**Section A**

**Multiple choice question**

**Attempt all questions 30\*1=30**

1. Software is defined as \_\_\_\_\_\_\_\_\_\_.
2. **Set of programs, documentation & configuration of data**
3. Set of programs
4. Instructions
5. Data
6. What is software engineering?
7. Designing a software
8. Testing a software
9. **Application of engineering principles to the design a software**
10. none
11. What is a functional requirement?
12. **Specifies the task the program must complete**
13. Specifies the task the program doesn’t complete
14. Specifies the task the program must not work
15. None
16. Attributes of good software is \_\_\_\_\_\_\_\_\_\_\_\_.
17. Development
18. Functionality
19. **Maintainability & functionality**
20. Maintainability
21. Software patch is defined as
22. Daily or routine fix
23. **Require or critical fix**
24. Emergency fix
25. None
26. Regardless of application area, project size, or complexity, software development work may be divided into three generic phases: the\_\_\_\_\_\_\_\_\_\_ phase, which focuses on what, the\_\_\_\_\_\_\_\_\_ phase, which focuses on how, and the\_\_\_\_\_\_\_\_\_ phase, which focuses on change.

i. support

ii. development

iii. definition

1. **iii , ii, i**
2. iii , i ,ii
3. i , ii , iii
4. ii, i ,ii
5. Agile Software Development is based on which of the following type?
6. Iterative development
7. Incremental development
8. **Both a and b**
9. Linear Development
10. According to IBM research, “31% of projects are abandoned before they are completed, 53% exceed their cost projections by an average of 189 percent, and 94 projects are restarted for every 100 projects.” What is the significance of these figures?
11. Lack of software ethics and understanding
12. **Management issues in the company**
13. Lack of adequate training
14. All of the above
15. The incorrect activity among the following for the configuration management of a software system is \_\_\_\_\_\_\_\_
16. Version
17. System
18. Change
19. **Internship**
20. Cohesion is a qualitative indication of the degree to which a module
21. Can be written more compactly
22. **Focuses on just one thing**
23. Is able to complete its function in a timely manner
24. Is connected to other moules and outside world
25. A design description of an object is known as a class.
26. Instance
27. Object
28. Case
29. **Both instance and object**
30. Coupling is a qualitative indication of the degree to which module
31. Can be written more compactly
32. Focuses on just one thing
33. Is able to complete its function in a timely manner
34. **Is connected to other moules and outside world**
35. The following is an example of

Struct PersonnelRecord

{

char name[100];

int socialSecurityNumber;

char department[10];

float salary;

};

1. Object
2. **Class**
3. Package
4. None
5. Which of the following does not affect the software quality and organizational performance?
6. **Marke**t
7. Product
8. Technology
9. People
10. \_\_\_\_\_\_\_\_\_ is a pidgin(simplified version of a language that develops as a means of communication between two or more groups that do not have a language in common)
11. Program design language
12. **Structure English**
13. Pseudocode
14. All of above
15. FAST stands for
16. Functional application specification technique
17. fast application specification technique
18. **Facilitated application specification technique**
19. Fault application specification technique
20. “Consider a system where, a heat sensor detects an intrusion and alerts the security company.” What kind of a requirement the system is providing ?
21. **Functional**
22. Non-functional
23. Known requirement
24. None
25. Choose the incorrect statement with respect to Non-Functional Requirement (NFR).
26. Product-oriented Approach – Focus on system (or software) quality
27. Process-oriented Approach – Focus on how NFRs can be used in the design process
28. **Quantitative Approach – Find measurable scales for the functionality attributes**
29. Qualitative Approach – Study various relationships between quality goals
30. starting from least to most important, choose the order of stakeholder.

