T.Y. B.Sc. (IT) : Sem. VI

Project management

Time : 2½ Hrs. Prelim Question Paper Solution Marks : 75

Q.1.Attempt any TWO question of the following : [10]

Q.1(a) List Boehm's top ten principles of conventional software management. [5]

Ans.: Boehm's top 10 principles of conventional software management are :

- Finding and fixing a software problem after delivery costs 100 times more than finding and fixing the problem in the early design phases.
- You can compress software development schedules 25% of nominal, but no more.
- For every $1 you spend on development, you will spend $2 on maintenance.
- Software development and maintenance costs are primarily a function of hi number of source lines of code
- Variations among people account for the biggest differences in software productivity.
- The overall ratio of software to hardware cost is still growing. In 1955, it was 15:85; In 1985, 85:15.
- Only about 15% of software development effort is devoted to programming.
- Software systems and products typically costs 3 times as much per SLOC as individual software programs. Software-system products cost 9 times as much.
- Walkthrough catch 60% of the errors.
- 80% of the contribution comes from 20% of the contributors.

Q.1(b) What are the important trends in improving software economics? [5]

Ans.:

Cost model parameters	Trends
Size Abstraction and component based Development technologies	 Higher order languages object-oriented Reuse commercial components
Process Methods and techniques	 Iterative development Process maturity models Architecture – first development Acquisition reform
Personnel People factors	Training and personnel skill development teamwork Win-win cultures
Environment Automation technologies and tools	 Integrated tools Open system Hardware platform performance Automation of coding, documents, testing, analyses
Quality Performance, reliability, accuracy	 Hardware platform performance Demonstration–based assessment Statistical quality control
Personnel People factors	 Training and personnel skill development teamwork Win-win cultures

Q.1(c) What are the factors that can be abstracted with software economics? Explain in detail. [5]

Ans.:
- Most software cost models can be abstracted into a function of five basic parameters: size, process, personnel, environment and required quality.
- The size of the end product (in human – generated components), which is typically quantified in terms of the number of source instructions or the number of function points required to develop the required functionality.
- The proves used to produce the end product, in particular the ability of the process to avoid non-value-adding activities(rework, bureaucratic delays, communications overhead)
- The capabilities of software engineering personnel and particularly their experience with the computer sciences issues and the applications domain issues of the project.
- The environment, which is made up of the tools and techniques available to support efficient software development and to automate the process.
- The required quality of the product, including its features, performance, reliability and adaptability.

Q.1(d) What are the three generations of Software development? Compare them. [5]

Ans.: The three generations of software development are as follows:

1. **Conventional:** Predictably bad: (60s/70s)
   - usually always over budget and schedule; missed requirements
   - All custom components; symbolic languages (assembler); some third generation languages (COBOL, Fortran, PL/1)
   - Performance, quality almost always less than great.
2. **Transition:** Unpredictable (80s/90s)
   - Infrequently on budget or on schedule
   - Enter software engineering; 'repeatable process;' **project management**
   - Some commercial products available – databases, networking, GUIs; But with huge growth in complexity, (especially to distributed systems) existing languages and technologies not enough for desired business performance
3. **Modern Practices:** Predictable (>2000s)
   - Usually on budget; on schedule. Managed, measured **process management**. Integrated environments;70% off-the-shelf components. Component-based applications RAD; iterative development; stakeholder emphasis.

Q2.Attempt any TWO question of the following : [10]

Q2(a) State and explain any 10 Principles of Conventional Software Engineering. [5]

Ans.:

1. **Make quality #1.** Quality must be quantified and mechanisms put into place to motivate its achievement.
   Defining quality commensurate with the project at hand is important but is not easily done at the outset of a project. Consequently, a modern process framework strives to understand the trade-offs among features, quality, cost, and schedule as early in the life cycle as possible. Until this understanding is achieved, it is not possible to specify or manage the achievement of quality.
2. **High-quality software is possible.** Techniques that have been demonstrated to increase quality include involving the customer, prototyping, simplifying design, conducting inspections, and hiring the best people.
   This principle is mostly redundant with the others.
3. **Give products to customers early.** No matter how hard you try to learn users' needs during the requirements phase, the most effective way to determine real needs is to give users a product and let them play with it.
   This is a key tenet of a modern process framework, and there must be several mechanisms to involve the customer throughout the life cycle. Depending on the domain, these mechanisms may include demonstrable prototypes, demonstration-based milestones, and alpha/beta releases.

