COURSE: PROJECT MANAGEMENT & SOFTWARE ENGINEERING

Revision: (2015) **Course Code: 5132**

Qn. No	Scoring Indicators	Split up Score	Sub Total	Total
I.	1. The processes that deal with the technical and management issues of software development are collectively called the **software process**. As a software project will have to engineer a solution and properly manage the project, there are clearly two major components in a software process—a **Development process** and a **Project management process.**	2	2	2
	2. A structure chart is a graphical representation of a system's structure; in particular, its modules and their interconnections. Each module is represented by a box	2	2	2
	3. Code review for a model is carried out after the module is successfully compiled and the all the syntax errors have been eliminated.Normally, two types of reviews are carried out on the code of a module. These two types code review techniques are code inspection and code walk through.	2	2	2
	4. Test Suite :A group of related test cases that are generally executed together to test some specific behavior or aspect of the SUT is often referred to as a test suite. A testing framework is also called a test harness.	1 + 1	2	2
	5. A risk is any anticipated unfavourable event or circumstance that can occur while a project is underway.	2	2	2
II	1. The classical waterfall model is an idealistic one since it assumes that no development error is ever committed by the engineers during any of the life cycle phases. However, in practical development environments, the engineers do commit a large number of errors in almost every phase	6	6	

of the life cycle. These defects usually get detected much later in the life cycle. Therefore, in any practical software development work, it is not possible to strictly follow the classical waterfall model.. This is drawback is over came by giving feedback from each phase in Iterative model. This model does not require all requirements to be known at the start, allows feedback from earlier iterations for next ones, and reduces risk as it delivers value as the project proceeds.

2.

1. Correct

2. Complete

3. Unambiguous

4. Verifiable

5. Consistent

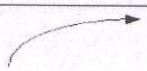
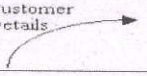
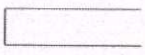
6. Ranked for importance and/or stability

1*6 | 6 | 6

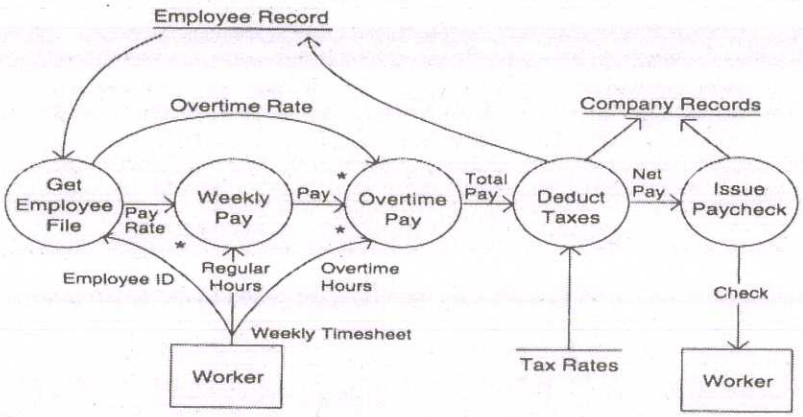
3.

- A DFD shows the flow of data through a system. It views a system as a function that transforms the inputs into desired outputs
- The DFD aims to capture the transformations that take place within a system to the input data so that eventually the output data is produced.

2 | 6 | 6

Name	Symbol	Description	Example
Entity		Used to represent people and organizations outside the system. They either input information to the system, accept output information from the system or both	Customer
Process		These are actions that are carried out with the data that flows around the system. A process accepts input data and produces data that it passes on to another part of the DFD	Verify Order
Data Flow		These represent the flow of data to or from a process	Customer Details 
Data Store		This is a place where data is stored either temporarily or permanently	Products

