SOFTWARE ENGINEERING

MODEL TEST PAPER-2
Fourth semester [B.Tech]

Sub Code : **ETCS-202** **Max Marks: 75**

Note : Attempt Question no. 1 and any one question from each unit

Q1(a) Define the terms risk mitigation, risk monitoring.

Ans.:- Risk mitigation refers to avoiding risks by developing a strategy for reducing the turn over thereby adopting a proactive approach to risks. Risk monitoring refers to monitoring the facts that may provide an indication of whether the risk is becoming more or less likely.

(b) Name the important approaches used in program debugging.

Ans.:- There are three debugging approaches commonly used
- It occurs as a consequence of successful testing.
- When a test can uncovers an error, then debugging, a process that results in the removal of errors occur.
- Debugging process begins with the execution of test cases.

(c) What is the difference between verification and validation ?

Ans.:- Verification is a set of activities that ensures that the software correctly implements a specified function while Validation is a set of activities that ensures that the software that has been built, is traceable to the customer requirements.

(d) Write short note on The spiral model of software process

Ans.:-The activities in this model can be organized like a spiral, that has many cycles. Typically the inner cycles represent the early phase of requirement analysis along with the prototyping and the outer spirals represent the classic software lifecycle.
This model has been divided into four quadrants, each quadrant representing a major activity like planning, risk analysis engineering and customer evaluation. The software process cycle begins with the planning activity represented by the first quadrant
of this model. Each cycle here begins with the identification of objectives for that cycle, the alternatives and constraints associated with that objective.

(e) Write out the reasons for the Failure of Water Fall Model?

Ans. Reasons for the Failure of Water Fall Model are :

- Real project rarely follow sequential Flow. Iterations are made in indirect manner.
- Difficult for customer to state all requirements explicitly.

- Customer needs more patients as working products reach only at deployment phase.

(f) What are the Drawbacks of RAD Model ?

Ans. Drawbacks of RAD Model are :

- Require sufficient number of Human Resources to create enough number of teams.
- Developers and Customers are not committed, system result in failure.
- Not Properly Modularized building component may Problematic.
- Not applicable when there is more possibility for Technical Risk.

(g) What are the prototyping approaches in software process?

Ans. The prototyping approaches in software process are :

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

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

(h) What are the Objectives of Requirement Analysis ?

Ans. Objectives of Requirement Analysis are :

- It describes what customer requires.
- It establishes a basis for creation of software design.
- It defines a set of requirements that can be validated once the software design is built.

(i) Write two techniques of requirement analysis.

Ans. **DATA DICTIONARY**

The data dictionary can be defined as an organized collection of all the data elements of the system with precise and rigorous definitions so that user and system analyst will have a common understanding of inputs, outputs, components of stores and intermediate calculations.

DATA FLOW DIAGRAM

Data Flow Diagram depicts the information flow and the transforms that are applied on the data as it moves from input to output .Level-0 DFD is called as fundamental system model or context model. In the context model the entire software system is represented by a single bubble with input and output indicated by incoming and outgoing arrows

(j) What is the difference between coupling and cohesion?

Coupling is the quantitative measure of degree to which classes are connected to one another. Coupling should be kept as low as possible**.**

Cohesion is the indication of relative functional strength of a module. It is natural extension of Information Hiding and Performs a single task, requiring little integration with other components.

UNIT-1

Q2(a) What is software? List out the important characteristics of software.

Ans.:- Software is a set of instructions of computer programs that when executed provide desired function and performance. It is both a process and a product. To gain an understanding of software, it is important to examine the characteristics of software, which differ considerably from those of hardware.

Software Characteristic

1). Software is developed or engineered, it is not manufactured.

Unlike hardware, software is logical rather than physical. It has to be designed well before producing it. In spite of availability of many automated software development tools, it is the skill of the individual, creativity of the developers and proper management by the project manager that counts for a good software product.

2). Software does not "wear out".

The hardware components start deteriorating – they are subjected to environmental maladies such as dust, vibration, temperature etc. and at some point of time they tend to breakdown. The defected components can then be traced and replaced.

3) Most software is custom-built, rather than being assembled from existing components

Most of the engineered products are first designed before they are manufactured. Designing includes identifying various components for the product before they are actually assembled. Here several people can work independently on these components thus making the manufacturing system highly flexible. in software , breaking a program into modules is a difficult task, since each module is highly interlinked with other modules.

b) Explain the waterfall model of software process. What are its limitations?

Ans.:-It is the simplest and the widely used process model for software development. Here the phases involved in the software development are organized in a linear order.