4. **Determine the problem before writing the requirements.** When faced with what they believe is a problem, most engineers rush to offer a solution. Before you try to solve a problem, be sure to explore all the alternatives and don't be blinded by the obvious solution.
   This principle is a clear indication of the issues involved with the conventional requirements specification process. The parameters of the problem become more tangible as a solution evolves. A modern process framework evolves the problem and the solution together until the problem is well enough understood to commit to full production.
5. **Evaluate design alternatives.** After the requirements are agreed upon, you must examine a variety of architectures and algorithms. You certainly do not want to use an "architecture" simply because it was used in the requirements specification.
   This principle seems anchored in the waterfall mentality in two ways: (1) The requirements precede the architecture rather than evolving together. (2) The architecture is incorporated in the requirements specification. While a modern process clearly promotes the analysis of design alternatives, these activities are done concurrently with requirements specification, and the notations and artifacts for requirements and architecture are explicitly decoupled.
6. **Use an appropriate process model.** Each project must select a process that makes the most sense for that project on the basis of corporate culture, willingness to take risks, application area, volatility of requirements, and the extent to which requirements are well understood.
   It's true that no individual process is universal. I use the term process framework to represent a flexible class of processes rather than a single rigid instance. Chapter 14 discusses configuration and tailoring of the process to the various needs of a project.
7. **Use different languages for different phases**. Our industry's eternal thirst for simple solutions to complex problems has driven many to declare that the best development method is one that uses the same notation throughout the life cycle. Why should software engineers use Ada for requirements, design, and code unless Ada were optimal for all these phases?
   This is an important principle. Chapter 6 describes an appropriate organization and recommended languages/notations for the primitive artifacts of the process.
8. **Minimize intellectual distance.** To minimize intellectual distance, the software's structure should be as close as possible to the real-world structure.
   This principle has been the primary motivation for the development of object-oriented techniques, component-based development, and visual modeling.
9. **Put techniques before tools.** An undisciplined software engineer with a tool becomes a dangerous, undisciplined software engineer.
   Although this principle is valid, it misses two important points: (1) A disciplined software engineer with good tools will outproduce disciplined software experts with no tools. (2) One of the best ways to promote, standardize, and deliver good techniques is through automation.
10. **Get it right before you make it faster.** It is far easier to make a working program run faster than it is to make a fast program work. Don't worry about optimization during initial coding.

Q.2(b) Explain the top 10 principles of Modern software management. [5]

Ans.: The Principles of Modern Software Management

1. Base the process on an architecture-first approach: This requires that a demonstrable balance be achieved among the driving requirements, the architecturally significant design decisions, and the life-cycle plans before the resources are committed for full-scale development.
2. Establish an Iterative life-cycle process that confronts risk early: With today's sophisticated software systems, it is possible to define the entire problem, design the entire solution, build the software, then test the end product in sequence. Instead, an iterative process that refines the problem understanding, an effective solution, and an effective plan over several iterations encourages a balanced treatment of all stakeholders objectives. Major risks must be addressed early to increase predictability and avoid expensive downstream scrap and rework.
3. Transition design methods to emphasize component-based development: Moving from a line-of-code mentality to a component-based mentality is necessary to reduce the amount of human-generated source code and custom development. A component is a cohesive set of preexisting lines of code, either in source or executable format, with a defined interface and behavior.

