MAHARASHTRA STATE BOARD OF TECHNICAL EDUCATION
(Autonomous)
(ISO/IEC – 27001 – 2005 Certified)
WINTER – 2012 EXAMINATION

Subject Code:12175 **Model Answer** **Page No.: 1/26**

Name of Course & Year/Sem.:- Computer Engineering Vth Semester

Q 1 a) Attempt Any Three of The Following.

i. Explain Changing nature of Software.

**(Any 8 points regarding Changing natures are expected* 1/2 mark for each points*)*

Ans-> Changing Nature of Software: -

Whenever one starts with the software implementation changes can occur any time. The software can be change due to any reason. But while implementing software one should be ready for such changes as if changes occur there shall not be drastic change in the system. The development team should manage to implement/ mould the implemented system so that the changes can be reflected and the user requirements meet. When change occur the team look for the current status of the system and from there onwards they starts implementing a system with new requirements of a user or changes which is to be implemented in a system.

ii. What are software Risks?

**(Any 8 points regarding Software Risks are expected* 1/2 mark for each point*)*

Ans-> Software Risk:- A software risk is anything which can cause a delay in a software or stops the progress of a system or even terminates the software project.

There are two basic types of risks:

a. Generic Risk

Generic Risk are the general purpose possible threat to every software product.

b. Product Specific Risk

Product Specific risk can be find out only by those with a clear understanding of the technology going to be used for that project, the people and the environment that is particular to the software that is to be built.

Different Categories of Risks: -

a. **Project Risk**: - Threaten the project plan. That is if project risk become real it is likely that project schedule will slip and that costs will increase. Project risks identity potential budgetary, schedule, personnel, resource, customer and requirement problem.
b. **Technical Risk**: - Threaten the quality and timeliness of the software to be produced. If technical risk becomes real, implementation may become difficult or impossible.
c. **Business Risk**: - Threaten the viability of the software to be built. Business risk often jeopardizes the product or the project.
d. **Market Risk**:- Building product that no one wants
e. **Strategic Risk**: - Building a product that no longer fits into the business strategy

f. **Management Risk**: - Building a product that the sale force doesn't understand.
g. **Budget Risk**: - Losing budgetary or personnel commitment

iii. What is quality control

Ans-> *(Any 8 points regarding quality control is expected 1/2 mark for each point)

The Quality control involves the series of inspections, reviews and tests used throughout the software process to ensure each work product meets the requirement placed upon it. Quality control includes a feedback loop to the process that created the work product. The combination of measurement and feedback allows us to tune the process when the work products created fail to meet their specifications.

A key concept of quality control is that all work products have defined, measureable specification to which we may compare the output of each process. The feedback loop is essential to minimize the defects produced.

iv. Differentiate between Validation and Verification

Ans-> *(Any four points of differentiation are expected 1 mark each)

Validation checks that the product design satisfies or fits the intended usage (high-level checking), i.e., the software meets the user requirement. This is done through dynamic testing and other forms of review.

Verification and validation are not the same thing, although they are often confused. Boehm (Boehm, 1979) succinctly expressed the difference between them:

- Validation: Are we building the right product?
- Verification: Are we building the product right?
- Verification: The process of evaluating software to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase.

Verification is the process of determining that a computer model, simulation, or federation of models and simulations implementations and their associated data accurately represents the developer's conceptual description and specifications.

Validation is the process of determining the degree to which a model, simulation, or federation of models and simulations, and their associated data are accurate representations of the real world from the perspective of the intended use(s)

Validation: The process of evaluating software during or at the end of the development process to determine whether it satisfies specified requirements.

Q 1 b) Attempt Any One Of The following.

i. Describe Design Modeling Principles

**(Any 6 Design Modeling Principles are expected with description 1 mark each)*

Ans-> Following are various Design Modeling Principles

Principle #1:- Design should be traceable to an analysis model

The analysis mode describes the information domain of the problem, user visible functions and various things about system. The design model translates this information into architecture: a set of sub systems that implement major functions, and a set of component level designs that are the realization of analysis classes.

Principle #2: - Always consider the architecture of the system to be built

Software architecture is a skeleton of the system to be built. It affects interfaces, data structures, program control flow and behavior, the manner in which testing can be conducted, the maintainability of the resultant system and much more.

Principle #3: - Design of data is as important as design of processing functions

Data design is an essential element of architectural design. The manner in which data objects are realized within the design cannot be left to chance. A well structured data design helps to simplify program flow, makes the design and implementation of software components easier, and makes overall processing more efficient.

Principle #4: - Interfaces must be designed with care

The manner in which data flows between the components of a system has much to do with processing efficiency, error propagation, and design simplicity. A well designed interface makes integration easier and assists the tester in validating component functions.

Principle #5: - User interface design should be tuned to the needs of the end-user

In every case it should stress ease of user. The user interface is the visible manifestation of the software. No matter how sophisticated its internal functions, no matter how comprehensive its data structures, no matter how well designed its architecture, a poor interface design often leads to the perception that the software is bad.

Principle #6: - Component level design should be functionally independent.
Functional independence is a measure of the "single mindedness" of a software component. The functionality that is delivered by a component should be cohesive – that is it should focus on one and only one function or sub-function.

Principle #7: - Component should be loosely coupled to one another and to the external environment.
Coupling is achieved in many ways – via a component interface, by messaging, through global data. As the level of coupling increases, the likelihood or error propagation also increases and the overall maintainability of the software decreases.

Principle #8: - Design representation should be easily understandable
The purpose of design is to communicate information to practitioners who will generate code, to those who will test the software, and to others who may maintain the software in the future. If the design is difficult to understand, it will not serve as an effective communication medium.

Principle #9 The design should be developed iteratively. With each iteration the designer should strive for greater simplicity.
Like almost all creative activities, design occurs iteratively. The first iterations works to refine the design and correct errors, but later iterations should strive to make the design as simple as possible.

ii. What are the core principles of Software Engineering?
(All 7 principles expected with 1-2 liner description of each; 2 marks for only list of all core principles)

Ans-> Following are the various Core Principles of Software Engineering.