In a typical waterfall model, a project begins with the feasibility analysis. on successfully demonstrating the feasibility of a project, the requirements analysis and project planning begins. The design starts after completing the requirements analysis and coding starts after completing the design phase.

The limitations of waterfall model are:

The model states that the entire set of requirements should be frozen before development begins. This is possible for small projects, but is difficult for large projects where the exact requirements may not be known in advance.

3. **Give the outline structure of SRS.** The waterfall model requires formal documents after each phase. This is not possible in GUI-based applications where the documentation will be very extensive.

The customer sees the software only at the end of the development phase. As a result, the customer cannot suggest any changes until the product is delivered.

Ans.:- The outline of SRS structure is:

Introduction

1.1 Purpose

1.2 Document conventions

1.3 Intended audience

1.4 Additional information Formatted: Bullets and Numbering

1.5 Contact information/SRS team members

1.6 References

Overall Description

2.1 Product perspective

2.2 Product functions

2.3 User classes and characteristics

2.4 Operating environment

2.5 User environment

2.6 Design/implementation constraints

2.7 Assumptions and dependencies

External Interface Requirements

3.1 User interfaces

3.2 Hardware interfaces

3.3 Software interfaces

3.4 Communication protocols and interfaces

System Features

4.1 System feature A

4.1.1 Description and priority

4.1.2 Action/result

4.1.3 Functional requirements

4.2 System feature B

Other Nonfunctional Requirements

5.1 Performance requirements

5.2 Safety requirements

5.3 Security requirements

5.4 Software quality attributes

5.5 Project documentation

5.6 User documentation

UNIT-2

Q4 (a) Why is design an important phase in software development life cycle? Describe design process.

Ans.:- Design is an important phase in the software development life cycle because it bridges the requirements specification and the final solution for satisfying the requirements.
The software design is an activity which is after the requirements analysis activity. This phase begins when the requirements document for the system to developed is available.
Design is an important phase in the software development life cycle, it bridges the requirements specification and the final solution for satisfying the requirements.
The design process for the software has two levels:-
1. System design or top-level design
2. Detailed design or logic design
System design
Using this, the modules that are needed for the system are decided, the specifications of these modules and how these modules need to be connected are also decided.
Detailed design
Using this, the internal design of the modules are decided or how the specifications of the modules can be satisfied are decided. This type of design essentially expands the system design to contain more detailed description of the processing logic and data structures so that the designs is sufficiently complete for coding.

(**b) Consider a large -scale project for which the manpower requirement is K=600PY and the development time is 3 years 6 months.**
(a) Calculate the peak manning and peak time

(b)what is the manpower cost after 1year and 2 months ?

Ans. (a) mo=K/td√e
therefore , mo=600/(3.5*1.648)
=104 persons
(b) y(t)=K(1-eat^2)
t=1 year and 2 months
=1.17 years
a=1/2td2=1/2*(3.5)2=0.041
y(1.17)=600(1-e-0.041(1.17)^2)
=32.6PY

Q5(a) What are the different types of Cohesion?

Ans. Different types of cohesion are:

(i)**Coincidentally Cohesive :** The modules in which the set of tasks are related with each other loosely then such modules are called coincidentally cohesive.

(ii)**Logically Cohesive :** A module that performs the tasks that are logically related with each other is called logically cohesive.

(iii)**Temporal Cohesion :** The module in which the tasks need to be executed in some specific time span is called temporal cohesive.

(iv)**Procedural Cohesion** : When processing elements of a module are related with one another and must be executed in some specific order then such module is called procedural cohesive.

(v)**Communicational cohesion :** When the processing elements of a module share the data then such module is called communicational cohesive.

(b) Consider a project with the following functional units:
no. of user inputs =50
no. of user outputs=40
no. of user enquiries=35
no. of user files=06
no. of external interfaces =04
Assume all CAF and weighing factors are average
compute the function points for the project.

Ans $UFP=\sum_{i=1}^{5} \sum_{j=1}^{3} Z_{ij} W_{ij}$

UFP=50*4+40*5+35*4+6*10+4*7
=200+200+140+60+28
=628.

CAF=$(0.65+0.01\sum F_i)$
=(0.65+0.01(14*3))
=0.65+0.42
=1.07

FP=UFP*CAF
=628*1.07
=672.

UNIT-3

Q6(a) Write difference between CMM and ISO 9000.