4. Establish a change management environment: The dynamics of iterative development, including concurrent workflows by different teams working on shared artifacts, necessitates objectively controlled baselines.
5. Enhance change freedom through tools that support round-trip engineering: Roundtrip engineering is the environment support necessary to automate and synchronize engineering information in different formats (such as requirements specifications, design models, source code, executable code, test cases). Without substantial automation of this bookkeeping, change management, documentation, and testing, it is difficult to reduce iteration cycles to manageable time frames in which change is encouraged rather than avoided. Change freedom is a necessity in an iterative process, and establishing an integrated environment is crucial.
6. Capture design artifacts in rigorous, model-based notation: A model-based approach (such as UML) supports the evolution of semantically rich graphical and textual design notations. Visual modeling with rigorous notations and a formal machine processable language provide for far more objective measures than the traditional approach of human review and inspection of ad hoc design representations in paper documents.
7. Instrument the process for objective quality control and progress assessment: Lifecycle assessment of the progress and the quality of all intermediate products must be integrated into the process. The best assessment mechanisms are well-defined measures derived directly from the evolving engineering artifacts and integrated into all activities and teams.
8. Use a demonstration-based approach to access intermediate artifacts: Transitioning the current state-of-the-product artifacts into an executable demonstration of relevant scenarios stimulates earlier convergence or integration, a more tangible understanding of design trade-offs, and earlier elimination of architectural defects.
9. Plan intermediate releases in groups of usage scenarios with evolving levels of detail: It is essential that the software management process drive toward early and continuous demonstrations within the operational context of the system, namely its use cases. The evolution of projects increments and generations must be commensurate with the current level of understanding of the requirements and architecture.
10. Establish a configurable process that is economically scalable: No single process is suitable for all software developments. A pragmatic process framework must be configurable to a broad spectrum of applications. The process must ensure that there is economy of scale and return on investment by exploiting a common process spirit, extensive process automation, and common architecture patterns and components.

Q.2(c) Draw the diagram of the Phases of the Life-cycle of a software development process. [5]
Describe Inception Phase in detail.

Ans.:

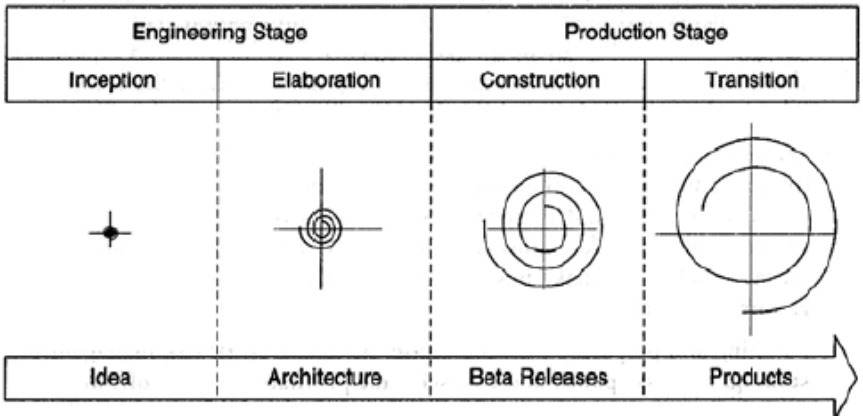


Fig. : The phases of the life-cycle process

Inception Phase

The overriding goal of the inception phase is to achieve concurrence among stakeholders on the life-cycle objectives for the project.

Primary Objectives

- Establishing the project's software scope and boundary conditions, including an operational

concept, acceptance criteria, and a clear understanding of what is and is not intended to be in the product.
- Discriminating the critical use cases of the system and the primary scenarios of operation that will drive the major design trade-offs.
- Demonstrating at least one candidate architecture against some of the primary scenarios.
- Estimating the cost and schedule for the entire project (including detailed estimates for the elaboration phase).
- Estimating potential risks (sources of unpredictability).