1. **Reason it all Exist**
A software system exists for one reason; to provide value to its users. All decisions should be made with this in mind. Before specifying a system requirement before nothing a piece of system functionality, before determining the hardware or platform development processes, one should identity the reason for which system is being build for.

2. **Keep it Simple and Stupid**
Software design is not haphazard process. There are many factors to consider in any design efforts. All design should be as simple as possible, but no simpler. This facilitates having a more easily understood, and easily maintained system. This is not to say that features, even internal features, should be discarded in the name of simplicity.

3. **Maintain The Vision**
   A Clear vision is essential to the success of a software project. Without that A project almost unfailingly ends up being "of two minds" about itself. Without conceptual integrity, a system threatens to become a patchwork of incompatible designs, held together.

4. **What You Produce, Others Will Consume**
   Seldom is an industrial strength software system constructed and used in vacuum. In some way or other, someone else will use, maintain, document or otherwise depend on being able to understand your system. So always specify design and implement knowing someone else will have to understand what you are doing.

5. **Be Open To The Future**
   While implementing a system a developer of a system shall always consider for the scope of improvement in a given system, by virtue of which there will be continuous upgrading in a given system and the system can be timely reviewed. As a result of this one can have better and improved system at the end of all iterations.

6. **Plan Ahead For Reuse**
   While implementing a system the development team can think about having a module which are already developed in a similar kind of system, resulting in a less time of development and as these modules are reused very little time can be spend on the testing of such module. The fact is few testing strategies like unit testing can be avoided as more often than not such modules are already working in a desired manner.

7. **Think**
   During the implementation phase; one shall think about the implementation of the system. More you think one a system better system can be implemented as better and improved ideas can be evolve. By thinking on a system in broader way it give a broader scope of a system.

Q 2. Attempt Any Four of the following

a. Describe Data Object and Data Attribute

**(2 Marks for each Data object & 2 Marks Data attribute)*

Ans-> **Data Object: -**
A data object is a representation of almost any composite information that must be understood by software. By composite information, we mean something that has a number of different properties or attributes.

A Data object can be an external entity, a thing, or an occurrence.
Data object are related to one another.
A data object encapsulates data only – there are no reference within a data object to operation that act on the data.

Data Attribute: -
Data object is having a ser of attributes. Data attributes defines the properties of data object and can have following characteristics.

1. Name an instance of the data object
2. Describe the instance
3. Make reference to another instance in another table.

One or more of the attributes must be defined as an identifier that is the identifier attributes becomes a key when one want to find an instances of the data object.
The set of attributes that is appropriate for a given data object is determined through an understanding of the problem

b. Explain White Box Testing
**(3 marks for White box testing basics; combined 1 marks for four points given below)*

Ans-> **White Box Testing: -**
White box testing is also called as glass testing, is a test case design philosophy that uses the control structure described as part of component level design to derive test cases. Using white box testing methods, the software engineer can derive test cases that:

1. Guarantee that all independent paths within a module have been exercised at least once.
2. Exercise all logical decisions on their true and false sides.
3. Execute all loops at their boundaries and within their operational bound
4. Exercise internal data structure to ensure their validity.

White box testing usually deals with the programming module and tries to verify the validity of the statement. All syntax level error can be found out with this Testing.

c. Describe the Tasks of Requirement Engineering
**(All 7 tasks expected with 1-2 liner description of each; 2 marks for only list of all)*

Ans -> Requirements engineering tasks

1. **Inception** : -
Inception means beginning. It is usually said that requirement engineering is a “communication intensive activity”. The customer and developer meet and they

decide the overall scope and nature of the problem statements. By having proper inception phase the developer will have clear idea about the system and as a result of that better understanding of a system can be achieved. Once the system is clear to the developer they can implement a system with better efficiency.

2. **Elicitation**: -Elicitation task will help the customer to define the actual requirement of a system. To know the objectives of the system or the project to be developed is a critical job. This phase will help people to determine the goal of a system and clear idea about the system can be achieved.

3. **Elaboration**:-
The information obtained from the customer during inception and elicitation is expanded and refined during elaboration. This requirement engineering activity focuses on developing a refined technical model of software functions, features and constraints.

4. **Negotiation**: -
This phase will involve the negotiation between what user actual expects from the system and what is actual feasible for the developer to build. Often it is seen that user always expect lot of things from the system for lesser cost. But based of the other aspect and feasibility of a system the customer and developer can negotiate on the few key aspect of the system and then they can proceed towards the implementation of a system

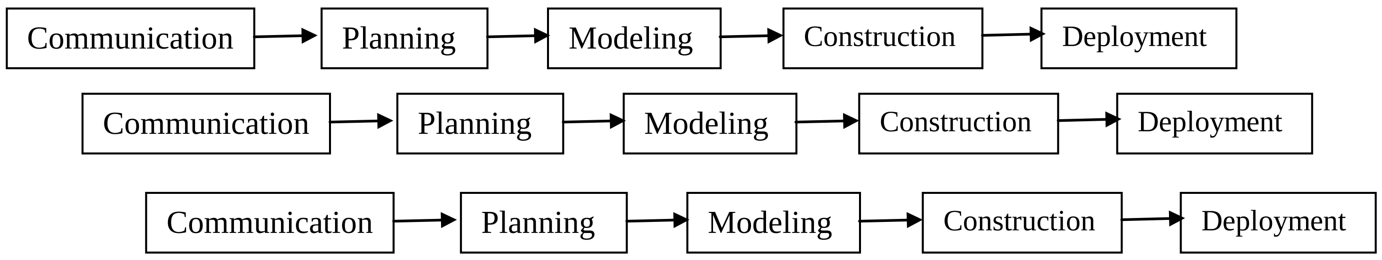
5. Specification
A specification can be a re-written document, a set of graphical models, a formal mathematical model, a collection of usage scenario, a prototype, or any combinations of these.
The specification is the final work product produced by the requirement engineers. It serves as the foundation for subsequent software engineering activities. It describes the function and performance of a computer based system and the constraints that will govern its development.