2

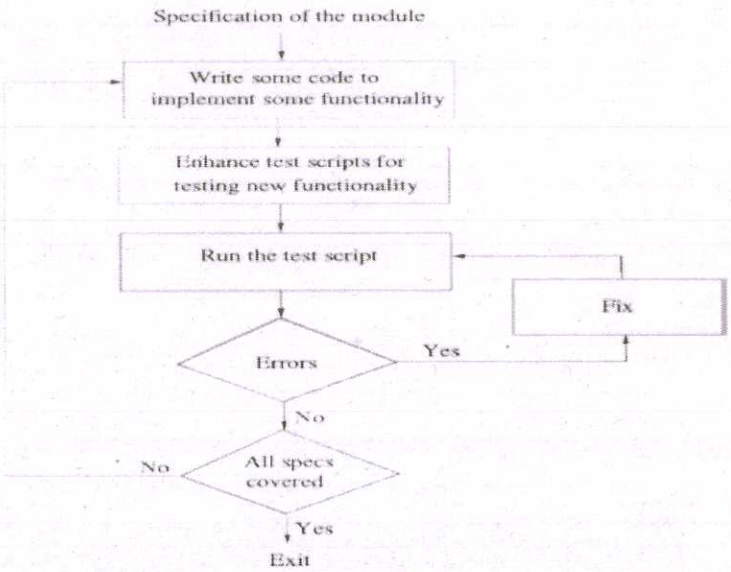


2

4.

6 6 6

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. In other words, the code is developed incrementally, testing it as it is built.



5.

3*2 6 6

The testing process for a project consists of three high-level tasks—

- Test planning,
- Test case design, and
- Test execution.

A test plan is a general document for the entire project that defines the scope, approach to be taken, and the schedule of testing, as well as

	identifies the test items for testing and the personnel responsible for the different activities of testing.The inputs for forming the test plan are: (1) project plan, (2) requirements document, and (3) architecture or design document. Test case specification gives, for each unit to be tested, all test cases, inputs to be used in the test cases, conditions being tested by the test case, and outputs expected for those test cases. During test case execution, defects are found. These defects are then fixed and testing is done again to verify the fix. To facilitate reporting and tracking of defects found during testing (and other quality control activities), defects found are often logged. Defect logging and tracking is considered one of the best practices for managing a project and is followed by most software organizations. 6. 1. Identify all the major activities that need to be carried out to complete the project. 2. Break down each activity into tasks. 3. Determine the dependency among different tasks. 4. Establish the estimates for the time durations necessary to complete the tasks. 5. Represent the information in the form of an activity network. 6. Determine task starting and ending dates from the information represented in the activity network. 7. Determine the critical path. A critical path is a chain of tasks that determines the duration of the project. 8. Allocate resources to tasks. 7. Risk management aims at reducing the chances of a risk becoming real as well as reducing the impact of a risks that becomes real. Risk	6 3*2	6 6	6 6

	management consists of three essential activities—**Risk identification, Risk assessment, and Risk mitigation.** A project can be subject to a large variety of risks. In order to be able to systematically identify the important risks which might affect a project, it is necessary to categorize risks into different classes. The project manager can then examine which risks from each class are relevant to the project. There are three main categories of risks which can affect a software project: project risks, technical risks, and business risks. The objective of risk assessment is to rank the risks in terms of their damage causing potential. For risk assessment, first each risk should be rated in two ways: • The likelihood of a risk becoming real (r). • The consequence of the problems associated with that risk (s). • Based on these two factors, the priority of each risk can be computed as follows: p = r * s • If all identified risks are prioritised, then the most likely and damaging risks can be handled first and more comprehensive risk abatement procedures can be designed for those risks. There are three main strategies for risk mitigation Avoid the risk ,Transfer the risk, Risk reduction			
III	a) Feasibility Study The main aim of feasibility study is to determine whether it would be financially and technically feasible to develop the product. Requirements Analysis& Specification	Name of phases - 5 Mark Explanation – 5 Marks	10	15

The aim of the requirements analysis and specification phase is to understand the exact requirements of the customer and to document them properly. This phase consists of two distinct activities, namely

- Requirements gathering and analysis, and
- Requirements specification

Design

The goal of the design phase is to transform the requirements specified in the SRS document into a structure that is suitable for implementation in some programming language. In technical terms, during the design phase the software architecture is derived from the SRS document

Coding & Unit Testing

The purpose of the coding and unit testing phase (sometimes called the implementation phase) of software development is to translate the software design into source code. Each component of the design is implemented as a program module. The end-product of this phase is a set of program modules that have been individually tested.