i. Managers

ii. Entry level Personnel

iii. Users

iv. Middle level stakeholder

a. i, ii, iv, iii

b. i, ii, iii, iv

**c. ii, iv, i, iii**

d. All of the mentioned

1. Arrange the tasks involved in requirements elicitation in an appropriate manner.

i. Consolidation

ii. Prioritization

iii. Requirements Gathering

iv. Evaluation

a. iii, i, ii, iv

**b. iii, iv, ii, i**

c. iii, ii, iv, i

d. ii, iii, iv, i

1. A chemical plant system may detect excessive pressure and open a relief valve to reduce these pressures before an explosion occurs. What kind of dependability and security issue the example states?
2. Hazard avoidance
3. Damage Limitation
4. Hazard detection
5. **Hazard detection and removal**
6. \_\_\_\_\_\_\_\_\_ and \_\_\_\_\_\_\_\_\_ are the two issues of Requirement Analysis.
7. Performance, design
8. Stakeholder, developer
9. **Functional, non-functional**
10. None
11. What kind of approach was introduced for elicitation and modelling to give a functional view of the system ?
12. Object Oriented Design (by Booch)
13. **Use Cases (by Jacobson)**
14. Fusion (by Coleman)
15. Object Modeling Technique (by Rumbaugh)
16. Which of the following statements explains portability in non-functional requirements?
17. **It is a degree to which software running on one platform can easily be converted to run on another platform**
18. It cannot be enhanced by using languages, OS’ and tools that are universally available and standardized
19. The ability of the system to behave consistently in a user-acceptable manner when operating within the environment for which the system was intended
20. None of the mentioned
21. Which one of the following is not a step of requirement engineering?
22. Elicitation
23. Design
24. **Analysis**
25. Documentation
26. \_\_\_\_\_\_\_\_\_\_ executes the loop task first, then tests a condition and repeats the task until the condition fails.
27. **Repeat until**
28. Condition
29. Do while tests
30. If then else
31. MTTF stands for
32. Minimum time to failure
33. Maximum time to failure
34. **Mean time to failure**
35. None
36. Which level of CMM is for process management?
37. Initial
38. **Optimizing**
39. Defined
40. Repeatable
41. According to ISO 9001, inspection and testing comes under which management responsibility?

a. **Process control**

b. Document control

c. Control of nonconforming products

d. Servicing

1. Test cases should uncover errors like
2. **Nonexistent loop termination**
3. Comparison of different data types
4. Incorrect logical operator or precedence
5. All

**Section B**

**Short Answer Questions**

**Answer any five (5) questions out of eight (8) questions 5\*6=30**

1. Why software maintenance is also called mini -SDLC? Elaborate the answer. **(UNIT 10)**

Software maintenance refers to the process of modifying or updating existing software systems to correct defects, improve performance, or adapt to changes in user requirements or the operating environment. It is an essential part of the software development life cycle (SDLC) and typically involves four main stages:

1. Problem and modification analysis
2. Modification design
3. Modification implementation
4. Testing and maintenance release.

Software maintenance is often referred to as a mini-SDLC because it involves many of the same steps as the original software development life cycle, but on a smaller scale.

The first stage, problem and modification analysis, is similar to the requirements gathering and analysis phase in the SDLC. In this stage, the maintenance team identifies the problem, analyzes it, and determines the best course of action.

The second stage, modification design, is similar to the design phase in the SDLC. In this stage, the maintenance team designs the modifications to be made to the software system.

The third stage, modification implementation, is similar to the coding and testing phases in the SDLC. In this stage, the maintenance team implements the modifications and tests them to ensure that they work correctly.

The fourth stage, testing and maintenance release, is similar to the deployment and maintenance phase in the SDLC. In this stage, the maintenance team tests the modifications to ensure that they work correctly and then releases the modified software system.

Overall, software maintenance is often referred to as a mini-SDLC because it involves many of the same steps as the original SDLC, but on a smaller scale. It is a critical process for ensuring the continued functionality and performance of software systems, and it requires careful planning, analysis, and implementation to be successful.

1. What are the standard and guidelines for software coding? **(UNIT 9)**

**Limited use of globals:**  
These rules tell about which types of data that can be declared global and the data that can’t be.