Essential Activities
- Formulating the scope of the project. This activity involves capturing the requirements and operational concept in an information repository that describes the user's view of the requirements. The information repository should be sufficient to define the problem space and derive the acceptance criteria for the end product.
- Synthesizing the architecture. Design trade-offs, problem space ambiguities, and available solution-space assets (technologies and existing components) are evaluated. An information repository is created that is sufficient to demonstrate the feasibility of at least one candidate architecture and an initial baseline of make/buy decisions so that the cost, schedule, and resource estimates can be derived.
- Planning and preparing a business case. Alternatives for risk management, staffing, iteration plans, and cost/schedule/profitability trade-offs are evaluated. The infrastructure (tools, processes, automation support) sufficient to support the lifecycle development task is determined.

Primary Evaluation Criteria
- Do all stakeholders concur on the scope definition and cost and schedule estimates?
- Are requirements understood, as evidenced by the fidelity of the critical use cases?
- Are the cost and schedule estimates, priorities, risks, and development processes credible?
- Do the depth and breadth of an architecture prototype demonstrate the preceding criteria? (The primary value of prototyping a candidate architecture is to provide a vehicle for understanding the scope and assessing the credibility of the development group in solving the particular technical problem.)
- Are actual resource expenditures versus planned expenditures acceptable?

Q2(d) Write a note on Vision document. [5]

Ans.: Most of the engineering artifacts are captured in rigorous engineering notations such as UML, programming languages, or executable machine codes. Because this book is written from a management perspective, it does not dwell on these artifacts. However, three engineering artifacts are explicitly intended for more general review, and they deserve further elaboration.

Vision Document

The vision document provides a complete vision for the software system under development and supports the contract between the funding authority and the development organization. Whether the project is a huge military-standard development (whose vision could be a 300-page system specification) or a small, internally funded commercial product (whose vision might be a two-page white paper), every project needs a source for capturing the expectations among stakeholders. A project vision is meant to be changeable as understanding evolves of the requirements, architecture, plans, and technology. A good vision document should change slowly. Figure 11 provides a default outline for a vision document.

I. **Feature set description**
   A. Precedence and priority
II. **Quality attributes and ranges**
III. **Required constraints**
   A. External interfaces
IV. **Evolutionary appendixes**
   A. Use cases
      1. Primary scenarios
      2. Acceptance criteria and tolerances
   B. Desired freedoms (potential change scenarios)

Fig. : Typical vision document outline.

The vision document is written from the user's perspective, focusing on the essential features of the system and acceptable levels of quality. The vision document should contain at least two appendixes. The first appendix should describe the operational concept using use cases (a visual model and a separate artifact). The second appendix should describe the change risks inherent in the vision statement, to guide defensive design efforts.

The vision statement should include a description of what will be included as well as those features considered but not included. It should also specify operational capacities (volumes, response times, accuracies), user profiles, and interoperation interfaces with entities outside the system boundary, where applicable. The vision should not be defined only for the initial operating level; its likely evolution path should be addressed so that there is a context for assessing design adaptability. The operational concept involves specifying the use cases and scenarios for nominal and off-nominal usage. The use case representation provides a dynamic context for understanding and refining the scope, for assessing the integrity of a design model, and for developing acceptance test procedures. The vision document provides the contractual basis for the requirements visible to the stakeholders.

Q.3Attempt any TWO question of the following : [10]

Q.3(a) Explain Iteration workflows. [5]

Ans.: Iteration Workflows

An iteration consists of a loosely sequential set of activities in various proportions depending on where the iteration is located in the development cycle. Each iteration is defined in terms of a set of allocated usage scenarios. The components needed to implement all selected scenarios are developed and integrated with the results of
previous iterations.

