MSS 🐱-2 Important Questions

https://vivekkj.in & https://linkedin.com/in/derick-davies-007813152/

SHORT QUESTIONS

1. **Analyze the importance of postmortem evaluation (Module 3)**

- **Post-mortem evaluation** (PME) is a mechanism to determine what went right and what went wrong when a software engineering process is applied in a specific project.
- Unlike an FTR (Formal Technical Review) that focuses on a specific work product, a PME examines the entire software project, focusing on both " excellences (i.e, achievements and positive experiences) and challenges (problems and negative experiences)".
- PME is attended by members of the software team and stakeholders. The aim is to identify excellences and challenges and to extract lessons learned from both.
- To see if quality control is effective, gather specific information about how reviews are done and what errors are found. This data helps evaluate how well your reviews work. Studies show that doing these checks gives a lot in return for the effort.

2. **Specify the importance of risk management in software project management. (Module 4)**

- Effective risk management helps you to recognize the strengths, weaknesses, opportunities, and dangers that your project faces.
- It defines how you will handle any risks to guarantee that you can recognize, minimize, or eliminate problems as they arise to ensure the success of your project.
- Risk management is crucial to successful project managers since accomplishing project goals is dependent on planning, preparation, results, and evaluation, all of which contribute to achieving project goals.
- Effective risk management ensures that issues do not result in budget or schedule delays.
- Because of the inherent uncertainties in software development, software risk management is critical.

3. **Identify any four types of requirements that may be defined for a software system (Module 3** it seems**)**

System requirements, in general, are meant to communicate the functions that the system should provide.

- **User requirements**: User requirements include natural language statements and system diagrams showing the services and operational limitations of the system.
- **System requirements**: A structured document describing the system's functions, services, and operational limitations in detail. It might be written in an agreement between the client and the contractor.
- **Functional requirements**: These are the services the system should deliver, how it should react to specific inputs, and how it should behave in specific situations.
- **Non-functional requirements**: Restrictions on the system's services or functions, such as time limitations, development process constraints, standards, and so on.

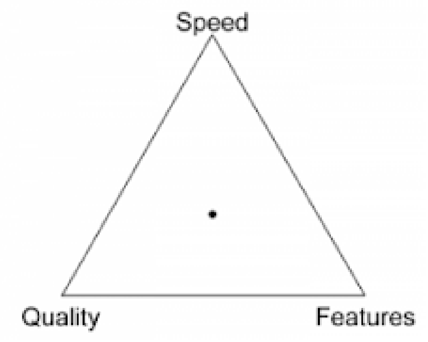
4. Compare White Box testing and Black box testing. (Module 3) (Dec 2022)

Black Box Testing	**White Box Testing**
It's a type of software testing in which the program's internal structure is hidden and nothing is known about it.	It is a method of software testing in which the tester is familiar with the program's internal structure.
Frequently done by software testers.	Frequently done by developers.
It is not necessary to have any prior experience with implementation.	It is necessary to have prior experience with implementation.
Also known as "outer" or "external" software testing.	Also known "Internal" software testing.
It is a software functional test.	It is a software structural test.
Testing can begin once the requirement specification paper has been completed.	Testing can begin once the detailed design document has been completed.

5. Discuss the software quality dilemma. (Module 5) (Dec 2022)

The problem with quality is essentially twofold:

- Build low-quality software, and no one will want to buy or use it.
- Spend too much time on quality standards, and no one will buy or utilize it since it is either too expensive or still under development.
- The pace with which software is released, the fidelity of its features, and the quality of the code base are all important aspects of any software project. In an ideal world, it should be in the middle, as shown in the following figure:



- As we come closer to one goal, we tend to move away from the others, as shown by the triangle. For eg: When it comes to speed, quality and feature authenticity are sacrificed to some extent.

6. Describe the different levels of CMMI model. (Module 5) (Dec 2021)

CMMI (Capability Maturity Model Integration) is a comprehensive process meta-model based on system and software engineering capabilities. It assesses an organization's process maturity levels across different stages. Each area (like project planning or requirements management) is evaluated against specific goals and practices and is rated according to the following capability levels in CMMI:

("IPMDQO: **I P**romise **M**y **D**og **Q**uit **O**rdering!")

- Level 0: **I**ncomplete → The process area isn't performed or doesn't achieve all goals and objectives defined by CMMI for level 1 capability.
- Level 1: **P**erformed → All defined goals in the process area are met.
- Level 2: **M**anaged → All capability level 1 criteria have been satisfied.
- Level 3: **D**efined → All capability level 2 criteria have been satisfied.
- Level 4: **Q**uantitatively managed → All capability level 3 criteria have been satisfied.
- Level 5: **O**ptimized → All capability level 4 criteria have been satisfied.

7. List out and explain fundamental project management activities. (Module 4) (Dec 2021)

The fundamental project management activities that are common to all organizations:

1. **Project planning** → Project managers are responsible for planning, estimating, and scheduling project development and assigning people to tasks.
2. **Risk management** → Project managers have to assess the risks that may affect a project, monitor these risks, and take action when problems arise.
3. **People management** → Project managers are responsible for managing a team of people. They have to choose people for their team and establish ways of working that lead to effective team performance.
4. **Reporting** → Project managers are usually responsible for reporting on the progress of a project to customers and to the managers of the company developing the software.
5. **Proposal writing** → The initial step of a software project may involve writing a proposal to secure a work contract. This proposal outlines the project's goals, methods, cost estimates, and schedules. It aims to justify why a specific team or organization should be awarded the project contract.

8. Explain various open-source licenses (Module 3)

The following are the various open source licenses models:

1. **The GNU General Public License (GPL)**. This is a so-called reciprocal license that simplistically means that if you use open-source software that is licensed under the GPL license, then you must make that software open source.

2. **The GNU Lesser General Public License (LGPL)**: This is a variant of the GPL license where you can write components that link to open-source code without having to publish the source of these components. However, if you change the licensed component, then you must publish this as open source.

3. **The Berkley Standard Distribution (BSD) License**: This is a nonreciprocal license, which means you are not obliged to re-publish any changes or modifications made to open-source code. You can

include the code in proprietary systems that are sold. If you use open-source components, you must acknowledge the original creator of the code.