Integration & System Testing

During the integration and system testing phase, the modules are integrated in a planned manner. During each integration step, the partially integrated system is tested and a set of previously planned modules are added to it. Finally, when all the modules have been successfully integrated and tested, system testing is carried out. The goal of system testing is to ensure that the developed system conforms to its requirements laid out in the SRS document.

Maintenance

Maintenance involves performing any one or more of the following three kinds of activities:

Correcting errors that were not discovered during the product development phase. This is called corrective maintenance.

	Improving the implementation of the system, and enhancing the functionalities of the system according to the customer's requirements. This is called perfective maintenance. Porting the software to work in a new environment. For example, porting may be required to get the software to work on a new computer platform or with a new operating system. This is called adaptive maintenance. b) Approaches are based on some common principles, some of which are – Working software is the key measure of progress in a project. – For progress in a project, therefore, software should be developed and delivered rapidly in small increments. - Even late changes in the requirements should be entertained (small-increment model of development helps in accommodating them). – Face-to-face communication is preferred over documentation. - Continuous feedback and involvement of customer is necessary for developing good-quality software. – Simple design which evolves and improves with time is a better approach than doing an elaborate design up front for handling all possible scenarios. – The delivery dates are decided by empowered teams of talented individuals (and are not dictated). Extreme programming (XP) is one of the most popular and well-known approaches in the family of agile methods.	5	5	
IV	a) In the prototyping model, a prototype is built before building the final system, which is used to further develop the requirements leading to	Explanation – 5 Marks	8	1·5

more stable requirements. This is useful for projects where requirements are not clear.

A prototype is a toy implementation of the system. A prototype usually exhibits limited functional capabilities, low reliability, and inefficient performance compared to the actual software.

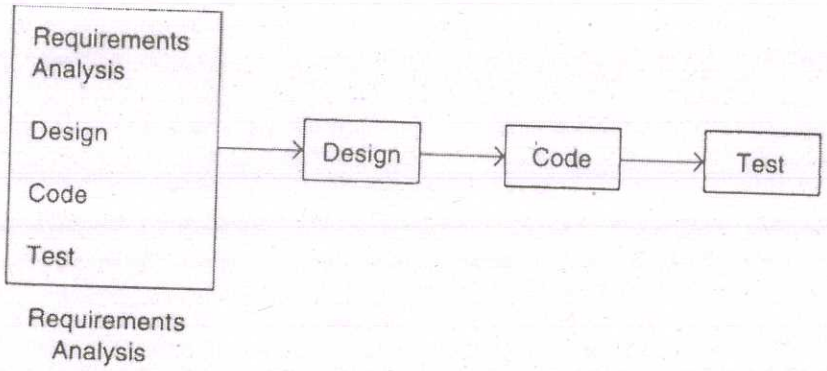
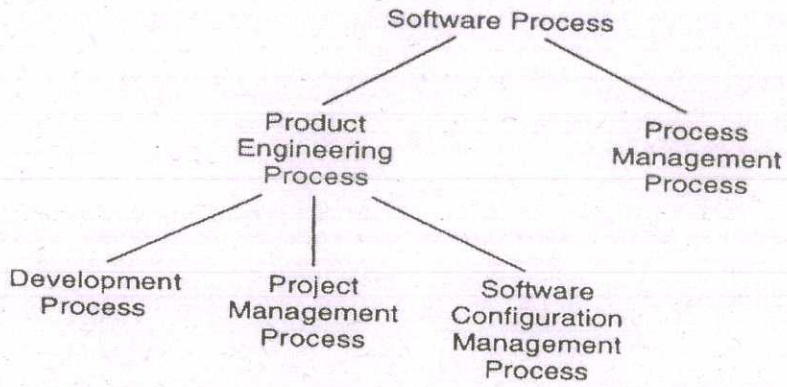