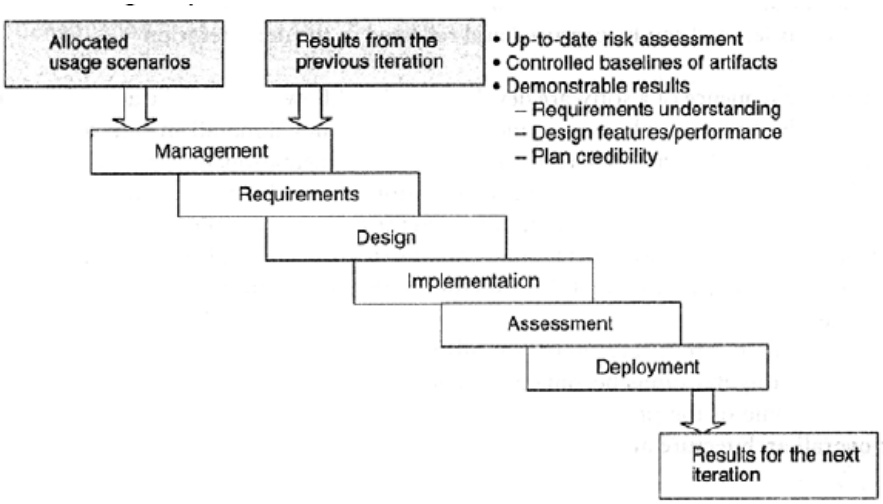


Fig. : The workflow of an iteration

An individual iteration's workflow, illustrated in Figure 2, generally includes the following sequence:
Management: iteration planning to determine the content of the release and develop the detailed plan for the iteration; assignment of work packages, or tasks, to the development team.

Environment: evolving the software change order database to reflect all new baselines and changes to existing baselines for all product, test, and environment components.
Requirements: analyzing the baseline plan, the baseline architecture, and the baseline requirements set artifacts to fully elaborate the use cases to be demonstrated at the end of this iteration and their evaluation criteria; updating any requirements set artifacts to reflect changes necessitated by results of this iteration's engineering activities.
Design: evolving the baseline architecture and the baseline design set artifacts to elaborate fully the design model and test model components necessary to demonstrate against the evaluation criteria allocated to this iteration; updating design set artifacts to reflect changes necessitated by the results of this iteration's engineering activities.
Implementation: developing or acquiring any new components, and enhancing or modifying any existing components, to demonstrate the evaluation criteria allocated to this iteration; integrating and testing all new and modified components with existing baselines.
Assessment: evaluating the results of the iteration, including compliance with the allocated evaluation criteria and the quality of the current baselines; identifying any rework required and determining whether it should be performed before deployment of this release or allocated to the next release; assessing results to improve the basis of the subsequent iteration's plan.
Deployment: transitioning the release either to an external organization (such as a user, independent verification and validation contractor, or regulatory agency) or to internal closure by conducting a post-mortem so that lessons learned can be captured and reflected in the next iteration.

Q.3(b) Write a note on Periodic Status Assessment. [5]

Ans.: Periodic Status Assessments

- Managing risks requires continuous attention to all the interacting activities of a software development effort.
- Periodic status assessments are management review conducted at regular intervals (monthly, quarterly) to address progress and quality indicators, ensure continuous attention to project dynamics, and maintain open communications among all stakeholders.
- The paramount objective of these assessments is to ensure that the expectations of all stakeholder (contractor, customer, user, subcontractor) are synchronized and consistent.
- Periodic status assessments serve as project snapshots. While the period may vary, the recurring event forces the project history to be captured and documented.

Status assessments provide the following:

- A mechanism for openly addressing, communicating, and resolving management issues, technical issues, and project risks
- Objective data derived directly from on-going activities and evolving product configurations.
- A mechanism for disseminating process, progress, quality trends, practices, and experience information to and from all stakeholders in an open forum
- Recurring themes from unsuccessful projects include status assessments that are 1)high-overhead activities, because the work associated with generating the status is separate from the everyday work, and 2)frequently canceled, because of higher priority issues that require resolution.
- Recurring themes from successful projects include status assessments that are (1) low overhead activities, because the material already exists as everyday management data, and (2) rarely canceled, because they are considered too important.

