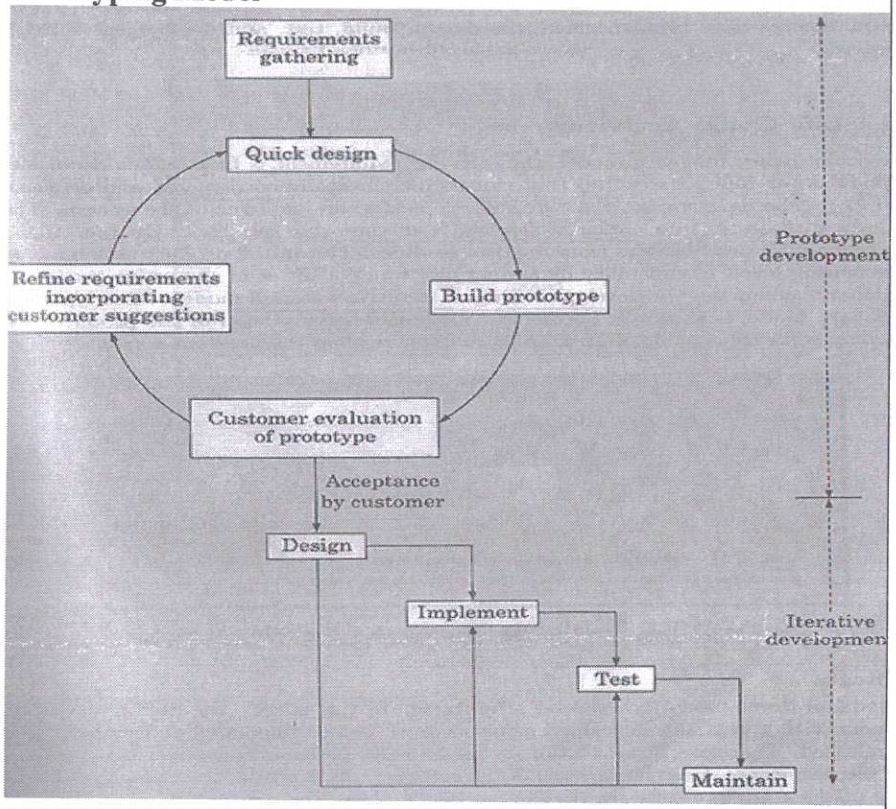
	PROJECT MANAGEMENT and SOFTWARE ENGINNERING Rev 2015 TED 5132 (ANSWER KEY)		
Q no	**Scoring Indicators**	**split score**	**Total score**
I	**PART A**		
1	Product engineering process and Process management process.	1+1	2
2	Procedural oriented approach (Traditional Design approach) Object-Oriented Design approach	1+1	2
3	The design of a system is a blueprint or a plan for a solution for the system	2	2
4	**Error** refers to the difference between the actual output of a software and the correct output. **Fault** is a condition that causes a system to fail in performing its required function	1+1	2
5	Quality Planning, Quality Assurance, Quality Control, Continuous improvement process	4*1/2	2
II	**PART B**		
1	**Maintenance Phase** **Corrective Maintenance :** Involves correcting errors that were not discovered during the product development phase. **Perfective Maintenance :** Involves improving the implementation of the system, and enhancing the functionalities of the system according to the customer's requirements. **Adaptive Maintenance :** Required for porting the software to work in a new environment	2+2+2	6
2	**Desirable Characteristics of an SRS** **Correct** – An SRS is correct if every requirement included in the SRS represents something required in the final system. **Complete** – An SRS is complete if everything the software is supposed to do and the responses of the software to all classes of input data are specified in the SRS. **Unambiguous** – An SRS is unambiguous if and only if every requirement stated has one and only one interpretation. **Verifiable** – A requirement is verifiable if there exists some cost-effective process that can check whether the final software meets that requirement. **Consistent** – An SRS is consistent if there is no requirement that conflicts with another. **Ranked for importance and/or stability** – An SRS is ranked for importance and/or stability if for each requirement the importance and stability of the requirement are indicated.	6*1	6