LONG QUESTIONS

1. **Differentiate various integration testing methods (Module 3) (7)(Dec 2022)**

Problems may arise when the modules are put together or interfaced. For example: one component may affect another. This leads to the need for integration tests to ensure the problems are resolved. The following are the various kinds of integration testing:

1. **Top-Down Integration Testing**:
- It is an incremental approach where modules are integrated by moving downward through the control hierarchy.
- Integration begins with the main control module (main program) which is used a test drivers while the remaining sub-modules are taken as stubs for processing.
- The modules are integrated based on any of the following approaches:
  - a) Breadth First Integration: Modules in the same level are integrated and tested and then moves to the next level.
  - b) Depth First Integration: Modules onto the left path are integrated and tested first. The major control is backtracked and modules in the right path are integrated and tested.

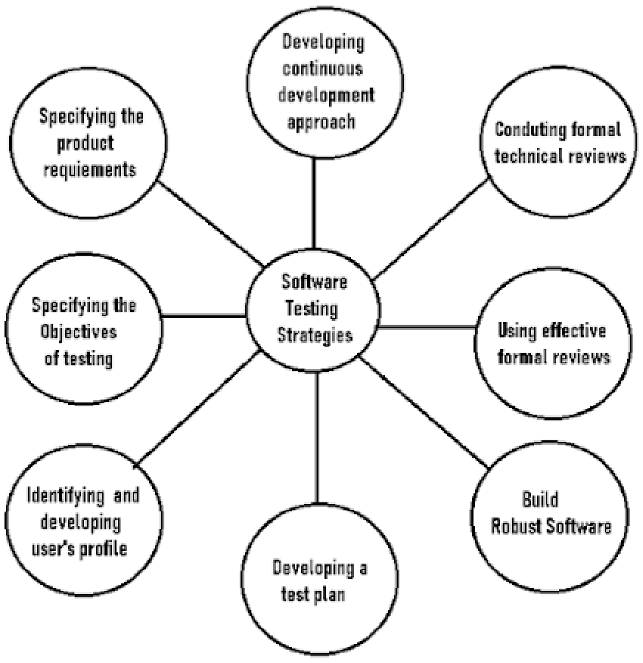
2. **Bottom-Up Integration Testing**:
- It is an incremental approach where atomic module are integrated by moving upward through the control hierarchy.
- The need for stubs are eliminated since the processing required for components subordinate to the same level are available.
- Here:
  1. Low level components are combined into clusters.
  2. A driver is written to coordinate test case input and output for the cluster.
  3. The cluster is tested.
  4. Drivers are removed and clusters are combined moving upward in the program structure.

3. Regression Testing:
- Each time a new module is integrated,
  1. New data flow paths are established
  2. New I/O may occur
  3. New Control logic is invoked
- Due to these changes, the integration may cause problems with functions that previously worked flawlessly. This is identified using regression testing.
- Regression testing is the re-execution of some subset of tests that have already been conducted to ensure that changes have not introduced unwanted side effects.
- Regression testing may be conducted manually, by re-executing a subset of all test cases or using automated capture/playback tools (for subsequent playback and comparison).
- Regression testing contains 3 different classes of test cases:
  1. A representative tests that will exercise all software functions.
  2. Additional tests that focus on software functions that are likely to be affected by the change.
  3. Component Tests that focus on the software components that have been changed.

4. Smoking Testing:

- Smoke testing is an integration testing approach that is commonly used when product software is developed.
- It determines whether the deployed software build is stable or not.
- Software components that have been translated into code are integrated into a build (Data files + libraries + reusable modules).
- Smoke tests is designed to expose errors that will keep the build from properly performing its function.
- The build is integrated with other builds, and the entire product is smoke tested daily.

2. Evaluate various software testing strategies **(Module 3)**


Software Testing Strategies
Developing continuous development approach
Conduting formal technical reviews
Specifying the product requiements
Specifying the Objectives of testing
Using effective formal reviews
Identifying and developing user's profile
Developing a test plan
Build Robust Software

- Software testing is used to determine whether the software contains any defaults or errors so that the errors can be decreased or removed, hence improving the software's quality.
- The basic goal is to build tests that detect many types of errors in a systematic manner with minimal time and effort, allowing for faster software development.
- Before any testing can begin, the product's needs must be identified and specified in a quantitative manner. There are various qualities of software, such as maintainability, likelihood, usability etc. All of these characteristics should be given in a specific order in order to receive accurate test results.
- Define the test goals in a clear and precise manner. There are several testing objectives, such as to improve effectiveness, find point of failure, determine cost of defects etc. All of these goals should be stated explicitly in the test plan.
- Identify the user's category and creat a profile for each user for the software. The interactions and communication among different kinds of users and the system to achieve the goal should be described in use case diagrams.
- Creating a test plan with a focus on rapid-cycle testing to add value. Rapid Cycle Testing (RCT) is a form of test that enhances quality by detecting and quantifying any adjustments that must be made to improve the software development process.
- Create robust software that is designed to test itself. The software should be able to detect and identify various types of faults. (Automated and regression testing should identify whether any changes in the code have adverse effects on its functionality)

- Use effective formal reviews as a filter before testing. Formal technical reviews are a tool for detecting faults that have not yet been noticed. Effective technical reviews completed prior to testing lower the amount of testing work and time.
- Conduct formal technical reviews for detection of any gaps in the testing strategy.
- Developing a continuous development methodology for the testing process. This will help in regulating quality during software development.

3. **Differentiate the formal and informal review techniques (Module 3) (7)(Dec 2022)**

Aspect	**Formal Review**	**Informal Review**
Purpose	Used to find defects in system software	Used to share knowledge, improve understanding of the system
Process	Structured, follows specific steps and guidelines. Has clear entry and exit criteria.	Flexible, no strict steps or rules. More discussion-based.
Participants	Dedicated team with specific roles and a leader.	Open to anyone interested, no specific roles.
Scope	Focuses on specific system parts or milestones.	Covers various system aspects without limits.
Documentation	Requires formal reports and detailed follow-ups.	Might not need formal documentation.
Time and Cost	Time-consuming, more expensive due to formalities.	Quicker, less costly as it's more flexible.
Benefits	Finds and fixes system problems systematically.	Encourages teamwork and faster responses.

4. **What are design patterns? What are the essential elements of design pattern? (Module 3) (7) (Dec 2022)**

Design patterns are reusable solutions to common software design problems that have been proven to be effective over time. They are templates for solving a particular type of problem that can be adapted to fit the specific needs of a particular software system.

Design patterns are often used to improve the quality, reliability, and maintainability of software systems.