Figure – 3 Marks

b)



Explanation - 5 Marks
Fig - 2 Marks

7

The processes that deal with the technical and management issues of software development are collectively called the **software process**. As a software project will have to engineer a solution and properly manage the project, there are clearly two major components in a software process—a **Development process** and a **Project management process.**

- The Development process specifies all the engineering activities that need to be performed,
- The Management process specifies how to plan and control these activities so that cost, schedule, quality, and other objectives are met.

	• The objective of software configuration control process component process is to primarily deal with managing change, so that the integrity of the products is not violated despite changes.			
V	a) The requirements process typically consists of three basic tasks: - - Problem or Requirement analysis - Requirements specification and - Requirements validation In Problem analysis an effort is to • understand the system behavior • constraints on the system, • its inputs and outputs, etc. Requirements specification The understanding obtained by problem analysis forms the basis of requirements specification, in which the focus is on clearly specifying the requirements in a document. As analysis produces large amounts of information and knowledge with possible redundancies, properly organizing and describing the requirements is an important goal of this activity Requirements validation Requirements validation focuses on ensuring that what have been specified in the SRS are indeed all the requirements of the software and making sure that the SRS is of good quality. The requirements process terminates with the production of the validated SRS b) The 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	2 Explanation - 3*2 3+ explanation 4	8 7	15

	A view represents the system as composed of some types of elements and relationships between them. Most of the proposed views generally belong to one of these three types – Module – Component and connector – Allocation **Module views** show how the software is structured as a set of implementation units, **C&C views** show how the software is structured as interacting runtime elements, and **allocation views** show how software relates to non-software structures. These three types of view of the same system form the architecture of the system.			
VI	Complexity Matrices are • *Network Metrics* • *Stability Metrics* • *Information Flow Metrics* Network metrics focus on the structure chart of a system • They attempt to define how "good" the structure or network is in an effort to quantify the complexity of the call graph • The simplest structure occurs if the call graph is a tree, with each node having two children • *As a result, the graph impurity metric is defined as **nodes - edges - 1*** • *In the case of a tree, this metric produces the result zero since there is always one more node in a tree than edges*	Listing – 3 Marks Explanation – 3 * 4 Marks	15	15

- *This metric is designed to make you examine nodes that have high coupling and see if there are ways to reduce this coupling*

Stability of a design is a metric that tries to quantify the resistance of a design to the potential ripple effects that are caused by changes in modules

- The creators of this metric argue that the higher the stability of a design, the easier it is to maintain the resulting system
- The basic idea is to define stability in terms of the number of assumptions made by other components about a particular component.
    - *This provides a stability value for each particular module*
    - *A formula is then given to combine the stability value of each module into a value for the entire system*
- In essense, the lower the amount of coupling between modules, the higher the stability of the overall system

Information flow metrics attempt to define the complexity of a system in terms of the total amount of information flowing through its modules

- *Approach 1*
    - A module's complexity depends on its intramodule complexity and its intermodule complexity
    - intramodule complexity is approximated by the (estimated) size of the module in lines of code
    - intermodule complexity is determined by the total amount of information (abstract data elements) flowing into a module (inflow) and the total amount of information flowing out of a module (outflow)
    - The module design complexity Dc is defined as Dc = size * (inflow*outflow)2
    - The term (inflow * outflow)2 refers to the total number of input and output combinations, and this number is

squared since the interconnections between modules are considered more important to determining the complexity of a module than its code size

Approach 2

- Approach 1 depends largely on the amount of information flowing in and out of the module
- Approach 2 is a variant that also considers the number of modules connected to a particular module; in addition, the code size of a module is considered insignificant with respect to a module's complexity
- The module design complexity D_c is defined as $\mathbf{D_c = (fan\_in * fan\_out) + (inflow*outflow)}$
- fan_in above refers to the number of modules that call this module, fan_out is the number of modules called by this module