3	**Role of software architecture** Software architecture of a system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them. **Understanding and communication:** An architecture description is primarily to communicate the architecture to its various stakeholders, which include the end users who will use the system **Reuse:** software can be assembled and it can be available for others to reuse. **Construction and Evolution:** different teams can separately work on different parts and the parts can be built independently. **Analysis:** the designers can select the design approach which best suit the needs, among the alternatives .	2+4	6
4	**Refactoring** Refactoring is the technique to improve existing code and prevent design change/ decay with time. 1. **Reduced coupling** **2. Increased cohesion** **3. Better adherence to open-closed principle** - Refactoring involves changing the code to improve one of the design properties, while keeping the external behavior the same. - The main risk of refactoring is that existing working code may "break" due to the changes being made. - Use test scripts available to test existing functionality. - With refactoring, code becomes continuously improving. - With refactoring, the quality of the design improves	2+2+2	6
5	**TESTING PROCESS** **Test Plan:** • A test plan is a document for the entire project • The inputs for forming the test plan are: • (1) Project plan, • (2) Requirements document, • (3) Architecture or design document. **Test Case Design** • Test case design has to be done separately for each unit. • **The test case specifications document** is also used to record the result of testing. **Test Case Execution** Executing the test cases may require construction of **driver modules or stubs**. During test case execution, defects are found.	2+2+2	6
6	**The main stages of project management lifecycle are**: **1. Initiation**: when the project starts - Identify the objective of the project, determine whether the project is feasible, and identify the major deliverables for the project. **2. Planning**: when all of the key decisions are made	6*1	6

	- Break down the larger project into smaller tasks, build the project team and prepare a schedule for the completion of assignments. **3.Execution** – When project work actually takes place. **4.Control** – When adjustments are made to the plan. **5.Monitoring** – When project progress is checked. **6.Termination** – When the project comes to an end		
7	Capability Maturity Model (CMMI) is a **bench-mark** for measuring the **maturity** of an organization's software process. It is a methodology used to develop and refine an organization's software development process. • **Initial** - Company has **no standard process** for software development. • **Managed** - The projects of the organization have ensured that requirements are managed • **Defined** - Company has a standard set of processes and controls for the entire organization • **Quantitatively Managed** - company has installed systems to measure the quality of those processes across all projects. • **Optimizing:** Company has accomplished all of the above and can see patterns in performance over time	1+5	6
	PART C		
III (a)	**Testing phase consists of:** 1. **Unit Testing** 2. **Integration** 3. **System Testing** **Unit Tesing Phase** : - Tested individual modules independently as a stand alone unit, and debugged **Integration Testing** Different modules are integrated in a planned manner: - Modules are almost never integrated in one shot. - Normally integration is carried out through a number of steps. - During integration step, the partially integrated system is tested. **System Testing** **α-testing** : performed by development team. **β-testing**: performed by friendly set of customers **Acceptance Testing**: performed by customer after the system has been delivered and determine whether to accept the delivered software or to reject it	3*3	9
(b)	**Advantages of Spiral model** 1. Good for large and critical projects 2. Working software is produced early during the lifecycle 3. Large amount of risk analysis	3+3	6

	Disadvantages of Spiral model 1. Involves higher cost 2. Not suitable for smaller projects 3. Project success depends on the risk analysis phase - requires high expertise in risk analysis		
IV (a)	**Prototyping Model**  The first phase of prototyping model is **prototype development.** This is followed by an **iterative development cycle**. In this model, prototyping starts with an initial requirements gathering phase. A quick design is carried out and a prototype is built. The developed prototype is submitted to the customer for his evaluation. Based on the customer feedback, the requirements are refined and the prototype is suitably modified. This cycle continues until the customer approves the prototype. Once the customer approves the prototype, the actual system is developed using the iterative waterfall approach. **Advantages of Prototyping model** 1. Benefits from user input 2. users get a better understanding of the system 3. Errors and risks can be detected **Disadvantages of Prototyping model** 1.Increases complexity of the overall system 2.Involves implementing and then repairing the way a system is built, so errors are an inherent part of the development process	Fig: 4 Expln: 4 ad&dis adv; 1	9

(b)	**Requirement Analysis phase** **1.Requirements gathering and analysis** Collect all relevant information regarding the product to be developed from the customer. The gathered requirements are analysed to remove the incompleteness and inconsistencies in them. **2.Requirements specification** The customer requirements identified. The 3 most important contents of this document are **functional requirements, non-functional requirements, and the goals of implementation.**	3+3	6
V (a)	**How to plan for a software project** Planning has two basic objectives— 1. establish reasonable cost, schedule, and 2. draw out a plan to deliver the project goals. The **inputs** to the planning activity are 1.The requirements specification 2.The architecture description. **Planning includes the following activities** 1. Effort Estimation 2. Quality Planning 3. Risk Management Planning 4. Project Monitoring Plan 5. Detailed Scheduling **Effort Estimation**: This determining the staffing level for a project during different phases. **Quality Planning**:The quality plan is the set of quality-related activities that a project plans to do to achieve the quality goal. **Risk Management Planning**: Risk management is an attempt to minimize the chances of failure caused by unplanned events. **Project Monitoring Plan**: Project managers need to get the status of project development. **Detailed Scheduling**: overall plan has to be translated into a detailed action plan. There are generally two **main outputs** of the planning activity: 1. The overall project management plan document. 2. The detailed plan, often referred to as the detailed project schedule.	Input: 2 Activities :5 Output: 2	9
(b)	**Detailed Design** **Logic/Algorithm Design.** • The goal is to specify the logic for the different modules. • Specifying the logic will require developing an algorithm that will implement the given specifications. • An algorithm is a procedure for solving a problem. **State Modeling of Classes**: to understand the behavior of a class view it as a combination of states and transitions between states. A state diagram attempts to represent only the logical states of the object.	3+3	6

VI (a)

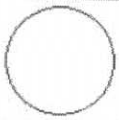
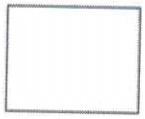
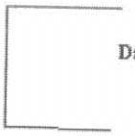
Data Flow Diagram

DFD is a hierarchical graphical model which shows the different functions of the system and data interchange among the functions.