The essential elements of design patterns include:

- **Pattern name**: The pattern should have a descriptive and recognizable name that conveys its purpose and function.
- **Problem description**: The pattern should describe the problem it solves, including the context in which it arises, the constraints that apply, and the goals that must be achieved.
- **Solution description**: The pattern should describe the solution to the problem, including the key elements, their relationships, and their interactions.
- **Consequences**: The pattern should describe the benefits and drawbacks of the solution, including its impact on other parts of the system, its maintainability, and its performance.

- **Examples**: The pattern should include one or more examples that illustrate how it can be used in practice, including code samples, diagrams, and other supporting material.

Design patterns are typically categorized into three main types:

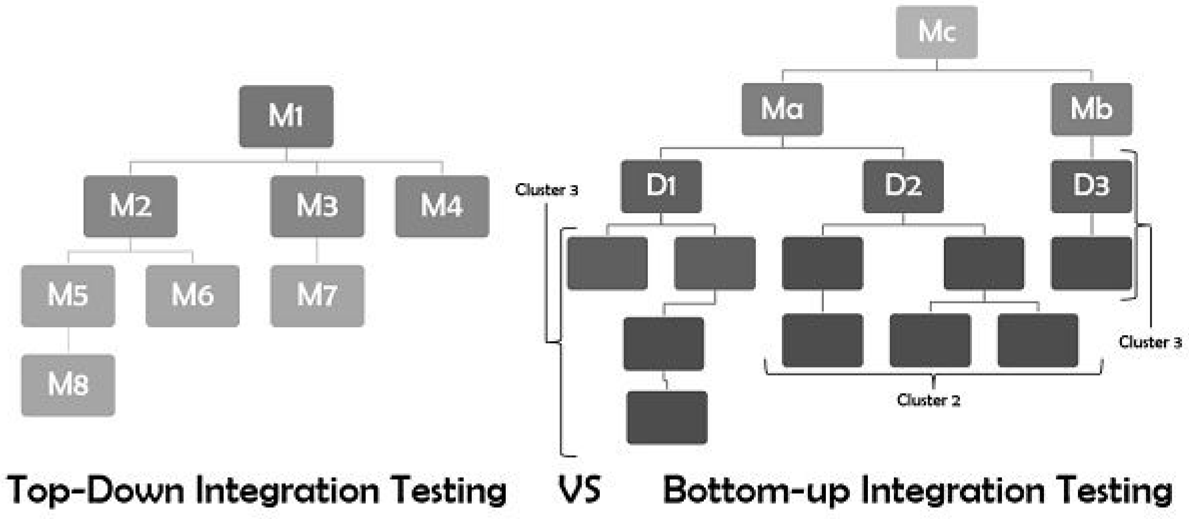
1. **Creational patterns**: These patterns are used to create objects or classes that are flexible, reusable, and scalable.
2. **Structural patterns:** These patterns are used to define the relationships between objects or classes that are flexible, reusable, and scalable.
3. **Behavioral patterns:** These patterns are used to define the interactions between objects or classes that are flexible, reusable, and scalable.

By using design patterns, software developers can create software systems that are more modular, reusable, and maintainable, leading to higher quality software and more efficient development processes.

5. Differentiate between Top-down and Bottom-up Integration testing methods with suitable diagrams.

(Module 3) (7) (Dec 2022)

Aspect	**Top-Down Integration**	**Bottom-Up Integration**
Approach	Starts testing from the top (main module) and proceeds towards lower-level modules.	Starts testing from the bottom (lower-level modules) and proceeds towards the top (main module)
Module Integration	Integrates higher-level modules first, simulating integration downwards.	Integrates lower-level modules first, simulating integration towards the main module.
Cluster	Does not employ clusters.	Groups low-level modules into clusters, building up towards larger units.
Dependencies	Uses stubs (simulated modules) for lower-level modules that are not yet developed.	Requires drivers (simulated main modules) for higher-level modules that are not yet ready. These drivers test clusters.
Control Flow	Controlled by the main module, from general to specific functionalities	Controlled by lower-level modules, from specific to general functionalities.



6. Differentiate COCOMO estimation models. (Module 4)

- The best known algorithmic cost modeling technique and tool is the COCOMO II model.
- COCOMO II is a freely available model that is supported with open-source tools.
- It comes with several submodels which produce high detailed estimates.
- These submodels that are part of the COCOMO II model are**:**

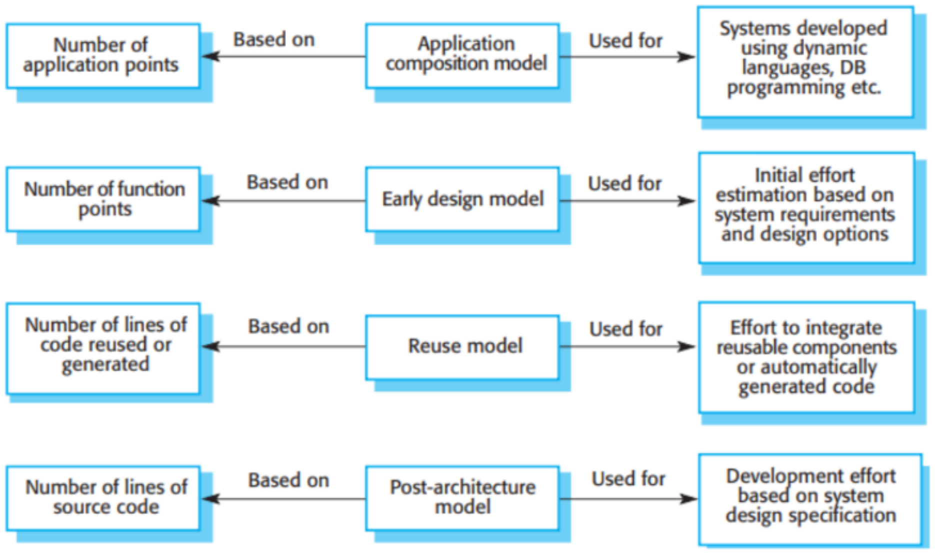

Number of application points
Based on
Application composition model
Used for
Systems developed using dynamic languages, DB programming etc.
Number of function points
Based on
Early design model
Used for
Initial effort estimation based on system requirements and design options
Number of lines of code reused or generated
Based on
Reuse model
Used for
Effort to integrate reusable components or automatically generated code
Number of lines of source code
Based on
Post-architecture model
Used for
Development effort based on system design specification

Fig: COCOMO estimation models

1. <u>The Application Composition Model</u>:

- This model supports the estimation of effort required for prototyping projects where the software is developed by composing existing components.
- It is based on an estimate of weighted application points divided by a standard estimate of application point productivity.
- Formula for effort computation:

- Formula for effort computation for system prototypes is:

$$PM = (NAP \times (1 - \%reuse/100)) / PROD$$

Where,

- **PM**: the effort estimate in person-months.
- **NAP**: the total number of application points in the delivered system.
- **%reuse**: an estimate of the amount of reused code in the development.
- **PROD**: the application-point productivity

- PROD depends on developer experience and software capabilities used for development.
- NAP is based on the following estimates:
  - No.of separate screens
  - No. of reports
  - No.of modules in imperative programming language
  - No.of lines of scripting language or DB programming code.