**Standard headers for different modules:**  
For better understanding and maintenance of the code, the header of different modules should follow some standard format and information. The header format must contain below things that is being used in various companies:

Name of the module

Date of module creation

Author of the module

Modification history

Synopsis of the module about what the module does

Different functions supported in the module along with their input output parameters

Global variables accessed or modified by the module

**Naming conventions for local variables, global variables, constants and functions:**  
Some of the naming conventions are given below:

Meaningful and understandable variables name helps anyone to understand the reason of using it.

Local variables should be named using camel case lettering starting with small letter (e.g. **localData**) whereas Global variables names should start with a capital letter (e.g. **GlobalData**). Constant names should be formed using capital letters only (e.g. **CONSDATA**).

It is better to avoid the use of digits in variable names.

The names of the function should be written in camel case starting with small letters.

The name of the function must describe the reason of using the function clearly and briefly.

**Indentation:**  
Proper indentation is very important to increase the readability of the code. For making the code readable, programmers should use White spaces properly. Some of the spacing conventions are given below:

There must be a space after giving a comma between two function arguments.

Each nested block should be properly indented and spaced.

Proper Indentation should be there at the beginning and at the end of each block in the program.

All braces should start from a new line and the code following the end of braces also start from a new line.

**Error return values and exception handling conventions:**  
All functions that encountering an error condition should either return a 0 or 1 for simplifying the debugging.

On the other hand, Coding guidelines give some general suggestions regarding the coding style that to be followed for the betterment of understandability and readability of the code. Some of the coding guidelines are given below :

**Avoid using a coding style that is too difficult to understand:**  
Code should be easily understandable. The complex code makes maintenance and debugging difficult and expensive.

**Avoid using an identifier for multiple purposes:**  
Each variable should be given a descriptive and meaningful name indicating the reason behind using it. This is not possible if an identifier is used for multiple purposes and thus it can lead to confusion to the reader. Moreover, it leads to more difficulty during future enhancements.

**Code should be well documented:**  
The code should be properly commented for understanding easily. Comments regarding the statements increase the understandability of the code.

**Length of functions should not be very large:**  
Lengthy functions are very difficult to understand. That’s why functions should be small enough to carry out small work and lengthy functions should be broken into small ones for completing small tasks.

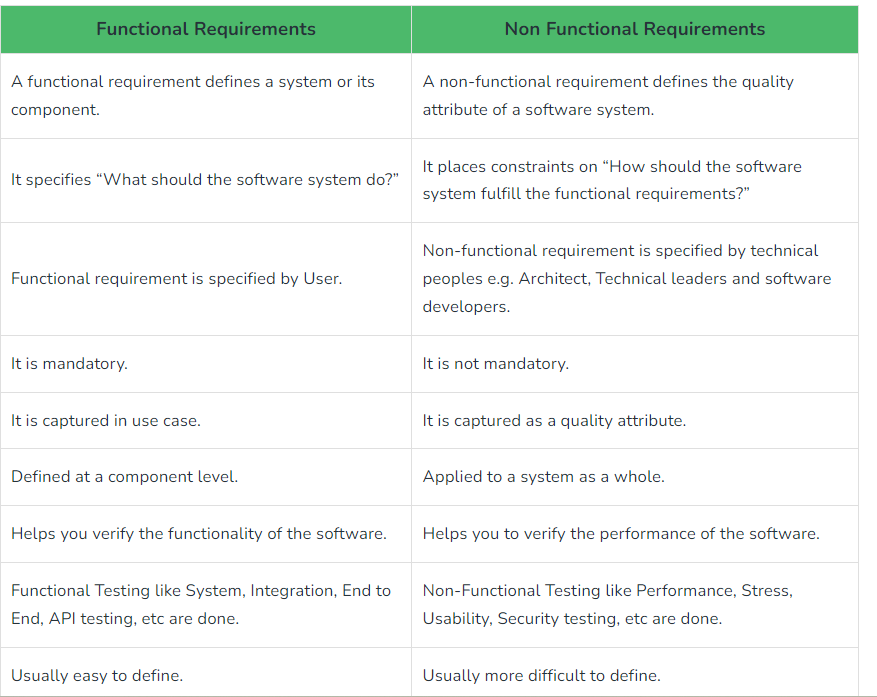
**Try not to use GOTO statement:**  
GOTO statement makes the program unstructured, thus it reduces the understandability of the program and also debugging becomes difficult.

