Software Quality -
Correctness, Maintainability, Integrity, Usability, DRE

1) Correctness : is the degree to which the software performs its required function.

The most common measure for correctness is defects per KLOC, where a defect is defined as a verified lack of conformance to requirements.

2) Maintainability : is the ease with which a program can be -

Corrected if an error is encountered,

Adapted if its environment changes, or

Enhanced if the customer desires a change in requirements.

We measure Maintainability using indirect measures.

For this we use **Time oriented metric** namely MTTC – Mean Time to Change

MTTC - The time it takes to analyze the change request, design an appropriate modification, implement the change, test it, and distribute the change to all users.

Programs are said to be more maintainable if they have lower MTTC

Another kind of metric for measuring Maintainability is to use a **cost-oriented metric** namely spoilage

Spoilage :- the cost to correct defects encountered after the software has been released to its end-users.

When the ratio of spoilage to overall project cost (for many projects) is plotted as a function of time, a manager can determine whether the overall maintainability of software produced by a software development organization is improving or not. Actions can then be taken in response to the insight gained from this information.

3) Integrity : This attribute measures a system's ability to withstand attacks (both accidental and intentional) to its security. Attacks can be made on all three components of software namely : programs, data, and documents.

To measure integrity, two additional attributes must be defined: threat and security.

Threat is the probability that an attack of a specific type will occur within a given time.

Security is the probability that the attack of a specific type will be repelled.

The integrity of a system can then be defined as

integrity = summation [(1 – threat) * (1 – security)]

where threat and security are summed over each type of attack.

4) Usability : user-friendliness

It can be measured in terms of four characteristics:

(1) the physical and or intellectual skill required to learn the system,

(2) the time required to become moderately efficient in the use of the system,

(3) the net increase in productivity measured when the system is used by someone who is moderately efficient, and

(4) a subjective assessment (sometimes obtained through a questionnaire) of users attitudes toward the system.

5) Defect Removal Efficiency (DRE) : - is a measure of the filtering ability of quality assurance and control activities as they are applied throughout all process framework activities.

When considered for a project as a whole, DRE is defined in the following manner:

DRE = $E / (E + D)$

where E is the number of errors found before delivery of the software to the end-user and D is the number of defects found after delivery.

The ideal value for DRE should be 1(One).

That is, when D is 0.

DRE will approach to 1 if E >> D

DRE can also be used within the project to assess a team's ability to find errors before they are passed to the next framework activity or software engineering task.

DREi = Ei / (Ei + Ei+1)

where Ei is the number of errors found during software engineering activity i and Ei+1 is the number of errors found during software engineering activity i+1 that are those errors that were not discovered in software engineering activity i.

A quality objective for a software team is to achieve DREi that approaches 1. That is, errors should be filtered out before they are passed on to the next activity.

ANS 7) B) **Project Manager responsibilities on software quality**

Most project management responsibilities are defined in procedures and work instructions; the project manager is the person in-charge of making sure that all the team members comply with the said procedures and instructions.