Classification

- Neither of these metrics is any good, unless they can tell us when to consider a module "too complex"
- To this end, an approach was developed to compare a module's complexity against the complexity of the other modules in its system
- **avg_complexity** is defined as the average complexity of the modules in the current design
- **std_deviation** is defined as the standard deviation in the design complexity of the modules in the current design
- A module can be classified as *error prone*, *complex*, or *normal* using the following conditions D_c is the complexity of a particular module
    - A module is error prone if D_c > avg_complexity + std_deviation
    - A module is complex if avg_complexity < D_c < avg_complexity + std_deviation
    - Otherwise a module is considered normal

VII	There are two basic approaches to designing the test cases to be used in testing: black-box and white-box. • In black-box testing, the tester only knows the inputs that can be given to the system and what output the system should give. It is also called functional or behavioral testing. • white-box is concerned with testing the implementation of the program. White-box testing is also called structural testing. <u>Black-box testing</u> In the black-box testing, test cases are designed from an examination of the input/output values only and no knowledge of design, or code is required. The following are the two main approaches to designing black box test cases. • Equivalence class portioning • Boundary value analysis **Equivalence Class Partitioning** • In this approach, the domain of input values to a program is partitioned into a set of equivalence classes. This partitioning is done such that the behavior of the program is similar for every input data belonging to the same equivalence class. The main idea behind defining the equivalence classes is that testing the code with any one value belonging to an equivalence class is as good as testing the software with any other value belonging to that equivalence class. Equivalence classes for a software can be designed by examining the input data and output data. The following are some general guidelines for designing the equivalence classes: • If the input data values to a system can be specified by a range of values, then one valid and two invalid equivalence classes should be defined. • If the input data assumes values from a set of discrete members of some domain, then one equivalence class for valid input	Listing – 2 Marks Explanation - 2 * 4 = 8 marks Advantages & Disadvantages 2 * 2.5 = 5 Marks	15	15

values and another equivalence class for invalid input values should be defined.

- **Example:** For a software that computes the square root of an input integer which can assume values in the range of 0 to 5000, there are three equivalence classes: The set of negative integers, the set of integers in the range of 0 and 5000, and the integers larger than 5000. Therefore, the test cases must include representatives for each of the three equivalence classes and a possible test set can be: {-5,500,6000}.

Boundary Value Analysis

- A type of programming error frequently occurs at the boundaries of different equivalence classes of inputs. The reason behind such errors might purely be due to psychological factors. Programmers often fail to see the special processing required by the input values that lie at the boundary of the different equivalence classes. For example, programmers may improperly use < instead of <=, or conversely <= for <. Boundary value analysis leads to selection of test cases at the boundaries of the different equivalence classes.
- **Example:**

For a function that computes the square root of integer values in the range of 0 and 5000, the test cases must include the following values: {0, - 1,5000,5001}.

BLACK BOX TESTING ADVANTAGES

- Tests are done from a user's point of view and will help in exposing discrepancies in the specifications.
- Tester need not know programming languages or how the software has been implemented.
- Tests can be conducted by a body independent from the developers, allowing for an objective perspective and the avoidance of developer-bias.

- Test cases can be designed as soon as the specifications are complete.

BLACK BOX TESTING DISADVANTAGES

- Only a small number of possible inputs can be tested and many program paths will be left untested.
- Without clear specifications, which is the situation in many projects, test cases will be difficult to design.