1. What is software engineering? Explain the principles of software engineering? **(UNIT 1)**

Software engineering is a systematic and disciplined approach to the development, operation, and maintenance of software. It involves the application of engineering principles to software development, including design, analysis, testing, maintenance, and documentation.

The principles of software engineering are as follows:

1. Maintainability: Software should be easy to modify, update, and maintain over its entire lifecycle. This requires good documentation, well-structured code, and appropriate testing and debugging techniques.
2. Dependability: Software should be reliable, secure, and fault-tolerant. This requires rigorous testing and verification, as well as attention to issues such as security and privacy.
3. Efficiency: Software should be designed and implemented to operate efficiently, using minimal resources such as CPU time and memory.
4. Usability: Software should be easy to use, with a user interface that is intuitive and efficient.
5. Portability: Software should be designed and implemented to operate on a variety of platforms and environments.
6. Verification and Validation: Software should be thoroughly tested and verified to ensure that it meets the specified requirements.
7. Requirements Management: Software engineering requires a systematic approach to requirements management, including requirements elicitation, analysis, specification, and verification.
8. Process Improvement: Software engineering requires a continuous improvement process, with ongoing assessment and improvement of development processes and techniques.
9. Write the difference between functional requirement and non-functional requirement? **(UNIT 4)**

****

1. Why elicitation and analysis a difficult process in requirement engineering? Explain. **(UNIT 5)**

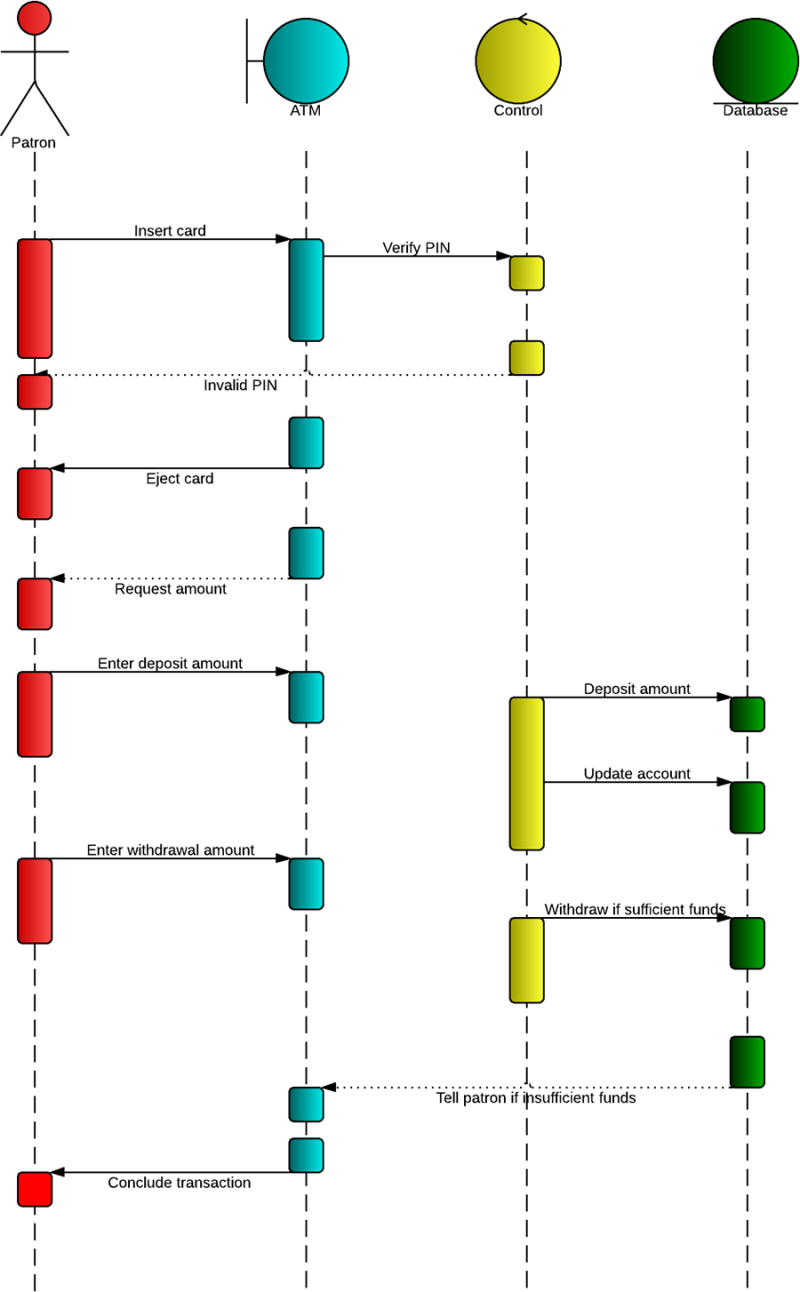
Elicitation and analysis are difficult processes in requirement engineering because of the following reasons:

1. Complex Stakeholder Environment: Requirements gathering involves multiple stakeholders, and each of them may have different requirements, expectations, and priorities. Managing these diverse requirements is challenging, and it requires a clear understanding of stakeholder expectations and needs.
2. Communication Barriers: The requirement engineer must communicate effectively with stakeholders, and this may be hindered by language barriers, technical jargon, and diverse backgrounds. Misunderstandings or gaps in communication can lead to faulty requirements.
3. Changing Requirements: Stakeholder needs and expectations may change during the development process, which can make it difficult to manage requirements. Such changes may lead to rework, delays, and cost overruns.
4. Technical Complexity: In many cases, the system being developed is complex, and this complexity can make it difficult to understand and document requirements. Requirements may also interact with each other, making it difficult to isolate individual requirements.
5. Domain Expertise: In some cases, the requirement engineer may not have domain expertise in the area being developed, making it difficult to understand and document requirements. This can result in misinterpretation of requirements, which can lead to the development of the wrong system.
6. Explain the sequence diagram using a suitable example**(UNIT 8)**

Sequence diagram is a type of UML (Unified Modeling Language) diagram that is used to depict the interaction between different objects or components of a system in a time-ordered sequence. It shows how objects collaborate with each other and exchange messages to accomplish a particular task.

The main components of a sequence diagram are:

* Actors: The external entities (users or other systems) that interact with the system being modeled.
* Objects: The components of the system being modeled that exchange messages to perform a task.
* Lifelines: A vertical dashed line representing the lifespan of an object in the system.
* Messages: The communication that takes place between the objects to achieve a specific task.



1. What do you mean by project management? Explain the project planning and project scheduling? **(UNIT 14)**

Project management is the process of leading a team to achieve specific goals and objectives within a defined scope, budget, and timeline. It involves planning, organizing, and controlling resources (such as people, time, and money) to achieve project goals.

Project planning is the process of defining the scope of the project, identifying the tasks and activities needed to complete the project, and estimating the time and resources required for each task. The main steps in project planning include:

1. Defining the project scope: This involves identifying the project objectives, deliverables, and requirements.
2. Identifying the tasks and activities: This involves breaking down the project into smaller tasks and identifying the dependencies between them.
3. Estimating time and resources: This involves estimating the time and resources required to complete each task.
4. Developing the project schedule: This involves creating a timeline for the project, including start and end dates for each task.