2. <u>The Early Design Model:</u>

- It is used during the early stages of the project development.
- This model assumes that user requirements have been agreed and initial stages of the system design process are underway.
- Early design estimates are most useful for exploring different ways of implementing the user requirements.
- Formula for effort computation is:

$$PM = 2.94 \times Size^{(1.1 \text{ to } 1.24)} \times M$$
$$M = PERS \times PREX \times RCPX \times RUSE \times PDIF \times SCED \times FSIL$$

Where,

- **PERS**: personnel capability
- **PREX**: personnel experience
- **RCPX**: product reliability and complexity
- **RUSE**: reuse required
- **PDIF**: platform difficulty
- **SCED**: schedule
- **FSIL**: support facilities

- Size is expressed in KSLOC -> Kilo Source Lines of Codes
- Multiplier M is based on 7 project and process attributes.

3. <u>The Reuse Model:</u>

- This model is used to estimate the effort required to integrate reusable or generated code.
- COCOMO II considers 2 types of reused code:
  1. Black-box code:
     - These are codes that can be reused without making changes to it.
     - Development Effort = 0
  2. White-box code:

- There are codes that has to be adapted to integrate it with new code.
- Development Effort != 0 as changes has to be manually made.

- The formula to estimate the reuse effort is:

$$\text{Effort} = A \times \text{ESLOC}^{B} \times M$$

Where,

- **A, B, and M** have the same values as used in the early design model.

$$\text{ESLOC} = (\text{ASLOC} \times (1 - \text{AT}/100) \times \text{AAM})$$

Where,

ESLOC = Equivalent Source Lines of Code
ASLOC = Actual Source Lines of Code
AT = % of reused codes
AAM = Adaptation Adjustment Multiplier

4. <u>The Post-architecture Model:</u>

- It is the most detailed model in COCOMO II
- It is used when you have an initial architectural design for the system.
- Formula for effort estimation is based on standard formula:

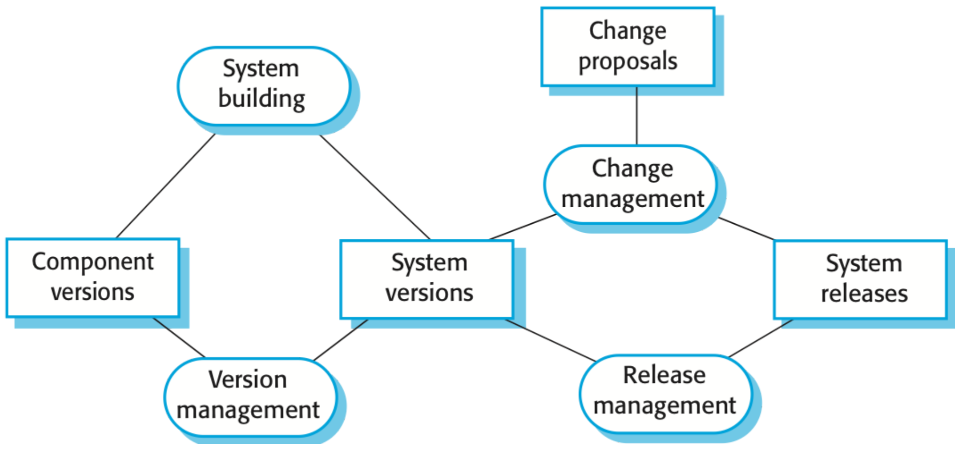
$$\text{Effort} = A \times \text{Size}^{B} \times M$$

- Overall Size is estimated based on the following 3 code size estimates:
  1. SLOC
  2. Reuse costs based on ESLOC
  3. No.of lines code that may change due to changing system requirements.
- The values of B is based on:
  1. Architecture Resolution
  2. Development Flexibility
  3. Precedentedness
  4. Team Cohesion
  5. Process Maturity

7. Define software configuration management. Explain different activities involved in configuration management with a neat diagram. (Module 4) **(10)(Dec 2021)**

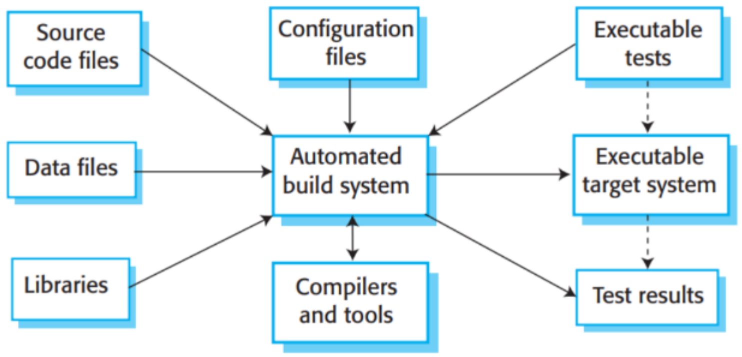
Configuration management

- Configuration management (CM) is concerned with the policies, processes, and tools for managing changing software systems.
- The configuration management of a software system product involves four closely related activities:



- **<u>Version Management</u>**
  - The process of keeping track of different versions of software components and the systems in which these components are used.
  - It also involves ensuring that changes made by different developers to these versions do not interfere with each other.
  - There are 2 types of modern version control system:
    - Centralized Systems
    - Distributed Systems