6. Validation
The work products produced as a consequence of requirements engineering are assessed for quality during a validation step. Requirements validation examines the specification to ensure that all software requirements have been stated unambiguously; that inconsistencies, omissions and errors have been detected and corrected, and that the work products conform to the standards established for the process, the project, and the product.

7. **Requirements management**
Requirement management begins with identification. Each requirement is assigned a unique identifier. Once requirement have been identified, traceability tables are developed.

d. Explain Incremental Model with Diagram
(1 mark for basic 1 mark for Diagram 2 marks for Description of all phases)

Ans-> **Incremental Model**: -The problem with Waterfall model is that it doesn't support back-tracking i.e. if a mistake takes place in any of the phase of Water fall model, it will be reflected in all other remaining phases. Hence to overcome this drawback Incremental Model comes into picture.
Basic Incremental Model has following Five Stages: -
Communication: - This phase will involve all basic communication with the customer so that developer of a system will try to understand basic requirement of a customer and to understand the scope of a system.
Planning: - This Phase will involve all the planning and strategically activities to be performed by the development team, this phase will also makes sure proper planning for the successful completion of the project is to be carried out
Modeling: - This phase will involve all the basic modeling of a system is conducted viz Analysis and Design modeling is carried out in this phase. After completion of this phase the development team will have complete analysis of a system along with the all required design structure of a system.
Construction: - This phase will involve the construction and Testing of a system. The developer of a system will develop an application using suitable technology and once done these modules will be forwarded to the testing team. The Testing team then will test an application to ensure the quality of a system. Once application is approved by QA team it is delivered to customer end.
Deployment: - This phase involves the Deployment of a system. i.e. installing a system at the customer end. At times this phase also involves a training session to the customer for their understanding.



e. Explain Generic characteristics of Software Testing

Ans->

(All Four characteritics.1 mark for each characteristic)

- To perform effective testing, a software team should conduct effective formal technical reviews. By doing this many errors will be eliminated before testing commences.
- Testing begins at the component level and works outwards towards the integration of the entire computer based system.
  Different testing techniques are appropriate at different points in time.
- Testing is conducted by the developer of the software and an independent test group.
- Testing and Debugging are different activities, but debugging must be accommodated in any testing strategy.

f. Describe "ISO 9000" Quality Standards

Ans-> *(Any 4 points for ISO 9000 are expected 1 mark for each point)*

A quality assurance system may be defined as the organizational structure, responsibilities, procedures, processes and resources for implementing quality management. Quality assurance systems are created to help organizations ensure their products and services satisfy customer expectations by meeting their specifications. ISO 9000 describes a quality assurance system in generic terms that can be applied to any business regardless of the products or services offered.

To become registered to one of the quality assurance system models contained in ISO 9000 a company's quality system and operations are scrutinized by third-party auditors for compliance to the standard and for effective operation. Upon successful registrations, a company is issued a certificate for a registration body represented by the auditors. Semiannual, surveillance audits ensure continued compliance to the standard.

Q3. Attempt any four of the following

a) What is software project tracking and control? Enlist activities of it.

Ans->*(2 marks for Software project tracking and control. 2 Marks for Activity list)*

Project tracking can be done using scheduling tools:-

Program evaluation and review technique (**PERT**) and critical path method (**CPM**) are two project scheduling methods that can be applied to software development. Both techniques are driven by information already developed in earlier project planning activities:

• Estimates of effort
• A decomposition of the product function
• The selection of the appropriate process model and task set
• Decomposition of task

Timeline (Gantt) charts enable software planners to determine what tasks will be need to be conducted at a given point in time (based on estimates for effort, start time, and duration for each task).
Software project scheduling activities:
1. Compartmentalization.
2. Interdependency.
3. Time allocation.
4. Effort validation
5. Defined responsibilities.
6. Defined outcomes.
7. Defined milestones.

b) Explain software Reliability and Availability.
Ans-> *(2 marks for Reliability and 2 marks for Availability)*

Reliability:- software Reliability is probability of failure free operation of a system. It is calculated as follows

MTBF = MTTF + MTTR

The acronyms MTTF and MTTR are mean-time-to-failure and mean-time-to-repair

MTBF= Mean Time Between Failure

Availability: -Software availability is the probability that a program is operating according to requirements at a given point in time and is defined as

Availability = [MTTF/(MTTF + MTTR)] X 100%

The MTBF reliability measure is equally sensitive to MTTF and MTTR. TThe availability measure is somewhat more sensitive to MTTR, an indirect measure of the maintainability of software.

c) Explain Change management
Ans-> (Four points.1 mark for each point)
Changes are inevitable when software is built. Configuration management is all about change control. Every software engineer has to be concerned with how changes made to work products are tracked and propagated throughout a project.

Software Configuration Management Tasks

•Identification (tracking multiple versions to enable efficient changes)
•Version control (control changes before and after release to customer)
•Change control (authority to approve and prioritize changes)
•Configuration auditing (ensure changes made properly)

Identification

To control and manage configuration items, each must be named and managed using an object-oriented approach
•Basic objects are created by software engineers during analysis, design, coding, or testing
• Aggregate objects are collections of basic objects and other aggregate objects
• An entity-relationship (E-R) diagram can be used to show the interrelationships among the objects

Version Control

•Combines procedures and tools to manage the different versions of configuration objects created during the software process
• An entity is composed of objects at the same revision level
• A variant is a different set of objects at the same revision level and coexists with other variants
• A new version is defined when major changes have been made to one or more objects

Change Control

Change request is submitted and evaluated to assess technical merit and impact on the other configuration objects and budget
•Change report contains the results of the evaluation
•Change control authority (CCA) makes the final decision on the status and priority of the change based on the change report

Software Configuration Audit Questions

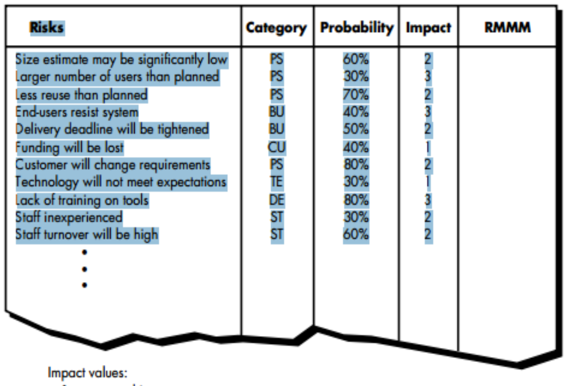
•Has the change specified by the ECO been made without modifications?
•Has an FTR been conducted to assess technical correctness?
•Was the software process followed and software engineering standards applied?

d) What is Risk Projection? Enlist Steps of Risk Projection.