DIFFERENCE	
ISO 900(INTERNATIONAL STANDARD ORGANISATION)	CMM (CABABILITY MATURITY MODEL)
It applies to any type of industry .	CMM is specially developed for software industry
ISO 9000 addresses corporate business process	CMM focuses on the software Engineering activities.
ISO 9000 specifies minimum requirement.	CMM gets nto technical aspect of software engineering.
ISO 9000 restricts itself to what is required.	It suggests how to fulfill the requirements.
ISO 9000 provides pass or fail criteria.	It provides grade for process maturity.
ISO 9000 has no levels.	CMM has 5 levels: Initial Repeatable Defined Managed Optimization
ISO 9000 does not specifies sequence of steps required to establish the quality system.	It reconnects the mechanism for step by step progress through its successive maturity levels.
Certain process elements that are in ISO are not included in CMM like: 1. Contract management 2. Purchase and customer supplied components 3. Personal issue management 4. Packaging ,delivery, and installation management	Similarly other process in CMM are not included in ISO 9000 1. Project tracking 2. Process and technology change management 3. Intergroup coordinating to meet customer's requirements 4. Organization level process focus, process development and integrated management.

Q6 (b) Mention all the software quality attributes.

Ans Flexibility
Flexibility is the ability of a software to adapt when external changes occur.

Traceability
Traceability is the ability of the Software to offer insight into the inner processing when required. A higher level of traceability is required at time of debugging a problem or at times of new interoperability testing.

Maintainability
Maintainability is the ability of a software to adapt to changes, improve over time, correct any bugs and be proactively fixed through preventive maintenance.

Testability
Testability is the ability of a software to be tested thoroughly before putting into production. This gets very useful in testing the new releases before production.

Reliability
High Reliability is the measure of how a product behaves in varying circumstances.

Robustness
Robustness is defined as the ability of a software product to cope with unusual situation.

Efficiency
Efficiency is the ability of the software to do the required processing on least amount of hardware.

Compatibility
Compatibility is the ability of the software to work with other systems.

Modularity
Modularity is the measure of the extent to which software is composed of separate, interchangeable components, each of which accomplishes one function and contains everything necessary to accomplish this. Modularity increases cohesion and reduces coupling and makes it easier to extend the functionality and maintain the code.

Q7 What are various Halstead software science measures?

Ans. Halstead Science is an estimation technique to find out size, time and effort of a software
Number of Operators and Operands
Halstead´s metrics is based on interpreting the source code as a sequence of tokens and classifying each token to be an operator or an operand.

First we need to compute the following numbers, given the program:
$n1$ = the number of distinct operators
$n2$ = the number of distinct operands
$N1$ = the total number of operators
$N2$ = the total number of operands

The number of unique operators and operands (n1 and n2) as well as the total number of operators and operands (N1 and N2) are calculated by collecting the frequencies of each operator and operand token of the source program.

From these numbers, following measures can be calculated:

Program length (N)
The program length (N) is the sum of the total number of operators and operands in the program:

N = N1 + N2

Vocabulary size (n)
The vocabulary size (n) is the sum of the number of unique operators and operands:

n = n1 + n2

Program volume (V)
The program volume (V) is the information contents of the program, measured in mathematical bits. It is calculated as the program length times the 2-base logarithm of the vocabulary size (n) :

V = N * log2(n)

Halstead's volume (V) describes the size of the implementation of an algorithm. The computation of V is based on the number of operations performed and operands handled in the algorithm.

Difficulty level (D)
The difficulty level or error proneness (D) of the program is proportional to the number of unique operators in the program.
D is also proportional to the ration between the total number of operands and the number of unique operands (i.e. if the same operands are used many times in the program, it is more prone to errors).

D = (n1 / 2) * (N2 / n2)

Program level (L)
The program level (L) is the inverse of the error proneness of the program.
I.e. a low level program is more prone to errors than a high level program.

L = 1 / D

Effort to implement (E)
The effort to implement (E) or understand a program is proportional to the volume and to the difficulty level of the program.

E = V * D

Estimated Program Length
According to Halstead, The first hypothesis of software science is that the length of a well Structured program is a function only of the number of unique operators and operands.
the estimated length is denoted by N^
N^= n1log2n1+n2log2n2

Potential Volume:
Amongst all the programs, the one that has minimal size is said to have the potential volume, V*. Halstead argued that the minimal implementation of any algorithm was through a reference to a procedure that had been previously written. The implementation of this algorithm would then require nothing more than invoking the procedure and supplying the operands for its inputs and output parameters.

V*= (2+n2*) log2(2+n2*)

Estimated program Level/difficulty
Halstead offered an alternate formula that estimates the program level.
L^ =2n2/n1N2
Hence, D =1 /L^

Effort:
Halstead hypothesized that the effort required to implement a program increases as the size of the program increases. It takes more effort to implement a program at a lower level than the higher level.
E= V/L^

UNIT-4

Q8(a) What is software testing ? Describe the two ways of testing any engineered software product.