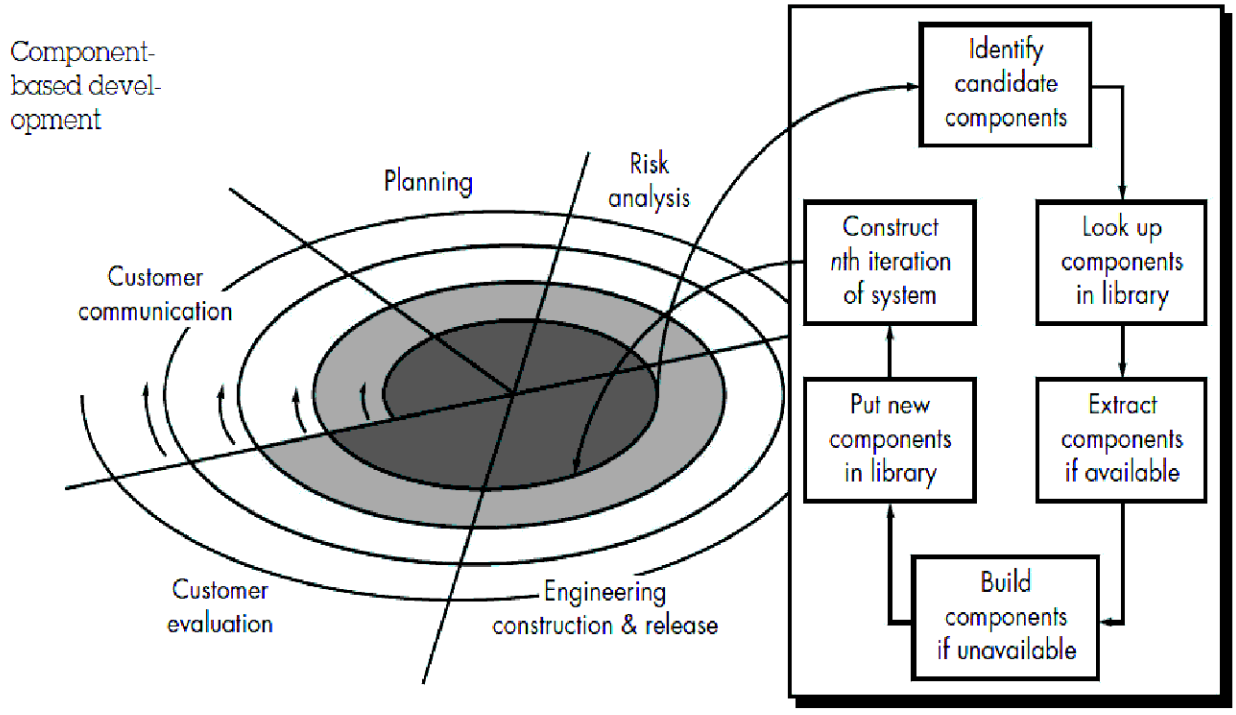
His tasks include professional hands-on and managerial tasks, particularly the following −

a) Professional hands-on tasks
b) Preparation of project and quality plans and their updates
c) Participation in joint customer–supplier committee
d) Close follow-up of project team staffing, including attending to recruitment, training and instruction
e) Management tasks

Project managers address the follow-up issues such as −

a) Performance of review activities and the consequent corrections
b) Software development and maintenance unit's performance, integration and system test activities as well as corrections and regression tests
c) Performance of acceptance tests
d) Software installation in remote customer sites and the execution of the software system by the customer
e) SQA training and instruction of project team members
f) Schedules and resources allocated to project activities
g) Customer requests and satisfaction
h) Evolving project development risks, application of solutions and control of results

ANS 8) A) **Component Based Development/CBD Model – leads to s/w reuse.**



ANS 8) B) **Risk Assessment & Mitigation**

Risk Assessment :

How to access overall project risk ?

The best approach is to prepare a set of questions that can be answered by project managers in order to asses the overall project risks. These questions can be

1. Will the project get proper support by the customer manager ?
2. Are the end-users committed to the software that has been produced ?
3. Is there a clear understanding of requirements ?
4. Is project scope stable ?
5. Are there team members with required skills ?
6. Are project requirements stable ?

Risk Assessment Impact : Three factors are considered

1) **Nature of risk :** denotes the type or kind of risk
2) **Scope of the risk :** means severity of the risk
3) **Timing :** of risk means determining at which phase of SDLC the risk will occur and how long it will persist.

Risk Exposure : can be calculated by following formula

Risk exposure = probability of occurrence of risk * cost

EX-

Consider a software project with 77 percent of risk Probability in which 15 components were developed from the scratch. Each component have on an average 500 LOC and each LOC have an average cost of $10. Then risk exposure can be calculated as,

=> No of components * LOC * cost of each LOC

=> 15 * 500 * 10 = $75000

Then risk exposure = probability of occurrence of risk *cost

=(77 / 100) * 75000

= $ 57750

Risk Mitigation : means preventing the risks to occur (risk avoidance)Following are the steps to be taken for mitigating the risks :

1. Communicate with the concerned staff to find probable risk

2. Find out and eliminate all those causes that can create risk before the project starts.

3. Develop a policy in an organization which will help to continue the project even though some staff leaves the organization.

4. Everybody in the project team should be acquainted with the current development activity.

5. Maintain the corresponding documents in timely manner. This documentation should be strictly as per the standards set by the organization.

6. Conduct timely reviews in order to speed up the work.

7. For conducting every critical activity during software development, provide the additional staff if required.