Ans-> (2 marks for definition and 2 marks for enlisting the activities)

Definition - Risk projection, also called risk estimation, attempts to rate each risk in two ways—the likelihood or probability that the risk is real and the consequences of the problems associated with the risk, should it occur.

Steps of Risk Projection (Estimation)
• Establish a scale that reflects the perceived likelihood of each risk
•Delineate the consequences of the risk
•Estimate the impact of the risk on the project and product
• Note the overall accuracy of the risk projection to avoid misunderstandings



Risks	Category	Probability	Impact	RMMM
Size estimate may be significantly low	PS	60%	2	
Larger number of users than planned	PS	30%	3	
Less reuse than planned	PS	70%	2	
End-users resist system	BU	40%	3	
Delivery deadline will be tightened	BU	50%	2	
Funding will be lost	CU	40%	1	
Customer will change requirements	PS	80%	2	
Technology will not meet expectations	TE	30%	1	
Lack of training on tools	DE	80%	3	
Staff inexperienced	ST	30%	2	
Staff turnover will be high	ST	60%	2	
• • •				

Impact values:
1—catastrophic

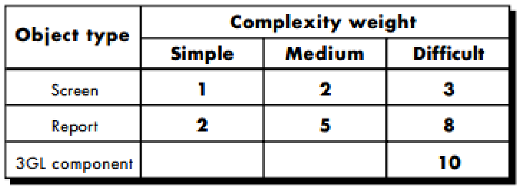
2.critical 3, marginal 4 negligible

e) Describe COCOMO and COCOMO II Models
Ans-> *(2 marks for COCOMO and 2 marks for COCOMO II model description)*

Basic COCOMO computes software development effort (and cost) as a function of program size. Program size is expressed in estimated thousands of source lines of code (SLOC). The basic COCOMO equations take the form
Effort Applied (E) = $a_b(KLOC)^{b}{}_{b}$ [man-months]
Development Time (D) = $c_b(\text{Effort Applied})^{d}{}_{b}$ [months]
People required (P) = Effort Applied / Development Time [count]
Basic COCOMO is good for quick estimate of software costs. However it does not account for differences in hardware constraints, personnel quality and experience, use of modern tools and techniques, and so on.

<u>COCOMO II</u>

Like all estimation models for software, the COCOMO II models require sizing information. Three different sizing options are available as part of the model hierarchy:object points, function points, and lines of source code.The COCOMO II application composition model uses object points and is illustrated in the following paragraphs.



Object type	Complexity weight		
	Simple	Medium	Difficult
Screen	1	2	3
Report	2	5	8
3GL component			10

NOP = (object points) x [(100 ———————————— %reuse)/100]
where NOP is defined as new object points.
To derive an estimate of effort based on the computed NOP value, a "productivityrate" must be derived.

PROD = NOP/person-month
estimated effort = NOP/PROD

Q4. a) Attempt Ant Three of The Following

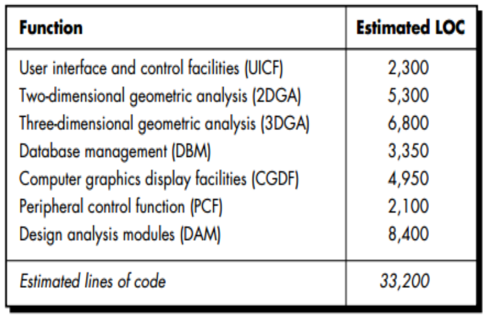
i. Explain Software Decomposition Techniques.

Ans-> *(1 mark for definition ,3 techniques- 1 mark each for each technique)*
Definition-
•Software project estimation is a form of problem solving, and in most cases, we decompose the problem, recharacterizing it as a set of smaller (and hopefully, more manageable) problems.
The decomposition approach was discussed from two different points of view: decomposition of the problem and decomposition of the process.

- Software sizing (fuzzy logic, function point, standard component, change)

Problem-based estimation (using LOC decomposition focuses on software functions, using FP decomposition focuses on information domain characteristics)

FIGURE 5.3 Estimation table for the LOC method



Function	Estimated LOC
User interface and control facilities (UICF)	2,300
Two-dimensional geometric analysis (2DGA)	5,300
Three-dimensional geometric analysis (3DGA)	6,800
Database management (DBM)	3,350
Computer graphics display facilities (CGDF)	4,950
Peripheral control function (PCF)	2,100
Design analysis modules (DAM)	8,400
Estimated lines of code	*33,200*

•Process-based estimation (decomposition based on tasks required to complete the software process framework

ii. Describe Integration Testing Approaches.

Ans-> *(2 Techniques Top-down and Bottom-up ; 2 marks for each technique)*
Definition- Integration testing, also known as integration and testing (I&T), is a software development process which program units are combined and tested as groups in multiple ways. In this context, a unit is defined as the smallest testable part of an application. Integration testing can expose problems with the interfaces among program components before trouble occurs in real-world program execution.

Integration Testing Strategies are
•Top-down integration testing
1. Main control module used as a test driver and stubs are substitutes for components directly subordinate to it.
2. Subordinate stubs are replaced one at a time with real components (following the depth-first or breadth-first approach).
3. Tests are conducted as each component is integrated.
4. On completion of each set of tests and other stub is replaced with a real component.
5. Regression testing may be used to ensure that new errors not introduced.

•Bottom-up integration testing
1. Low level components are combined in clusters that perform a specific software function.
2. A driver (control program) is written to coordinate test case input and output.
3. The cluster is tested.
4. Drivers are removed and clusters are combined moving upward in the program structure.

iii. Describe Communication Principles Statements.