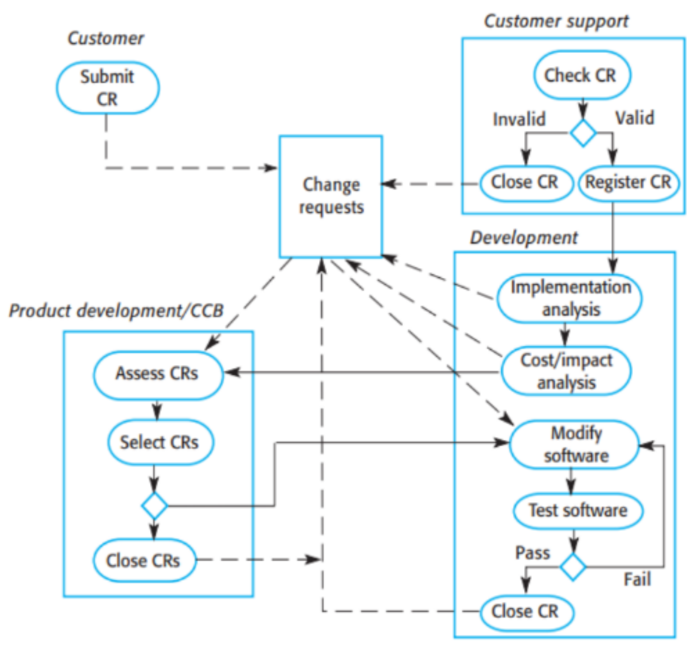
- **<u>System Building</u>**
  - The process of creating a complete, executable system by compiling and linking the system components, external libraries, configuration files, and other information.
  - System building involves assembling a large amount of information about the software and its operating environment. Therefore,we use an automated build tool to create a system build.



- 3 different system platforms involved in building process:
  - Development system → which includes development tools such as compilers and source code editors.
  - Build server → which is used to build definitive, executable versions of the system.
  - Target environment → which is the platform on which the system executes.

- <u>**Change Management**</u>

  - Change management is a structured process that handles proposed system changes by carefully assessing them, ensuring they're controlled and organized.
  - It prioritizes cost-effective adjustments and can use adaptable tools, from simple issue trackers to more complex software, without enforcing rigid processes.



  - Tools to support change management may be a relatively simple issue or bug tracking systems.
  - Issue tracking systems allow anyone to report a bug or make a suggestion for a system change, and they keep track of how the development team has responded to the issues.

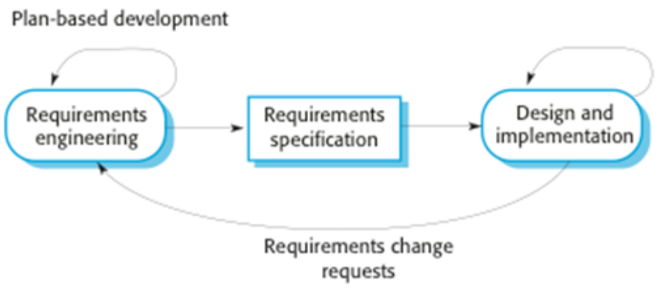
- **Release Management:**
  - A system release is a version of a software system that is distributed to customers.
  - 2 types of releases in mass-market softwares are:
    - **Major releases** → which deliver significant new functionality. Customers usually have to pay for them.
    - **Minor releases** → which repair bugs and fix customer problems that have been reported. These are usually distributed free of charge.

  - A software release may also include:
    - **Configuration files** → defining how the release should be configured for particular installations;
    - **Data files** → such as files of error messages
    - **An installation program** → used to help install the system on target hardware;
    - **Electronic and paper documentation** → describing the system;
    - **Packaging and associated publicity** → that have been designed for that Release.

    - Release creation is the process of creating the collection of files and documentation that include all components of the system release.

8. **Explain plan-driven development and project schedule. (Module 4)** **(7)(Dec 2022)**

Plan Driven Development

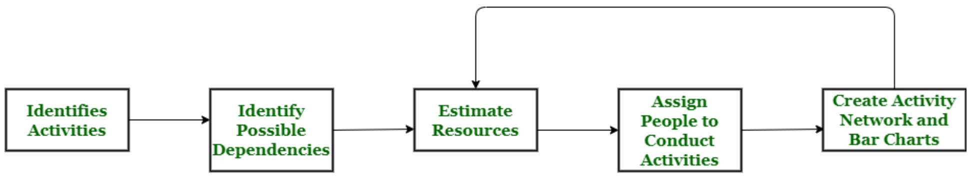

Plan-based development
Requirements engineering
Requirements specification
Design and implementation
Requirements change requests

- Plan-driven or plan-based development is an approach to software engineering where the development process is planned in detail.
- A project plan is created that records the work to be done, who will do it, the development schedule, and the work products.
- Managers use the plan to support project decision making and as a way of measuring progress.
- The problem with plan-driven development is that early decisions have to be revised because of changes to the environments in which the software is developed and used. Delaying planning decisions avoids unnecessary rework.
- The arguments in favor of a plan-driven approach are that early planning allows organizational issues (availability of staff, other projects, etc.) to be taken into account. Potential problems and dependencies are discovered before the project starts, rather than once the project is underway.

Project Schedule

- Project scheduling is the process of deciding how the work in a project will be organized as separate tasks, and when and how these tasks will be executed.
- Project Scheduling involves:
    - Estimating calendar time to complete the tasks
    - Estimating effort required
    - Estimating hardware and software requirements
    - Assigning people to tasks

Project Scheduling Process:


Identifies Activities
Identify Possible Dependencies
Estimate Resources
Assign People to Conduct Activities
Create Activity Network and Bar Charts

Project Scheduling Process

Schedule Presentation

- Project schedules may simply be documented in a table or spreadsheet showing the tasks, estimated effort, duration, and task dependencies.
- Two types of visualization are commonly used:
  - **Calendar-based bar charts** show who is responsible for each activity, the expected elapsed time, and when the activity is scheduled to begin and end.
  - **Activity networks** show the dependencies between the different activities.
- Project activities are the basic planning element. Each activity has:

  1. A duration in calendar days or months
  2. An effort estimate, which shows the number of person-days or person-months to complete the work;
  3. A deadline by which the activity should be complete; and
  4. A defined endpoint, which might be a document, the holding of a review meeting, the successful execution of all tests, or the like

Milestones and deliverables:

Milestones are short reports that are used for progress reporting, whereas deliverables are more substantial project outputs such as a requirements document or the initial implementation of a system.

9. **Compare CMMI and ISO 9001:2000.(Module 5)** **(7)(Dec 2022)**

CMMI (Capability Maturity Model Integration)

CMMI is a process improvement framework that helps organizations enhance their product and service development processes. It was developed by the Software Engineering Institute (SEI) at Carnegie Mellon University and has evolved to encompass various disciplines beyond software engineering. CMMI defines a set of best practices and guidelines that organizations can adopt to improve their capabilities, achieve better performance, and optimize their processes.