Level 1 DFD : Examine the SRS document

Higher level DFDs : Each high-level function is separately decomposed into sub functions:

DFD symbols:

Symbol	Meaning
 Process	Single process: A circle is used to represent the entire system.
	Data flow: An arrow is used to represent the flow of data between the process and external entities.
 External entity	External entity: A square or rectangle represents any person or organisation that sends data to or receives data from the system.
 Data store	Data store: An open rectangle represents the location where data is stored. It could be a filing cabinet, hard disk.

Explanation:3
Symbols:3
Example:3

9

(b)

Cohesion

Cohesion of a module represents how tightly bound the internal elements of the module are to one another. There are several levels of cohesion.

1. Coincidental
2. Logical
3. Temporal
4. Procedural
5. Communicational
6. Sequential
7. Functional

Coincidental cohesion occurs when there is no meaningful relationship among the elements of a module. It is the **lowest** cohesion.

A module has **logical** cohesion if there is some logical relationship between the elements of a module

Temporal cohesion is the same as logical cohesion, except that the elements are also related in time and are executed together.

A **procedurally** cohesive module contains elements that

6

6

	belong to a common procedural unit. A module with **communicational** cohesion has elements that are related by a reference to the same input or output data. When the elements are together in a module because the output of one forms the input to another, we get **sequential** cohesion. **Functional** cohesion is the strongest cohesion. All the elements of the module are related to performing a single function.		
VII (a)	**Incrementally Developing Code** **An Incremental Coding Process** A better process for coding is to develop the code incrementally. • That is, write code for implementing only part of the functionality of the module. This code is compiled and tested with some quick tests to check the code that has been written so far. When the code passes these tests, the developer proceeds to add further functionality to the code, which is then tested again. **Test-Driven Development** • Instead of writing code and then developing test cases to check the code, in TDD follows the other way—a programmer first writes the test scripts, and then writes the code to pass the tests. • The whole process is done incrementally, with tests being written based on the specifications and code being written to pass the tests. • This is a relatively new approach, which has been adopted in the extreme programming (XP) methodology. **Pair Programming** • In pair programming, code is not written by individual programmers but by a pair of programmers. This pair together writes the code. • When errors are noticed, they are pointed out and corrected.	3+3+3	9
(b)	**Unit testing** • **Unit testing** is undertaken after a module has been coded and carried out at programmer level during the coding phase. • During unit testing, testers execute the unit with a variety of test cases and study the actual behavior of different functions of a system independently. • Before carrying out unit testing, the unit test cases have to be designed. • the programmer finds the defect in the program • After removing the defect, the programmer will generally execute the test case • **Drivers /stubs** play the role of the "calling" module and are often responsible for getting the test data, executing the unit with the test data, and then reporting the result.	6	6

VIII (a)

White box testing

1.Control-flow based Criteria

Let the **control flow graph** of a program **P** be **G**. A **node** in this graph represents a block of statements that is always executed together. An **edge** (i, j) (from node i to node j) represents a possible transfer of control after executing the last statement of the block represented by node i to the first statement of the block represented by node j. A node corresponding to a block whose first statement is the start statement of P is called the **start node** of G, and a node corresponding to a block whose last statement is an exit statement is called an **exit node**. A **path** is a finite sequence of nodes. A **complete path** is a path whose first node is the start node and the last node is an exit node.

Statement Coverage Criteria

This criteria requires that each statement of the program be executed at least once during testing. No test case is needed that ensures that the condition in the if statement evaluates to false. Consider the program

```
int abs (X)
int X;
{
        if ( X > =0) X = 0-X;      // Error
        return (X);
}
```

Suppose we execute the function with the test case {x=0}. The statement coverage criterion will be satisfied by testing with this set, but the error will not be revealed.

Branch Coverage Criteria

This criteria requires that each edge in the control flow graph be traversed at least once during testing.

Eg:

```
int check (x)
int x;
{
        if ( ( x >=0) && (x <=200))
                check=True;
        else
                check=False:
}
```

The module is incorrect, as it is checking for x <=200 instead of 100

Path Coverage Criteria

This criteria requires that all possible paths in the control flow graph be executed during testing. It is also called **all-paths** criterion. Testing based on this criterion is called **path testing.** The difficulty with this criterion is that programs that contain loops can have an infinite number of possible paths. Even then, path coverage criteria is the strongest among the three.