Ans-> **(1-2 liner Description of any Four principles are expected out of following. 1mark for each principle)*

1) Listen to the speaker and concentrate on what is being said
2) Prepare before you meet by researching and understanding the problem
3) Someone should facility the meeting and have an agenda
4) Face-to-face communication is best, but also have a document or presentation to focus the discussion
5) Take notes and document decisions
6) Strive for collaboration and consensus
7) Stay focused on a topic; modularize your discussion
8) If something is unclear, draw a picture
9) Move on to the next topic a) after you agree to something, b) if you cannot agree to something, or c) if a feature or function is unclear and cannot be clarified at the moment
10) Negotiation is not a contest or a game; it works best when both parties win

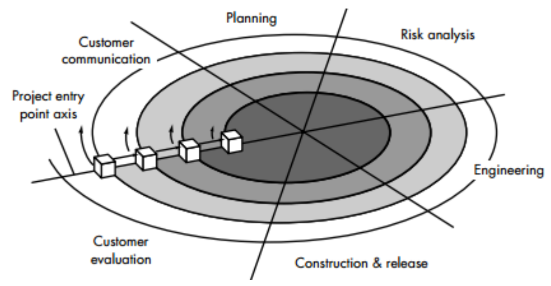
iv. Explain Brute Force Approach used as debugging Strategies.
Ans-> *(2 marks for debugging concept and 2 marks for brute force*)

Debugging is not testing but always occurs as a consequence of testing.
Results are assessed and a lack of correspondence between expected and actual performance is encountered. In many cases, the non-corresponding data are a symptom of an underlying cause as yet hidden. The debugging process attempts to match symptom with cause, thereby leading to error correction.
The debugging process will always have one of two outcomes:
(1) the cause will be found and corrected, or
(2) the cause will not be found.
In the latter case, the person performing debugging may suspect a cause, design a test case to help validate that suspicion, and work toward error correction in an iterative fashion.

The Brute Force
The brute force category of debugging is probably the most common and least efficient method for isolating the cause of a software error. We apply brute force debugging methods when all else fails. Using a "let the computer find the error" philosophy, memory dumps are taken, run-time traces are invoked, and the program is loaded with WRITE statements. We hope that somewhere in the morass of information that is produced we will find a clue that can lead us to the cause of an error. Although the mass of information produced may ultimately lead to success, it more frequently leads to wasted effort and time.

Q4. b) Attempt Any One of The Following
I. Explain different tasks of regions of Spiral Model with diagram.
Ans-> *(2 mark for diagram , 4 marks for explanation)*

Diagram



- **Customer communication**—tasks required to establish effective communication between developer and customer.
- **Planning**—tasks required to define resources, timelines, and other project related information.

- **Risk analysis**—tasks required to assess both technical and management risks.
- **Engineering**—tasks required to build one or more representations of the application.
- **Construction and release**—tasks required to construct, test, install, and provide user support (e.g., documentation and training).
- **Customer evaluation**—tasks required to obtain customer feedback based on evaluation of the software representations created during the engineering stage and implemented during the installation stage.

II. Explain CMMI with its levels.

Ans-> *(2 marks for CMMI definition, 4 marks for explanation)*

Definition- Capability Maturity Model Integration (CMMI) is a process improvement approach that helps organizations improve their performance.

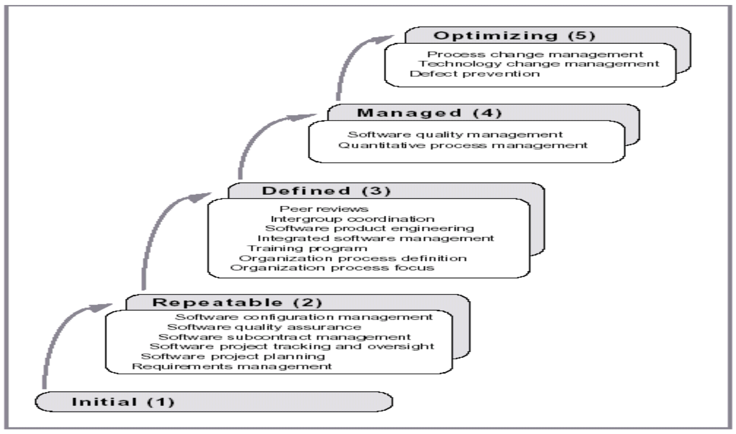

Optimizing (5)
Process change management
Technology change management
Defect prevention
Managed (4)
Software quality management
Quantitative process management
Defined (3)
Peer reviews
Intergroup coordination
Software product engineering
Integrated software management
Training program
Organization process definition
Organization process focus
Repeatable (2)
Software configuration management
Software quality assurance
Software subcontract management
Software project tracking and oversight
Software project planning
Requirements management
Initial (1)

Figure 3.2 The Key Process Areas by Maturity Level

Level 1: Initial. The software process is characterized as ad hoc and occasionally even chaotic. Few processes are defined, and success depends on individual effort.
Level 2: Repeatable. Basic project management processes are established to track cost, schedule, and functionality. The necessary process discipline is in place to repeat earlier successes on projects with similar applications.
Level 3: Defined. The software process for both management and engineering activities is documented, standardized, and integrated into an organization wide software process. All projects use a documented and approved version of the organization's process for developing and supporting software. This level includes all characteristics defined for level 2.
Level 4: Managed. Detailed measures of the software process and product quality are collected. Both the software process and products are quantitatively

understood and controlled using detailed measures. This level includes all characteristics defined for level 3.

Level 5: Optimizing. Continuous process improvement is enabled by quantitative feedback from the process and from testing innovative ideas and technologies. This level includes all characteristics defined for level 4.

Q5. Attempt any <u>TWO</u> of the following:

a). Explain Modeling Practice in Software Engineering with principles.