Key Characteristics of CMMI

- Maturity Levels: CMMI is organized into five maturity levels, each representing a stage of proces improvement. These levels are: Initial, Managed, Defined, Quantitatively Managed, and Optimizing.
- Process Areas: CMMI defines specific process areas that address different aspects of an organization’s activities. Some examples include Requirements Management, Project Planning, Configuration Management, and Process and Product Quality Assurance.
- Continuous Improvement: CMMI encourages organizations to focus on continuous improvement by implementing and refining processes over time. It emphasizes the importance of data-driven decision-making and performance measurement.
- Applicability: CMMI is commonly used in industries such as software development, systems engineering, and project management, but its principles can be adapted to other sectors as well.

ISO 9001

ISO 9001 is an international standard for quality management systems (QMS) established by the International Organization for Standardization (ISO). It provides a systematic approach for organizations to ensure the consistent delivery of products and services that meet customer requirements and comply with relevant regulations.

Key Characteristics of ISO 9001

- Process Approach: ISO 9001 adopts a process-based approach to quality management. It emphasizes the identification, understanding, and management of key processes within an organization to achieve desired outcomes.

- Customer Focus: ISO 9001 places significant emphasis on meeting customer needs and enhancing customer satisfaction. It requires organizations to monitor customer feedback and act upon it for continual improvement.
- Risk-based Thinking: The standard encourages organizations to identify and address potential risks that could impact the achievement of quality objectives. It promotes a proactive approach to risk management.
- Applicability: ISO 9001 is applicable to organizations of all sizes and industries, making it one of the most widely adopted and recognized quality management standards globally.

The following table provides a detailed comparison between CMMI and ISO 9001:

Aspect	**CMMI (Capability Maturity Model Integration)**	**ISO-9001**
Focus	Process Improvement and Capability Development	Quality Management System (QMS)
Structure	Maturity Levels and Process Areas	Process-based Approach and Clauses
Development Phases	Evolved from CMM and CMM-SW to CMMI	Originally Released in 1987 and Revised Periodically
Applicability	Software Development, Systems Engineering, etc.	All Types of Organizations and Industries
Customer Satisfaction Focus	Emphasized within specific process areas	Emphasized as a key principle
Continuous Improvement	Strong focus on continuous improvement and learning	Encouraged through the use of the PDCA cycle
Risk Management	Not a primary focus but may be addressed indirectly	Risk-based thinking explicitly required
Third-party Certification	CMMI is typically not used for external certification	ISO 9001 can be externally audited and certified

10. Explain the Software Risk management process with the help of a neat diagram. (Module 4) (7) (Dec 2021, Dec 2022)

Risk Management Process

- **Risk identification:** You should identify possible project, product, and business risks.
- **Risk analysis:** You should assess the likelihood and consequences of these risks.
- **Risk planning:** You should make plans to address the risk, either by avoiding it or by minimizing its effects on the project.
- **Risk monitoring:** You should regularly assess the risk and your plans for risk mitigation and revise these plans when you learn more about the risk.

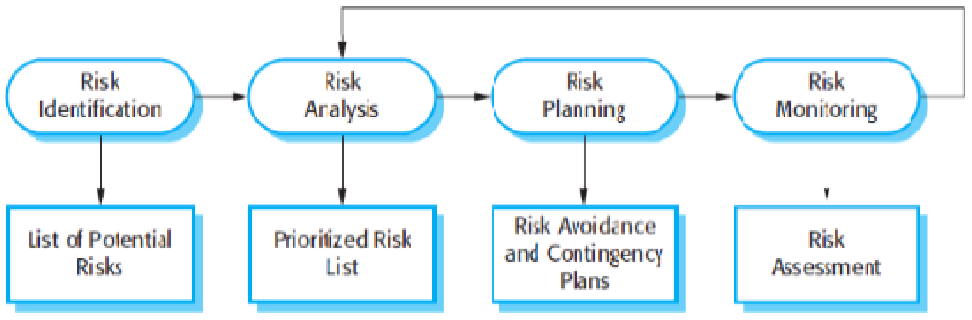


Fig 1. Risk management process

- The risk management process is an iterative process that continues throughout a project. Once you have drawn up an initial risk management plan, you monitor the situation to detect emerging risks. As more information about the risks becomes available, you have to re-analyze the risks and decide if the risk priority has changed. You may then have to change your plans for risk avoidance and contingency management.
    - **Risk Identification:**
        - Risk identification is the first stage of the risk management process. It is concerned with identifying the risks that could pose a major threat to the software engineering process, the software being developed, or the development organization.
        - Risk identification may be a team process in which a team gets together to brainstorm possible risks. Alternatively, project managers may identify risks based on their experience of what went wrong on previous projects.
        - Six types of risk may be included in a risk checklist:

            1. Estimation risks arise from the management estimates of the resources required to build the system.
            2. Organizational risks arise from the organizational environment where the software is being developed.
            3. People risks are associated with the people in the development team.
            4. Requirements risks come from changes to the customer requirements and the process of managing the requirements change.
            5. Technology risks come from the software or hardware technologies that are used to develop the system.
            6. Tools risks come from the software tools and other support software used to develop the system.
    - **Risk Analysis**
        - During the risk analysis process, you have to consider each identified risk and make a judgment about the probability and seriousness of that risk.. You have to rely on your judgment and experience of previous projects and the problems that arose in them. It is not possible to make precise, numeric assessment of the probability and seriousness of each risk. Rather, you should assign the risk to one of a number of bands:
        - The probability of the risk might be assessed as insignificant, low, moderate, high, or very high.
        - The effects of the risk might be assessed as catastrophic (threaten the survival of the project), serious (would cause major delays), tolerable (delays are within allowed contingency), or insignificant.
        - Once the risks have been analyzed and ranked, you should assess which of these risks are most significant.
        - Your judgment must depend on a combination of the probability of the risk arising and the effects of that risk.
        - In general, catastrophic risks should always be considered, as should all serious risks that have more than a moderate probability of occurrence.
    - **Risk Planning**
        - The risk planning process develops strategies to manage the key risks that threaten the project.
        - For each risk, you have to think of actions that you might take to minimize the disruption to the project if the problem identified in the risk occurs.
        - You should also think about the information that you need to collect while monitoring the project so that emerging issues can be detected before they become serious.
        - In risk planning, you have to ask “what-if” questions that consider both individual risks, combinations of risks, and external factors that affect these risks.
        - Based on the answers to these “what-if” questions, you may devise strategies for managing the risks.
        - These strategies fall into three categories:

        - **Avoidance strategies:** Following these strategies means that the probability that the risk will arise is reduced. An example of a risk avoidance strategy is the strategy for dealing with defective components
        - **Minimization strategies:** Following these strategies means that the impact of the risk is reduced. An example of a risk minimization strategy is the strategy for staff illness
        - **Contingency plans:** Following these strategies means that you are prepared for the worst and have a strategy in place to deal with it. An example of a contingency strategy is the strategy for organizational financial issues.
    - **Risk Monitoring**
        - Risk monitoring is the process of checking that your assumptions about the product, process, and business risks have not changed.
        - You should regularly assess each of the identified risks to decide whether that risk is becoming more or less probable.
        - You should also think about whether the effects of the risk have changed.
        - To do this, you have to look at other factors, such as the number of requirements change requests, which give you clues about the risk probability and its effects.
        - You should monitor risks regularly at all stages in a project. At every management review, you should consider and discuss each of the key risks separately.
        - You should decide if the risk is more or less likely to arise, and if the seriousness and consequences of the risk have changed.