Ans.:- Software testing is the process of testing the functionality and correctness of software by running it.
The two ways of testing any engineered software product :-
1) White-Box Testing
2) Black-Box testing
Black-box testing
It is also known as functional testing. Knowing the specified function that the product has been designed to perform, tests can be conducted to demonstrate that each function is fully operational
Example :- Boundary value analysis. EXAMPLE :- basic path testing.

(b) Explain integration testing.

Ans Integration testing is the phase in software testing in which individual software modules are combined and tested as a group. It occurs after unit testing and before validation testing. Integration testing takes as its input modules that have been unit tested, groups them in larger aggregates, applies tests defined in an integration test plan to those aggregates, and delivers as its output the integrated system ready for system testing.
The purpose of integration testing is to verify functional, performance, and reliability requirements placed on major design items. These "design items", i.e. assemblages (or groups of units), are exercised through their interfaces using black box testing, success and error cases being simulated via appropriate parameter and data inputs. Simulated usage of shared data areas and inter-process communication is tested and individual subsystems are exercised through their input interface. Test cases are constructed to test whether all the components within assemblages interact correctly, for example across procedure calls or process activations, and this is done after testing individual modules, i.e. unit testing. The overall idea is a "building block" approach, in which verified assemblages are added to a verified base which is then used to support the integration testing of further assemblages.

Q9(a)What is software maintenance? Describe various categories of maintenance

Ans. Software maintenance in software engineering is the modification of a software product after delivery to correct faults, to improve performance or other attributes.
A common perception of maintenance is that it merely involves fixing defects.
Software maintenance is a very broad activity that includes error correction, enhancements of capabilities, deletion of obsolete capabilities, and optimization. Because change is inevitable, mechanisms must be developed for evaluation, controlling and making modifications.

Various categories of maintenance are:
- Corrective maintenance: Reactive modification of a software product performed after delivery to correct discovered problems.
- Adaptive maintenance: Modification of a software product performed after delivery to keep a software product usable in a changed or changing environment.
- Perfective maintenance: Modification of a software product after delivery to improve performance or maintainability.
- Preventive maintenance: Modification of a software product after delivery to detect and correct latent faults in the software product before they become effective faults.

(b) Discuss methods of cyclomatic complexity of code segment ?

Ans. Cyclomatic complexity is a software metric. It is used to indicate the complexity of a program. It is a quantitative measure of logical strength of the program. It directly measures the number of linearly independent paths through a program's source code.. Cyclomatic complexity is computed using the control flow graph of the program: the nodes of the graph correspond to indivisible groups of commands of a program, and a directed edge connects two nodes if the second command might be executed immediately after the first command. Cyclomatic complexity may also be applied to individual functions, modules, methods or classes within a program.

There are three different ways to compute the cyclomatic complexity.

Method 1:
Given a control flow graph G of a program, the cyclomatic complexity V(G) can be computed as:
V(G) = E – N + 2
where N is the number of nodes of the control flow graph and E is the number of edges in the control flow graph. For the CFG of example shown in fig. 10.4, E=7 and N=6. Therefore, the cyclomatic complexity = 7-6+2 = 3.

Method 2:
An alternative way of computing the cyclomatic complexity of a program from an inspection of its control flow graph is as follows:
V(G) = Total number of bounded areas + 1
In the program's control flow graph G, any region enclosed by nodes and edges can be called as a bounded area. This is an easy way to determine the McCabe's cyclomatic complexity. But, what if the graph G is not planar, i.e. however you draw the graph, two or more edges intersect? Actually, it can be shown that structured programs always yield planar graphs. But,

presence of GOTO's can easily add intersecting edges. Therefore, for non-structured programs, this way of computing the McCabe's cyclomatic complexity cannot be used. The number of bounded areas increases with the number of decision paths and loops. Therefore, the McCabe's metric provides a quantitative measure of testing difficulty and the ultimate reliability. This method provides a very easy way of computing the cyclomatic complexity of CFGs, just from a visual examination of the CFG. On the other hand, the other method of computing CFGs is more amenable to automation, i.e. it can be easily coded into a program which can be used to determine the cyclomatic complexities of arbitrary CFGs.

Method 3:

The cyclomatic complexity of a program can also be easily computed by computing the number of decision statements of the program. If N is the number of decision statement of a program, then the McCabe's metric is equal to N+1.