(Modeling practice explanation 2 marks, 3 marks for Analysis modeling and 3 marks Design Modeling)

Ans-> **Modeling practice**

One creates a model to gain a better understanding of the actual entry to be built. When the entry is physical thing, one can build a model that is identical in form and shape but smaller in scale. However, when the entity is software, model must take a different form. It must be capable of representing the information that software transforms, the architecture and functions that enable the transformation to occur, the features that the users desires, and the behavior of the system as the transformation is taking place. Models must accomplish these objectives at different level of abstraction. There are two types of Modeling as follows analysis and design

Analysis Modeling:- This represents the customer requirement by depicting the software in three different domains. The information domain, the function domain and the behavioral domain. Following are the principles of Analysis modeling.

Analysis Modeling Principles

1. The information domain of a problem must be represented and understood.
2. The functions that the software performs must be defined
3. The behavior of the software (as a consequence of external events) must be represented.
4. The models that depict information function and behavior must be partitioned in a manner that uncovers detail in a layered (or hierarchical) fashion.
5. The analysis task should move from essential information toward implementation detail.

Design Modeling Principles:- This represents characteristics of the software that help practioners to construct it effectively; the architecture, the user interface and component level details are included. . Following are the principles of Design modeling

Design Modeling Principles

1. Design should be traceable to an analysis model
2. Always consider the architecture of the system to be built
3. Design of data is as important as design of processing functions
4. Interfaces must be designed with care
5. User interface design should be tuned to the needs of the end-user
6. Component level design should be functionally independent.
7. Component should be loosely coupled to one another and to the external environment.
8. Design representation should be easily understandable

b) What is RMMM Strategy? Explain the issues involved in it.
(Description of RMMM ***3 marks****, 10 points on issues involved* ***5 marks*** *(strategies, factors also accepted)*

Ans-> Risk mitigation, monitoring, and management (RMMM) plan

A risk management strategy can be included in the software project plan or the risk management steps can be organized into a separate Risk Mitigation, Monitoring and Management Plan. The RMMM plan documents all work performed as part of risk analysis and is used by the project manager as part of the overall project plan.

Once RMMM has been documented and the project has begun, risk mitigation and monitoring steps commence. Risk mitigation is a problem avoidance activity. Risk monitoring is a project tracking activity with three primary objectives: (1) to assess whether predicted risks do, in fact, occur; (2) to ensure that risk aversion steps defined for the risk are being properly applied; and (3) to collect information that can be used for future risk analysis. In many cases, the problems that occur during a project can be traced to more than one risk. Another job of risk monitoring is to attempt to allocate origin (what risk(s) caused which problems throughout the project).

An effective strategy must consider three issues:

• Risk avoidance
• Risk monitoring
• Risk management and contingency planning

If a software team adopts a proactive approach to risk, avoidance is always the best strategy. This is achieved by developing a plan for **risk mitigation**.

To mitigate this risk, project management must develop a strategy for reducing turnover. Among the possible steps to be taken are

• Meet with current staff to determine causes for turnover (e.g., poor working conditions, low pay, and competitive job market).
• Mitigate those causes that are under our control before the project starts.
• Once the project commences, assume turnover will occur and develop techniques to ensure continuity when people leave.

- Organize project teams so that information about each development activity is widely dispersed.
- Define documentation standards and establish mechanisms to be sure that documents are developed in a timely manner.
- Conduct peer reviews of all work (so that more than one person is "up to speed”).
- Assign a backup staff member for every critical technologist.

As the project proceeds, risk monitoring activities commence. The project manager monitors factors that may provide an indication of whether the risk is becoming more or less likely. In the case of high staff turnover, the following factors can be monitored:

- General attitude of team members based on project pressures.
- The degree to which the team has jelled.
- Interpersonal relationships among team members.
- Potential problems with compensation and benefits.
- The availability of jobs within the company and outside it.

In addition to monitoring these factors, the project manager should monitor the effectiveness of risk mitigation steps. RMMM steps incur additional project cost. Part of risk management, therefore, is to evaluate when the benefits accrued by the RMMM steps are outweighed by the costs associated with implementing them. In essence, the project planner performs a classic cost/benefit analysis.

c) What are the categories of software? Explain
*(8 categories **4 marks ½ marks for each category,** explanation **4 marks (answer may differ in different edition of Ref. book)**)*

Ans-> **System software** System software is a collection of programs written to service utilities) process complex, but determinate, information structures.

Real-time software: Software that monitors/analyzes/controls real-world events as they occur is called real time.

Business software: Business information processing is the largest single software application area. Discrete "systems" (e.g., payroll, accounts receivable/payable, inventory) have evolved into management information system (MIS) software that accesses one or more large databases containing business information.

Engineering and scientific software: Engineering and scientific software have been characterized by "number crunching" algorithms. Applications range from astronomy to volcanology, from automotive stress analysis to space shuttle orbital dynamics, and from molecular biology to automated manufacturing.

Embedded software: Intelligent products have become commonplace in nearly every consumer and industrial market. Embedded software resides in read-only memory and is used to control products and systems for the consumer and industrial markets.

Personal computer software: The personal computer software market has burgeoned over the past two decades. Word processing, spreadsheets, computer graphics, multimedia, entertainment, database management, personal and business financial applications, external network, and database access are only a few of hundreds of applications.
Web-based software: The Web pages retrieved by a browser are software that incorporates executable instructions (e.g., CGI, HTML, Perl, or Java), and data.
Artificial intelligence software: Artificial intelligence (AI) software makes use of non-numerical algorithms to solve complex problems that are not amenable to computation or straightforward analysis.