Tests can be redundant if the software designer/ developer has already run a test case

White box Testing

- White box testing is concerned with testing the implementation of the program.
- The intent of this testing is to exercise the different programming structures and data structures used in the program. White-box testing is also called structural testing, and we will use the two terms interchangeably.
- To test the structure of a program, structural testing aims to achieve test cases that will force the desired coverage of different structures.
- The criteria for structural testing are generally quite precise as they are based on program structures, which are formal and precise.
- There exist several popular white-box testing methodologies:
    - Statement coverage
    - branch coverage
    - path coverage
    - condition coverage
    - mutation testing
    - data flow-based testing

Advantages

	• Enforces the determination of software correctness as explained in the processing paths. • Allows performance of line coverage, it allows the tester to identify the code that has not yet been executed and test cases can be applied to these lines of code. • It ensures quality of coding work and apply to coding standards. Disadvantages • As knowledge of code and internal structure is a prerequisite, a skilled tester is needed to carry out this type of testing, which increases the cost. And it is nearly impossible to look into every bit of code to find out hidden errors, which may create problems, resulting in failure of the application			
VIII	a) **Coding standards.** Good software development organizations usually develop their own coding standards and guidelines depending on what best suits their organization and the type of products they develop. The following are some representative coding standards. • **Rules for limiting the use of global** • **Contents of the headers preceding codes for different modules:** • **Naming conventions for global variables, local variables, and constant identifiers:** • **Error return conventions and exception handling mechanisms:** **Coding guidelines** • **Do not use a coding style that is too clever or too difficult to understand:** • **Avoid obscure side effects:** • **Do not use an identifier for multiple purposes:** • **The code should be well-documented:** • **The length of any function should not exceed 10 source lines;**	4 4	8	15

	• **Do not use goto statements:** b) Code review for a model is carried out after the module is successfully compiled and the all the syntax errors have been eliminated. Normally, two types of reviews are carried out on the code of a module. These two types code review techniques are code inspection and code walk through. • Code walk through is an informal code analysis technique. In this technique, after a module has been coded, successfully compiled and all syntax errors eliminated. A few members of the development team are given the code few days before the walk through meeting to read and understand code. Each member selects some test cases and simulates execution of the code by hand (i.e. trace execution through each statement and function execution). The main objectives of the walk through are to discover the algorithmic and logical errors in the code. The members note down their findings to discuss these in a walk through meeting where the coder of the module is present. • In contrast to code walk through, the aim of code inspection is to discover some common types of errors caused due to oversight and improper programming. Following is a list of some classical programming errors which can be checked during code inspection: Use of uninitialized variables. Jumps into loops. Nonterminating loops. Incompatible assignments. Array indices out of bounds. Improper storage allocation and deallocation. Mismatches between actual and formal parameter in procedure calls. Use of incorrect logical operators or incorrect precedence among operators. Improper modification of loop variables. Comparison of equally of floating point variables, etc.	Listing – 2 Marks Explanation 5 Marks	7	

	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.			
IX	a) • Once a project is found to be feasible, software project managers undertake project planning. Project planning is undertaken and completed even before any development activity starts. Project planning consists of the following essential activities Estimating the following attributes of the project: – Project size: What will be problem complexity in terms of the effort and time required to develop the product? – Cost: How much is it going to cost to develop the project? – Duration: How long is it going to take to complete development? – Effort: How much effort would be required? The effectiveness of the subsequent planning activities is based on the accuracy of these estimations. • Scheduling manpower and other resources • Staff organization and staffing plans • Risk identification, analysis, and abatement planning • Miscellaneous plans such as quality assurance plan, configuration management plan, etc.	6	6	15

	b) COCOMO (COnstructive COst MOdel) proposed by Boehm. • Divides software product developments into 3 categories: – Organic – Semidetached – Embedded. Elaboration of Product classes • <u>Organic:</u> – Relatively small groups • working to develop well-understood applications. • <u>Semidetached:</u> – Project team consists of a mixture of experienced and inexperienced staff. • <u>Embedded:</u> – The software is strongly coupled to complex hardware, or real-time systems. For each of the three product categories: From size estimation (in KLOC), Boehm provides equations to predict: • project duration in months • effort in programmer-months • Gives only an approximate estimation: – Effort = a1* (KLOC)*a2 – Tdev = b1 * (Effort) *b2 • KLOC is the estimated kilo lines of source code,	Product class – 3 marks Effort estimation - 3 Marks Time Estimation – 3 Marks	9	