Q.3(c) Enlist the top seven workflows and explain any four of them in detail. [5]

Ans.: There are seven top-level workflows:

- Management workflow: controlling the process and ensuring win conditions for all stakeholders
- Environment workflow: automating the process and evolving the maintenance environment
- Requirements workflow: analyzing the problem space and evolving the requirements artifacts
- Design workflow: modeling the solution and evolving the architecture and design artifacts

- Implementation workflow: programming the components and evolving the implementation and deployment artifacts
- Assessment workflow: assessing the trends in process and product quality
- Deployment workflow: transitioning the end products to the user

Table 1 : The artifacts and life–cycle emphases associated with each workflow.

Workflow	Artifacts	Life–Cycle Phase Emphasis
Management	Business case Software development Plan Status assessments Vision Work breakdown structure	Inception: Prepare business case and vision Elaboration: Plan development Construction: Monitor and control development Transition: Monitor and control deployment
Environment	Environment Software change order database	Inception: Define development environment and change management infrastructure Elaboration: Install development environment and establish change management database Construction: Maintain development environment and software change order database Transition: Transition maintenance environment and software change order database
Requirements	Requirements set Release specifications Vision	Inception: Define operational concept Elaboration: Define architecture objectives Construction: define iteration objectives Transition: Refine release objectives
Design	Design set Architecture description	Inception: Formulate architecture concept Elaboration: Achieve architecture baseline Construction: Design components Transition: Refine architecture and components
Implementation	Implementation set Deployment set	Inception: Support architecture prototypes Elaboration: Produce architecture baseline Construction: Produce complete componentry Transition: Maintain components
Assessment	Release specifications Release descriptions User manual Deployment set	Inception: Assess plans, vision, prototypes Elaboration: Assess architecture Construction: Assess interim releases Transition: Assess product releases
Deployment	Deployment set	Inception: Analyze user community Elaboration: Define user manual Construction: Prepare transition materials Transition: Transition product to user

Q.3(d) "In order to ensure the consistency on various artifacts, the major milestones concentrate on objective, operational capabilities and release issues". Explain this statement. **[5]**

Ans.: Life-Cycle Objectives Milestone

- The life-cycle objectives milestone occurs at the end of the inception phase. The goal is to present to all stakeholders a recommendation on how to proceed with development, including a plan, estimated cost and schedule, and expected benefits and cost savings.
- The vision statement and the critical issues relative to requirements and the operational concept are addressed. A draft architecture document and a prototype architecture demonstration provide evidence of the completeness of the vision and the software development plan. A successfully completed life-cycle objectives milestone will result in authorization from all stakeholders to proceed with the elaboration phase.

Initial Operational Capability Milestone

- The initial operational capability milestone occurs late in the construction phase. The goals are to assess the readiness of the software to begin the transition into customer/user sites and to

authorize the start of acceptance testing. Issues are addressed concerning installation instructions, software version descriptions, user and operator manuals, and the ability of the development organization to support user sites.

- Software quality metrics are reviewed to determine whether quality is sufficient for transition. The readiness of the test environment and the test software for acceptance testing is assessed. Acceptance testing can be done incrementally across multiple iterations or can be completed entirely during the transition phase. The initiation of the transition phase is not necessarily the completion of the construction phase. These phases typically overlap until an initial product is delivered to the user for self-sufficient operation.

Product Release Milestone

- The product release milestone occurs at the end of the transition phase. The goal is to assess the completion of the software and its transition to the support organization, if any. The results of acceptance testing are reviewed, and all open issues are addressed.
- These issues could include installation instructions, software version descriptions, user and operator manuals, software support manuals, and the installation of the development environment at the support sites. Software quality metrics are reviewed to determine whether quality is sufficient for transition to the support organization.

Q.4Attempt any TWO question of the following : [10]

Q.4(a) Write a note on 'Line-of-Business' organizations. [5]

Ans.: The main features of the default organization are as follows:

- Responsibility for process definition and maintenance is specific to a cohesive line of business, where process commonality makes sense. For example, the process for developing avionics software is different from the process used to develop office applications.
- Responsibility for process automation is an organizational role and is equal in importance to the process definition role. Projects achieve process commonality primarily through the environment support of common tools.
- Organizational roles may be fulfilled by a single individual or several different teams, depending on the scale of the organization. A 20-person software product company may require only a single person to fulfill all the roles, while a 10,000-person telecommunications company may require hundreds of people to achieve an effective software organization.