Q6. Attempt any FOUR of the following:

a) Explain Unit Testing in detail.
*(Figure **1 mark**, definition **1 mark**, Unit Test Considerations **1 mark**, Unit Test Procedures **1 mark**)*

Ans->

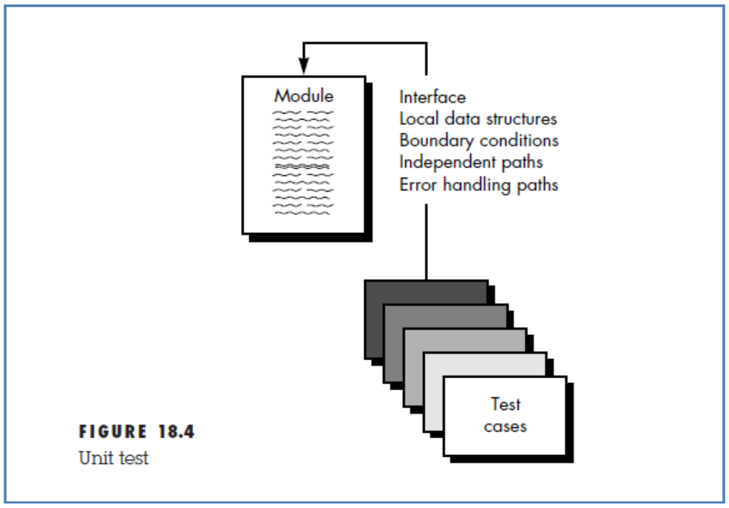
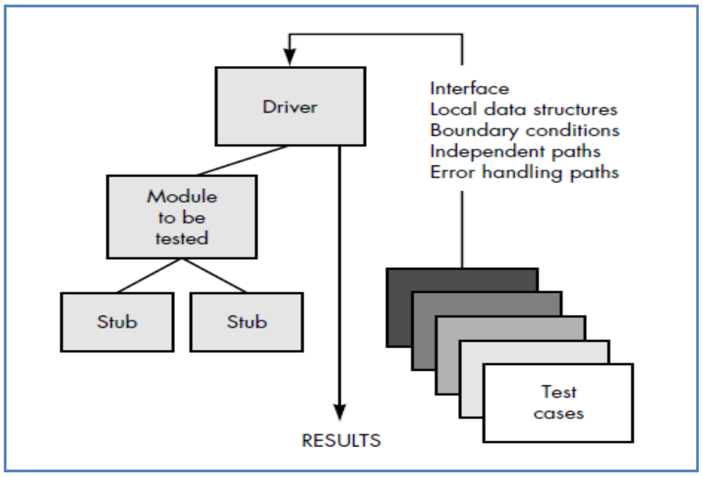


FIGURE 18.4
Unit test

OR



Unit testing focuses verification effort on the smallest unit of software design—the software component or module. Using the component-level design description as a guide, important control paths are tested to uncover errors within the boundary of the module. The relative complexity of tests and uncovered errors is limited by the constrained scope established for unit testing. The unit test is white-box oriented, and the step can be conducted in parallel for multiple components.

Unit Test Considerations

The module interface is tested to ensure that information properly flows into and out of the program unit under test. The local data structure is examined to ensure that data stored temporarily maintains its integrity during all steps in an algorithm's execution. Boundary conditions are tested to ensure that the module operates properly at boundaries established to limit or restrict processing. All independent paths (basis paths) through the control structure are exercised to ensure that all statements in a module have been executed at least once. And finally, all error handling paths are tested.

Unit Test Procedures

Unit testing is normally considered as an adjunct to the coding step. After source level code has been developed, reviewed, and verified for correspondence to component level design, unit test case design begins. A review of design information provides guidance for establishing test cases that are likely to uncover errors. Each test case should be coupled with a set of expected results.

b) Explain "Software Engineering As A Layered Technology"

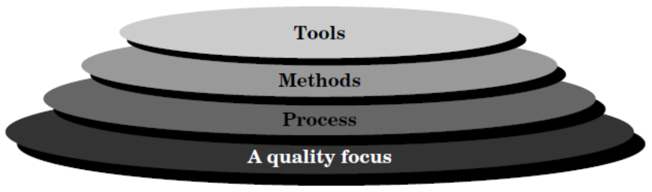
Ans-> *(Fig & defn **1 mark**, explanation 3 **marks**)*

Fritz Bauer defn serves as a basis for discussion:

[Software engineering is] the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines.

Software Engineering: (1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software. (2) The study of approaches.

FIGURE 2.1
Software engineering layers



(Software engineering) must rest on an organizational commitment to quality. The bedrock that supports software engineering is a quality focus.

The foundation for software engineering is the process layer. Software engineering process is the glue that holds the technology layers together and enables rational and timely development of computer software. Process defines a framework for a set of key process areas that must be established for effective delivery of software engineering technology. The key process areas form the basis for management control of software projects and 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.

Software engineering methods provide the technical how-to's for building software. Methods encompass a broad array of tasks that include requirements analysis, design, program construction, testing, and support. Software engineering methods rely on a set of basic principles that govern each area of the technology and include modeling activities and other descriptive techniques.

Software engineering tools provide automated or semi-automated support for the process and the methods. When tools are integrated so that information created by one tool can be used by another, a system for the support of software development, called computer-aided software engineering, is established. CASE combines software, hardware, and a software engineering database (a repository containing important information about analysis, design, program construction, and testing) to create a software engineering environment analogous to CAD/CAE (computer-aided design/engineering) for hardware.

c) Describe Cleanroom Approach.
(Diagram 1 ***mark****, explanation 3* ***marks****)*

Ans->

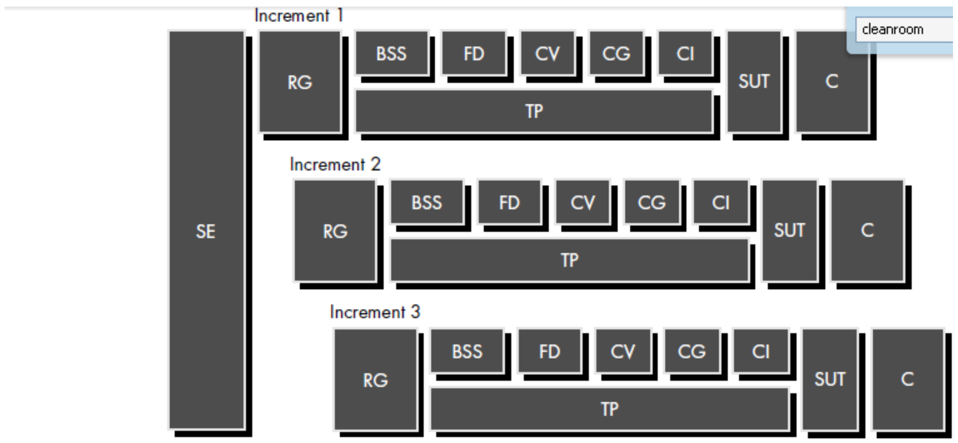


FIGURE 26.1
The cleanroom process model

SE — system engineering	CG — code generation
RG — requirements gathering	CI — code inspection
BSS — box structure specification	SUT — statistical use testing
FD — formal design	C — certification
CV — correctness verification	TP — test planning

The cleanroom approach makes use of a specialized version of the incremental software model. A pipeline of software increments is developed by small independent software engineering teams. As each increment is certified, it is integrated in the whole. Hence, functionality of the system grows with time.