Project scheduling involves creating a timeline for the project, including start and end dates for each task. The project schedule should take into account the availability of resources, any dependencies between tasks, and any potential risks or issues that may arise during the project. The main steps in project scheduling include:

1. Identifying the critical path: This involves identifying the sequence of tasks that must be completed in order to finish the project on time.
2. Allocating resources: This involves assigning resources (such as people or equipment) to each task.
3. Developing the project timeline: This involves creating a timeline for the project, including start and end dates for each task.
4. Monitoring and controlling the schedule: This involves tracking progress against the project timeline, identifying any delays or issues, and taking corrective action as needed.

Effective project planning and scheduling are critical for ensuring that projects are completed on time, within budget, and to the satisfaction of stakeholders.

1. What are the test strategies for convention software? Explain. **(UNIT 11)**

Test strategies are plans for executing the testing process in order to ensure that the software meets the specified requirements and is of high quality. The following are some test strategies that can be used for conventional software:

1. Unit Testing Strategy: This strategy involves testing individual units or components of the software in isolation to ensure they function correctly. It is usually done by the developers themselves as part of the development process.
2. Integration Testing Strategy: This strategy involves testing the interaction between different modules or components of the software to ensure they work correctly together. It is usually done after the unit testing is completed.
3. System Testing Strategy: This strategy involves testing the entire software system as a whole to ensure it meets the specified requirements and works as intended. It includes testing the software's functionality, performance, security, and usability.
4. Acceptance Testing Strategy: This strategy involves testing the software from the user's perspective to ensure it meets their expectations and requirements. It is usually done by the end-users or clients before the software is released.
5. Regression Testing Strategy: This strategy involves retesting the software after modifications or enhancements have been made to ensure that the changes did not introduce any new defects or issues.
6. Exploratory Testing Strategy: This strategy involves testing the software in an unstructured and informal manner to uncover defects or issues that may have been missed by other testing strategies.

**Section C**

**Comprehensive question answer**

**Answer any two (2) questions out of three (3) questions 2\*20=40**

1. Case

Nepal is currently facing a problem regarding the waste control problem. Now you are given a task to create a software regarding the waste management system. Now prepare a proposal to Kathmandu Metropolitan Office regarding the development of the software assuming all the necessary itinerary that are required for development of the software. Also include the objective for creating that software and time frame work that is required for completing the software. **(From all chapter)**

1. Explain the concept of module and modularization. What is the necessity of architectural design in software engineering? Why a good user interface is required in software design? Explain **(UNIT 7) (6+6+8)**

In software engineering, a module refers to a self-contained unit of code that performs a specific task. It can be a function, class, or even a single line of code. A module can be reused in different parts of a software system or even in different software systems. Modularization is the process of dividing a software system into smaller, independent modules that can be developed, tested, and maintained separately. The main goal of modularization is to make a software system more manageable, flexible, and easy to maintain by dividing it into smaller, well-organized modules.

Architectural design is a crucial phase in software engineering as it lays the foundation for the entire software system. The main necessity of architectural design are:

1. Provides a high-level overview: The architectural design provides a high-level overview of the system and helps to understand the software's components, their functions, and how they interact with each other.
2. Promotes maintainability: A well-designed architecture makes the software system more maintainable. By separating the system into modules, it becomes easier to make changes to a specific module without affecting the entire system. This makes the maintenance process more manageable and efficient.
3. Facilitates scalability: A well-designed architecture allows for the easy addition of new features or modules without affecting the existing system's functionality. This scalability is necessary as the software needs to adapt to the changing needs of the users and business.
4. Enables reuse: A modular architecture allows for the reuse of modules across different systems, reducing development time and cost.
5. Enhances performance: An optimized architecture can improve the software system's performance by reducing redundant operations and minimizing resource utilization.
6. Ensures reliability: A well-designed architecture ensures the reliability of the software system by reducing the likelihood of errors and providing fault tolerance mechanisms.