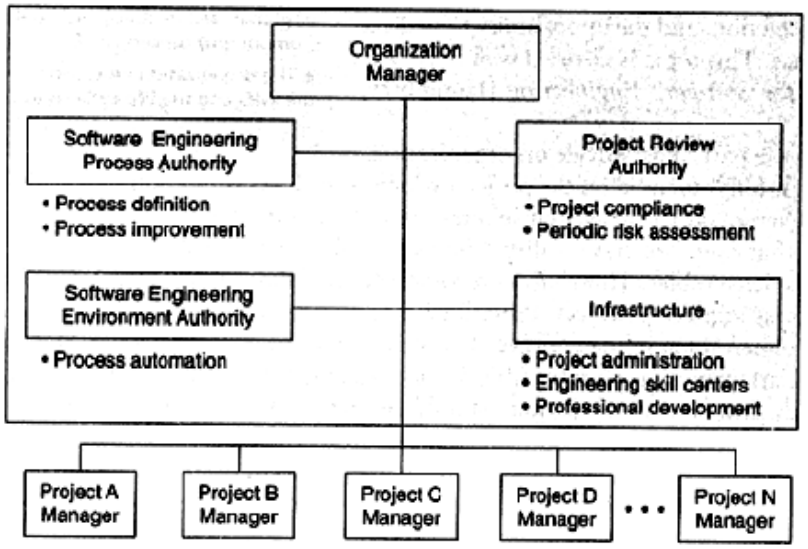


Fig. : Default roles in a software line-of-business organization

Software Engineering Process Authority

- The Software Engineering Process Authority (SEPA) facilitates the exchange of information and process guidance both to and from project practitioners.

- This role is accountable to the organization general manager for maintaining a current assessment of the organization's process maturity and its plan for future process improvements.
- The SEPA must help initiate and periodically assess project processes. Catalyzing the capture and dissemination of software best practices can be accomplished only when the SEPA understands both the desired improvement and the project context.
- The SEPA is a necessary role in any organization. It takes on responsibility and accountability for the process definition and its maintenance modification, improvement, technology insertion).
- The SEPA could be a single individual, the general manager, or even a team of representatives. The SEPA must truly be an authority, competent and powerful, not a staff position rendered impotent by ineffective bureaucracy.

Project Review Authority

- The Project Review Authority (PRA) is the single individual responsible for ensuring that a software project complies with all organizational and business unit software policies, practices, and standards.
- A software project manager is responsible for meeting the requirements of a contract or some other project compliance standard, and is also accountable to the PRA. The PRA reviews both the project's conformance to contractual obligations and the project's organizational policy obligations.
- The customer monitors contract requirements, contract milestones, contract deliverables, monthly management reviews, progress, quality, cost, schedule, and risk. The PRA reviews customer commitments as well as adherence to organizational policies, organizational deliverables, financial performance, and other risks and accomplishments.

Software Engineering Environment Authority

- The Software Engineering Environment Authority (SEEA) is responsible for automating the organization's process, maintaining the organization's standard environment, training projects to use the environment, and maintaining organization-wide reusable assets.
- The SEEA role is necessary to achieve a significant return on investment for a common process.
- Tools, techniques, and training can be amortized effectively across multiple projects only if someone in the organization (the SEEA) is responsible for supporting and administering a standard environment.
- In many cases, the environment may be augmented, customized, or modified, but the existence of an 80% default solution for each project is critical to achieving institutionalization of the organization's process and a good ROI on capital tool investments.

Infrastructure

- An organization's infrastructure provides human resources support, project-independent research and development, and other capital software engineering assets.
- The infrastructure for any given software line of business can range from trivial to highly entrenched bureaucracies.
- The typical components of the organizational infrastructure are as follows:
- Project administration: time accounting system; contracts, pricing, terms and conditions; corporate information systems integration.
- Engineering skill centers: custom tools repository and maintenance, bid and proposal support, independent research and development.
- Professional development: internal training boot camp, personnel recruiting, personnel skills database maintenance, literature and assets library, technical publications.