11. Explain elements of Software Quality Assurance and SQA Tasks. (Module 5) (7)(Dec 2022)

<u>Elements Of Software Quality Assurance</u>

- Software quality assurance encompasses a broad range of concerns and activities that focus on the management of software quality.
    - **Standards:** The IEEE, ISO, and other standards organizations have produced a broad array of software engineering standards and related documents. Standards may be adopted voluntarily by a software engineering organization or imposed by the customer or other stakeholders. The job of SQA is to ensure that standards that have been adopted are followed and that all work products conform to them.
    - **Reviews and audits:** Technical reviews are a quality control activity performed by software engineers for software engineers. Their intent is to uncover errors. Audits are a type of review performed by SQA personnel with the intent of ensuring that quality guidelines are being followed for software engineering work.
    - **Testing:** Software testing is a quality control function that has one primary goal—to fi nd errors. The job of SQA is to ensure that testing is properly planned and efficiently conducted so that it has the highest likelihood of achieving its primary goal.
    - **Error/defect collection and analysis:** The only way to improve is to measure how you're doing. SQA collects and analyzes error and defect data to better understand how errors are introduced and what software engineering activities are best suited to eliminating them.
    - **Change management:** Change is one of the most disruptive aspects of any software project. If it is not properly managed, change can lead to confusion, and confusion almost always leads to poor quality. SQA ensures that adequate change management practices have been instituted.
    - **Education:** Every software organization wants to improve its software engineering practices. A key contributor to improvement is education of software engineers, their managers, and other stakeholders. The SQA organization takes the lead in software process improvement and is a key proponent and sponsor of educational programs.
    - **Vendor management:** The job of the SQA organization is to ensure that high-quality software results by suggesting specific quality practices that the vendor should follow (when possible), and incorporating quality mandates as part of any contract with an external vendor.

    - **Security management:** With the increase in cyber crime and new government regulations regarding privacy, every software organization should institute policies that protect data at all levels, establish firewall protection for Web Apps, and ensure that software has not been tampered with internally. SQA ensures that appropriate process and technology are used to achieve software security.
    - **Safety:** Because software is almost always a pivotal component of human-rated systems (e.g., automotive or aircraft applications), the impact of hidden defects can be catastrophic. SQA may be responsible for assessing the impact of software failure and for initiating those steps required to reduce risk.
    - **Risk management:** Although the analysis and mitigation of risk is the concern of software engineers, the SQA organization ensures that risk management activities are properly conducted and that risk-related contingency plans have been established.

<u>SQA Tasks</u>

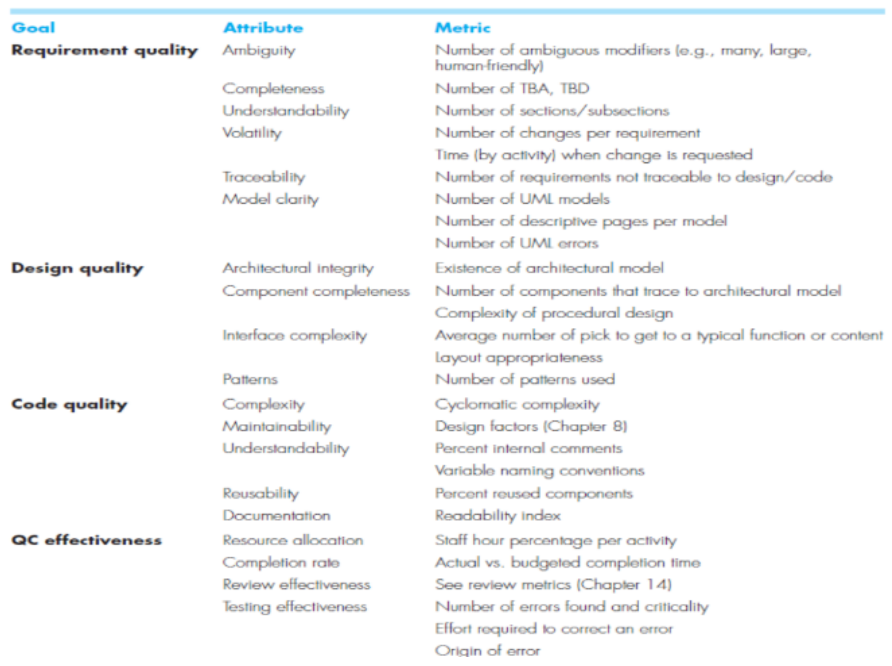
- The charter of the SQA group is to assist the software team in achieving a high-quality end product. The Software Engineering Institute recommends a set of SQA activities that address quality assurance planning, oversight, record keeping, analysis, and reporting. These activities are performed (or facilitated) by an independent SQA group that:
    - **Prepares an SQA plan for a project:** The plan is developed as part of project planning and is reviewed by all stakeholders. Quality assurance activities performed by the software engineering team and the SQA group are governed by the plan. The plan identifies evaluations to be performed, audits and reviews to be conducted, standards that are applicable to the project, procedures for error reporting and tracking, work products that are produced by the SQA group, and feedback that will be provided to the software team.
    - **Participates in the development of the project's software process description**: The software team selects a process for the work to be performed. The SQA group reviews the process description for compliance with organizational policy, internal software standards, externally imposed standards (e.g., ISO-9001), and other parts of the software project plan.
    - **Reviews software engineering activities to verify compliance with the defined software process**: The SQA group identifies, documents, and tracks deviations from the process and verifies that corrections have been made.
    - **Audits designated software work products to verify compliance with those defined as part of the software process:** The SQA group reviews selected work products; identifies, documents, and tracks deviations; verifies that corrections have been made; and periodically reports the results of its work to the project manager.
    - **Ensures that deviations in software work and work products are documented and handled according to a documented procedure:** Deviations may be encountered in the project plan, process description, applicable standards, or software engineering work products.
    - **Records any noncompliance and reports to senior management**: Noncompliance items are tracked until they are resolved.