Overall system or product requirements are developed using the system engineering methods. Once functionality has been assigned to the software element of the system, the pipeline of cleanroom increments is initiated. The following tasks occur:

Increment planning. A project plan that adopts the incremental strategy is developed. The functionality of each increment, its projected size, and a cleanroom development schedule are created. Special care must be taken to ensure that certified increments will be integrated in a timely manner.

Requirements Gathering: Using techniques a more-detailed description of customer-level requirements (for each increment) is developed.

Box Structure Specification: A specification method that makes use of box structures is used to describe the functional specification. Conforming to the operational analysis principles, box structures isolate and separate the creative definition of behavior, data, and procedures at each level of refinement.

Formal Design: Using the box structure approach, cleanroom design is a natural and seamless extension of specification. Although it is possible to make a clear distinction between the two activities, specifications (called black boxes) are iteratively refined (within an increment) to become analogous to architectural and component-level designs (called state boxes and clear boxes, respectively).

Correctness Verification: The cleanroom team conducts a series of rigorous correctness verification activities on the design and then the code. Verification begins with the highest-level box structure (specification) and moves toward design detail and code. The first level of correctness verification occurs by applying a set of correctness questions. If these do not demonstrate that the specification is correct, more formal (mathematical) methods for verification are used.

Code Generation, Inspection, And Verification: The box structure specifications, represented in a specialized language, are translated into the appropriate programming language. Standard walkthrough or inspection techniques are then used to ensure semantic conformance of the code and box structures and syntactic correctness of the code. Then correctness verification is conducted for the source code.

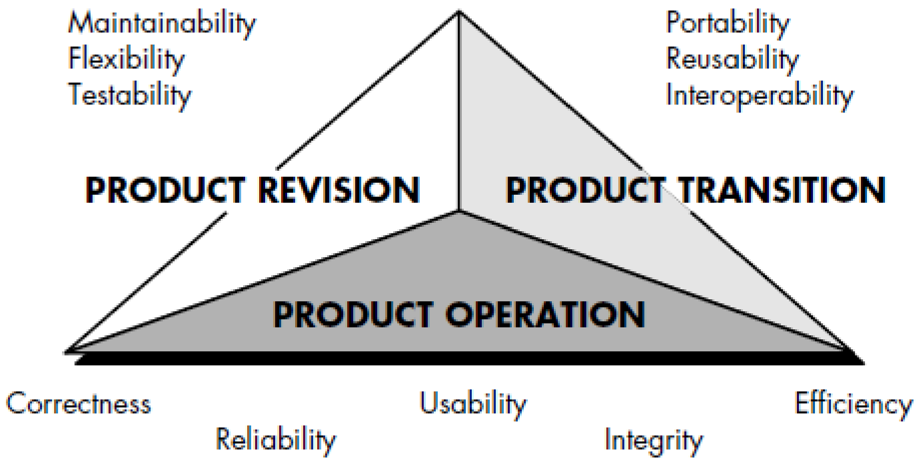
Statistical Test Planning: The projected usage of the software is analyzed and a suite of test cases that exercise a probability distribution of usage are planned and designed. Cleanroom activity is conducted in parallel with specification, verification, and code generation.

Statistical Use Testing. Recalling that exhaustive testing of computer software is impossible, it is always necessary to design a finite number of test cases. Statistical use techniques execute a series of tests derived from a statistical sample (the probability distribution noted earlier) of all possible program executions by all users from a targeted population.

Certification: Once verification, inspection, and usage testing have been completed (and all errors are corrected), the increment is certified as ready for integration.

d) What are McCall's Quality Factor.
*(Explanation of factors **3 marks**, Diagram/List of Factors 1 **mark**)*

Ans->



Referring to the factors, McCall and his colleagues provide the following descriptions:
Correctness. The extent to which a program satisfies its specification and fulfills the customer's mission objectives.
Reliability. The extent to which a program can be expected to perform its intended function with required precision. [It should be noted that other, more complete definitions of reliability have been proposed.]
Efficiency. The amount of computing resources and code required by a program to perform its function.
Integrity. Extent to which access to software or data by unauthorized persons can be controlled.
Usability. Effort required to learn, operate, prepare input, and interpret output of a program.
Maintainability. Effort required to locate and fix an error in a program. [This is a very limited definition.]
Flexibility. Effort required to modify an operational program.
Testability. Effort required to test a program to ensure that it performs its intended function.
Portability. Effort required to transfer the program from one hardware and/or software system environment to another.
Reusability. Extent to which a program [or parts of a program] can be reused in other applications—related to the packaging and scope of the functions that the program performs.
Interoperability. Effort required to couple one system to another.

e) Explain Characteristics of Software Testing Strategies
*(4 characteristics **1 mark each**)*

Ans-> Software testing strategies provide the software developer with a template for testing and all have the following generic characteristics:
• Testing begins at the component level2 and works "outward" toward the integration of the entire computer-based system.
• Different testing techniques are appropriate at different points in time.
• Testing is conducted by the developer of the software and (for large projects) an independent test group.
• Testing and debugging are different activities, but debugging must be accommodated in any testing strategy.

OR

Following are the General Characteristics of Software testing:
(1-2 liner Description for the following Characteristics)
1. Testability
2. Operability
3. Observability

4. Controllability
5. Decomposability
6. Simplicity
7. Understandability