	• a1,a2,b1,b2 are constants for different categories of software products, • Tdev is the estimated time to develop the software in months, • Effort estimation is obtained in terms of person months (PMs). Development Effort Estimation • Organic : – Effort = 2.4 (KLOC)1.05 PM • Semi-detached: – Effort = 3.0(KLOC)1.12 PM • Embedded: Effort = 3.6 (KLOC)1.20PM Development Time Estimation • Organic: – Tdev = 2.5 (Effort)0.38 Months • Semi-detached: – Tdev = 2.5 (Effort)0.35 Months • Embedded: – Tdev = 2.5 (Effort)0.32 Months			
X	CMMI can be used two ways: capability evaluation and software process assessment. • Capability evaluation provides a way to assess the software process capability of an organization.	Listing – 5 Marks Explanation – 5 * 2 = 10 Marks	15	15

- software process assessment is used by an organization with the objective to improve its process capability. Thus, this type of assessment is for purely internal use.

Five level model of maturity and capability

1: Initial/Performed

2: Managed

3: Defined

4: Quantitatively Managed

5: Optimizing

Level 1: Initial.

- A software development organization at this level is characterized by ad hoc activities. Very few or no processes are defined and followed. Since software production processes are not defined, different engineers follow their own process and as a result development efforts become chaotic. Therefore, it is also called chaotic level. The success of projects depends on individual efforts and heroics. When engineers leave, the successors have great difficulty in understanding the process followed and the work completed. Since formal project management practices are not followed, under time pressure short cuts are tried out leading to low quality.

Level 2: Repeatable.

- At this level, the basic project management practices such as tracking cost and schedule are established. Size and cost estimation techniques like function point analysis, COCOMO, etc. are used. The necessary process discipline is in place to repeat earlier success on projects with similar applications. Please remember that opportunity to repeat a process exists only when a company produces a family of products.

- **Level 3: Defined.**

	At this level the processes for both management and development activities are defined and documented. There is a common organization-wide understanding of activities, roles, and responsibilities. The processes though defined, the process and product qualities are not measured. ISO 9000 aims at achieving this level • **Level 4: Managed.** • At this level, the focus is on software metrics. Two types of metrics are collected. Product metrics measure the characteristics of the product being developed, such as its size, reliability, time complexity, understandability, etc. Process metrics reflect the effectiveness of the process being used, such as average defect correction time, productivity, average number of defects found per hour inspection, average number of failures detected during testing per LOC, etc. Quantitative quality goals are set for the products. The software process and product quality are measured and quantitative quality requirements for the product are met. Various tools like Pareto charts, fishbone diagrams, etc. are used to measure the product and process quality. The process metrics are used to check if a project performed satisfactorily. Thus, the results of process measurements are used to evaluate project performance rather than improve the process. • **Level 5: Optimizing.** • At this stage, process and product metrics are collected. Process and product measurement data are analyzed for continuous process improvement. For example, if from an analysis of the process measurement results, it was found that the code reviews were not very effective and a large number of errors were detected only during the unit testing, then the process may be fine tuned to make the review more effective. Also, the lessons learned from specific projects are incorporated in to the process. Continuous process improvement is achieved both by carefully analyzing the quantitative feedback from the process	3		

measurements and also from application of innovative ideas and technologies. Such an organization identifies the best software engineering practices and innovations which may be tools, methods, or processes. These best practices are transferred throughout the organization.

CMM Level	Focus	Key Process Ares
1. Initial	Competent people	
2. Repeatable	Project management	Software project planning Software configuration management
3. Defined	Definition of processes	Process definition Training program Peer reviews
4. Managed	Product and process quality	Quantitative process metrics Software quality management
5. Optimizing	Continuous process improvement	Defect prevention Process change management Technology change management