12. List out the metrics that are used to measure software quality. Justify how metrics interpret the quality of the software? (Module 5) (5)(Dec 2021)

- The SQA activities described in the preceding section are performed to achieve a set of pragmatic goals:
    - **Requirement's quality:** The correctness, completeness, and consistency of the requirements model will have a strong influence on the quality of all work products that follow. SQA must ensure that the software team has properly reviewed the requirements model to achieve a high level of quality.
    - **Design quality**: Every element of the design model should be assessed by the software team to ensure that it exhibits high quality and that the design itself conforms to requirements. SQA looks for attributes of the design that are indicators of quality.
    - **Code quality:** Source code and related work products (e.g., other descriptive information) must conform to local coding standards and exhibit characteristics that will facilitate maintainability. SQA should isolate those attributes that allow a reasonable analysis of the quality of code.

- **Quality control effectiveness:** A software team should apply limited resources in a way that has the highest likelihood of achieving a high-quality result. SQA analyzes the allocation of resources for reviews and testing to assess whether they are being allocated in the most effective manner.

FIGURE 21.1 Software quality goals, attributes, and metrics
Source: Adapted from [Hya96].



Goal	Attribute	Metric
Requirement quality	Ambiguity	Number of ambiguous modifiers (e.g., many, large, human-friendly)
	Completeness	Number of TBA, TBD
	Understandability	Number of sections/subsections
	Volatility	Number of changes per requirement
		Time (by activity) when change is requested
	Traceability	Number of requirements not traceable to design/code
	Model clarity	Number of UML models
		Number of descriptive pages per model
		Number of UML errors
Design quality	Architectural integrity	Existence of architectural model
	Component completeness	Number of components that trace to architectural model
		Complexity of procedural design
	Interface complexity	Average number of pick to get to a typical function or content
		Layout appropriateness
	Patterns	Number of patterns used
Code quality	Complexity	Cyclomatic complexity
	Maintainability	Design factors (Chapter 8)
	Understandability	Percent internal comments
		Variable naming conventions
	Reusability	Percent reused components
	Documentation	Readability index
QC effectiveness	Resource allocation	Staff hour percentage per activity
	Completion rate	Actual vs. budgeted completion time
	Review effectiveness	See review metrics (Chapter 14)
	Testing effectiveness	Number of errors found and criticality
		Effort required to correct an error
		Origin of error

13. Describe in detail about the Software Process Improvement (SPI) process. **(Module 5)** **(10, 7)(Dec 2021, Dec 2022)**

- **Software Process Improvement (SPI)** encompasses a set of activities that will lead to a better software process and, as a consequence, higher quality software delivered in a more timely manner.
- SPI implies that:
  1. The elements of an effective software process can be defined effectively;
  2. An existing organizational approach to software development can be assessed against those elements; and
  3. A meaningful strategy for improvement can be defined.
- The SPI strategy transforms the existing approach to software development into something that is more focused, more repeatable, and more reliable.
- Because SPI is not free, it must deliver a return on investment.
- The effort and time that is required to implement an SPI strategy must pay for itself in some measurable way. To do this, the results of improved process and practice must lead to a reduction in software "problems" that cost time and money.
- **SPI Process**

  1. Assessment and Gap Analysis:

  The first road map activity, called assessment, allows you to get your bearings. The intent of assessment is to uncover both strengths and weaknesses in the way your organization applies the existing software process and the software engineering practices that populate the process. Assessment examines a wide range of actions and tasks that will lead to a high-quality process

  The difference between local application and best practice represents a "gap" that offers opportunities for improvement. The degree to which consistency, sophistication, acceptance, and commitment are achieved indicates the amount of cultural change that will be required to achieve meaningful improvement.

  2. Education and Training

  It follows that a key element of any SPI strategy is education and training for practitioners, technical managers, and more senior managers who have direct contact with the software organization. Three types of education and training should be conducted: generic software engineering concepts and methods, specific technology and tools, and communication and quality-oriented topics.

3. Selection and Justification

First, you should choose the process model that best fits your organization, its stakeholders, and the software that you build. You should decide which of the set of framework activities will be applied, the major work products that will be produced, and the quality assurance checkpoints that will enable your team to assess progress. If the SPI assessment activity indicates that you have specific weaknesses (e.g., you have no formal SQA functions), you should focus attention on process characteristics that will address these weaknesses directly.

Ideally, everyone works together to select various process and technology elements and moves smoothly toward the installation or migration activity. In reality, selection can be a rocky road. It is often difficult to achieve consensus among different constituencies. If the criteria for selection are established by committee, people may argue endlessly about whether the criteria are appropriate and whether a choice truly meets the criteria that have been established.

4. Installation and Migration

Installation is the first point at which a software organization feels the effects of changes implemented as a consequence of the SPI road map. In some cases, an entirely new process is recommended for an organization. Framework activities, software engineering actions, and individual work tasks must be defined and installed as part of a new software engineering culture. Such changes represent a substantial organizational and technological transition and must be managed very carefully.

In other cases, changes associated with SPI are relatively minor, representing small, but meaningful modifications to an existing process model. Such changes are typically referred to as process migration. Today, many software organizations have a “process” in place. The problem is that it doesn’t work effectively. Therefore, an incremental migration from one process (that doesn’t work as well as desired) to another process is a more effective strategy

5. Evaluation

The evaluation activity assesses the degree to which changes have been instantiated and adopted, the degree to which such changes result in better software quality or other tangible process benefits, and the overall status of the process and the organizational culture as SPI activities proceed.

Both qualitative factors and quantitative metrics are considered during the evaluation activity. From a qualitative point of view, past management and practitioner attitudes about the software process can be compared to attitudes polled after installation of process changes. Quantitative metrics are collected from projects that have used the transitional or “to be” process and compared with similar metrics that were collected for projects that were conducted under the “as is” process.

6. Risk Management

A software organization should manage risk at three key points in the SPI process: prior to the initiation of the SPI road map, during the execution of SPI activities (assessment, education, selection, installation), and during the evaluation activity that follows the instantiation of some process characteristic. In general, the following categories can be identified for SPI risk factors: budget and cost, content and deliverables, culture, maintenance of SPI deliverables, mission and goals, organizational management, organizational stability, process stakeholders, schedule for SPI development, SPI development environment, SPI development process, SPI project management, and SPI staff.