Here are some reasons why a good user interface is important in software design:

1. Improved usability: A good UI ensures that the software is easy to use and navigate, reducing the likelihood of user errors and improving overall efficiency.
2. Better user experience: A well-designed UI can make the user experience more enjoyable and engaging, leading to increased user satisfaction.
3. Increased productivity: A good UI can help users complete tasks more quickly and easily, which can increase productivity and save time.
4. Increased adoption: A well-designed UI can make software more appealing and user-friendly, increasing the likelihood that users will adopt it.
5. Reduced training costs: A software application with a good UI requires less training for new users, reducing the time and costs associated with training and support.
6. What is software reliability matrices? Explain different software reliability matrices. Why CMM model is used in software engineering? Explain the process of organization getting an ISO certification. **(UNIT 6,14,13) (2+7+6+5)**

oftware reliability metrics are used to measure the ability of software to perform its intended function without failure, over a specified period, and under stated conditions.

Different software reliability metrics are:

1. Mean Time Between Failures (MTBF): It measures the average time between two consecutive failures in the software. It is calculated by dividing the total operating time by the number of failures. MTBF helps to determine the software's reliability and is often used as a benchmark for comparing different software products.
2. Failure Rate: It is the number of failures per unit of time. It helps to estimate the probability of failure of the software within a given time period.
3. Mean Time To Failure (MTTF): It is the average time between the start of software operation and the first failure. It is a measure of software's reliability.
4. Mean Time To Repair (MTTR): It is the average time taken to repair the software after a failure. It is a measure of software's maintainability.
5. Availability: It is the percentage of time the software is available for use. It is calculated by dividing the total time the software is available by the total time it is required.
6. Mean Residual Life (MRL): It is the average time between a failure and the end of the software's useful life. It helps to predict when the software may fail and when it should be replaced.
7. Defect Density: It is the number of defects per unit of software size. It helps to identify the areas of the software that need improvement and the effectiveness of the testing process.

The Capability Maturity Model (CMM) is a model that provides a framework for software development process improvement. It is used to evaluate and improve the maturity of software development processes within an organization.

The use of CMM has become popular in software engineering because it offers a number of benefits. Some of these benefits include:

1. Standardization: CMM provides a standard framework for software process improvement that can be used across different organizations.
2. Process Improvement: CMM provides a way to measure the maturity of software development processes and identify areas for improvement. This helps organizations to continuously improve their software development processes.
3. Quality Improvement: By improving the software development process, CMM helps organizations to produce high-quality software that meets the needs of the end-users.
4. Cost Reduction: By improving the software development process, CMM helps organizations to reduce the cost of software development and maintenance.
5. Risk Management: CMM helps organizations to manage the risks associated with software development by providing a structured framework for process improvement.

The process of an organization getting an ISO certification involves the following steps:

1. Determine the ISO standard: The first step is to determine which ISO standard(s) the organization wants to be certified against. For instance, an organization may choose ISO 9001 for quality management or ISO 27001 for information security management.
2. Develop a quality management system: Once the standard is determined, the organization needs to develop a quality management system (QMS) that meets the requirements of the chosen standard. This system will describe the processes and procedures that the organization will follow to meet the requirements of the standard.
3. Implement the QMS: The next step is to implement the QMS throughout the organization. This involves training employees on the new processes and procedures, making any necessary changes to the organization's operations, and ensuring that everyone is following the new system.
4. Perform an internal audit: Before seeking certification, the organization needs to perform an internal audit of its QMS. The purpose of this audit is to identify any areas that need improvement before an external auditor assesses the system.
5. Select a certification body: The organization needs to select a certification body accredited to perform audits to the standard it has chosen. The certification body will review the organization's QMS to ensure that it meets the requirements of the standard.
6. External audit: The certification body will then perform an external audit of the organization's QMS. This audit will typically involve a review of the QMS documentation, interviews with employees, and observation of processes in action.
7. Receive the certification: If the organization's QMS meets the requirements of the standard, it will receive an ISO certification. This certification is typically valid for a certain period of time, after which the organization will need to be recertified.

Best of luck