6+3 | 9

	2.Test Case Generation and Tool Support Once a coverage criterion is decided, two problems have to be solved to use the chosen criteria for testing. 1. Decide if a set of test cases satisfy the criterion. 2. Generate a set of test cases for a given criterion. Tools can be used to determine whether the criterion has been satisfied. These tools can aid the testing process. There are many tools available for statement and branch coverage. To get the coverage data, the execution of the program during testing has to be closely monitored. Three phases in generating coverage data: 1. Instrument the program with probes 2. Execute the program with test cases 3. Analyse the results of the probe data Probe insertion can be done automatically by a pre-processor. The execution of the program is done by the tester. After testing, the coverage data is displayed by the tool		
(b)	**Code Inspection** ★ Code inspection is a code review technique for reduction in coding errors and to produce high quality code. The basic goal of inspections is to improve the quality of code by finding defects - Code inspection is conducted by programmers and for programmers - It is a structured process with defined roles for the participants. - The focus is on identifying defects, not fixing them. - Inspection data is recorded and used for monitoring the effectiveness of the inspection process. The different stages in this process are: planning, self-review, group review meeting, and rework and follow-up. These stages are generally executed in a linear order.	Expln: 3 List: 3	6
IX (a)	**Risk Management** A possibility of suffering from loss in software development process is called a software risk. A software risk can be of two types : (1) internal risks that are within the control of the project manager (2) external risks that are beyond the control of project manager. Risk Management comprises of following processes: 1. Risk Identification 2. Risk Analysis 3. Risk Planning 4. Risk Monitoring Identification - Make note of all possible risks, which may occur in the project. Analyse/Categorize - During risk analysis process, assess probability	Risk:3 process : 4*1.5 = 6	9

	and seriousness of each risk. Categorize known risks into high, medium and low risk intensity as per their possible impact on the project. Manage/Mitigate - Consider each risk and develop a strategy to manage that risk. For this, make plan to avoid or face risks. Attempt to minimize their side-effects. Monitor - Closely monitor the potential risks and their early symptoms. Also monitor the effects of steps taken to mitigate or avoid them.		
(b)	**COCOMO** stands for **CO**nstructive **CO**st **MO**del **Basic COCOMO** It gives single estimate based on the product attributes which can be derived from the equation: **$PM = C * KDSI^S * M$** Where, PM is the effort measured in person-months C is a complexity factor (2.4/3.0/3.6) KDSI is a product metric Exponent S is a close to1 M is multiplier based on process,product and development attributes **Intermediate COCOMO** The following attributes are taken into considerations: • Product attributes • Hardware attributes • Personnel attributes • Project attributes **Detailed COCOMO** Detailed COCOMO considers all parameters of the intermediate model The five phases of detailed COCOMO are:- • Plan and requirement. • System design. • Detailed design. • Module code and test. • Integration and test.	3*2	6
X (a)	**Configuration Management** Configuration management is a process of **tracking** and **controlling** the **changes** in software in terms of the **requirements**, **design**, functions and **development** of the product. **Software configuration management (SCM) Process** **1.Configuration Identification** **2.Configuration Control** **3.Configuration Status Accounting** **4.Configuration Authentication** **1.Configuration Identification** **Programs**, **documents** and **data** are called **Configurable Item(CI).**	CM meaning :1 SCM Process : 4*2=8	9

	➢ The end product is made up of a group of CIs. ➢ The status of CIs at a given time is called **'a baseline'.** ➢ It serves as a reference point in the software development life cycle. ➢ Each baseline is the **sum of older baseline and a series of approved changes made on the CI.** **Configuration Control (CC)** ➢ Process of **deciding** and **coordinating** the approved changes for the proposed CIs and also **implementing** the changes on the appropriate baseline is **Configuration Control.** **Configuration Status Accounting (CSA)** ➢ It is a **bookkeeping process** – involves tracking of what is in each version of software and the changes that lead to this version. **Configuration Authentication(CA)** ➢ It is the process of assuring that the new baseline has incorporated all the planned and approved changes. ➢ It involves **verifying** that all the **functional aspects** of software are complete.		
(b)	**Resource Management** ❖ This may include human resource, productive tools and software libraries. ❖ The resources are available in limited quantity and stay in the organization as a pool of assets. ❖ Allocating extra resources increases development cost in the end. ❖ So it is necessary to estimate and allocate enough resources for the project. Resource management includes - ➢ Defining proper organization project by creating a project team and allocating responsibilities to each team member. ➢ Determining resources required at a particular stage and their availability. ➢ Manage Resources by generating resource request when they are required and de-allocating them when they are no more needed.	6	6