Q.4(b) Explain "Round Trip Engineering" with a neat diagram. [5]

Ans.: Round–Trip Engineering

- As the software industry moves into maintaining different information sets for the engineering artifacts, more automation support is needed to ensure efficient and error free transition of data from one artifact to another.
- Round–trip engineering is the environment support necessary to maintain consistency among the engineering artifacts.
- Figure 9 depicts some important transitions between information repositories. The automated translation of design models to source code (both forward and reverse engineering) is fairly well established.
- The automated translation of design models to process (distribution) models is also becoming straightforward through technologies such as ActiveX and the Common Object Request Broker Architecture (CORBA).
- Compilers and linkers have long provided automation of source code into executable code.
- As architectures start using heterogeneous components, platforms, and languages, the complexity of building, controlling, and maintaining large–scale webs of components introduces new needs for configuration control and automation of build management.
- However, today's environments do not support automation to the greatest extent possible. For example, automated test case construction from use case and scenario descriptions has not yet evolved to support anything except the most trivial examples, such as unit test scenarios.
- The primary reason for round–trip engineering is to allow freedom in changing software engineering data sources. This configuration control of all the technical artifacts is crucial to maintaining a consistent and error–free representation of the evolving product. It is not necessary, however, to have bi–directional transitions in all cases.
- For example, although we should be able to construct test cases for scenarios defined for a given logical set of objects, we cannot "reverse engineer" the objects solely from the test cases.
- Similarly, reverse engineering of poorly constructed legacy source code into an object–oriented design model may be counterproductive.
- Translation from one data source to another may not provide 100% completeness. For example, translating design models into C++ source code may provide only the structural and declarative aspects of the source code representation. The code components may still need to be fleshed out with the specifics of certain object attributes or methods.

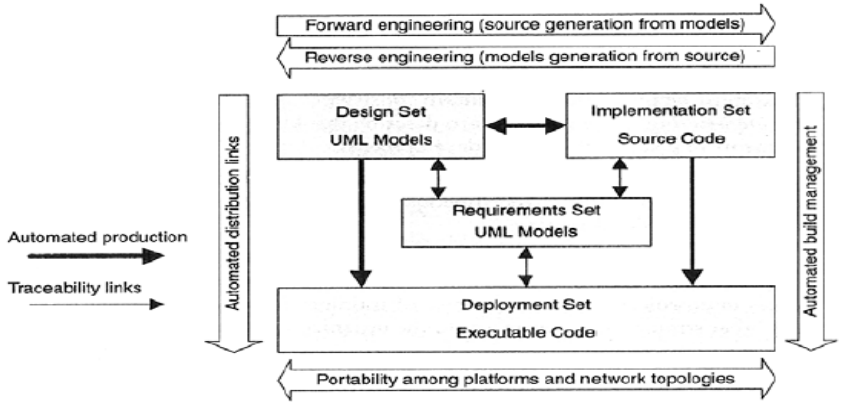


Fig. : Round-trip engineering

Q.4(c) Explain the basic fields of Software Change Order with the help of a template of the same. [5]

Ans.: The atomic unit of software work that is authorized to create, modify, or obsolesce components within a configuration baseline is called a software change order (SCO). Software change orders are a key mechanism for partitioning, allocating, and scheduling software work against an established software baseline and for assessing progress and quality. The basic fields of the SCO are title, description, metrics, resolution, assessment, and disposition.

(i) Title : This field represents the report name finalized upon acceptance by the configuration control board (CCB). It is suggested by the originator.

(ii) Description : This includes the details of the mentioned titles, error occurred, summarized report, data and name of the originator that helps to define the changes required.

(iii) Metrics : This field defines the category of the initiated process and the comparison between actual and processed estimates. This is important for planning, scheduling & for assessing quality improvements.

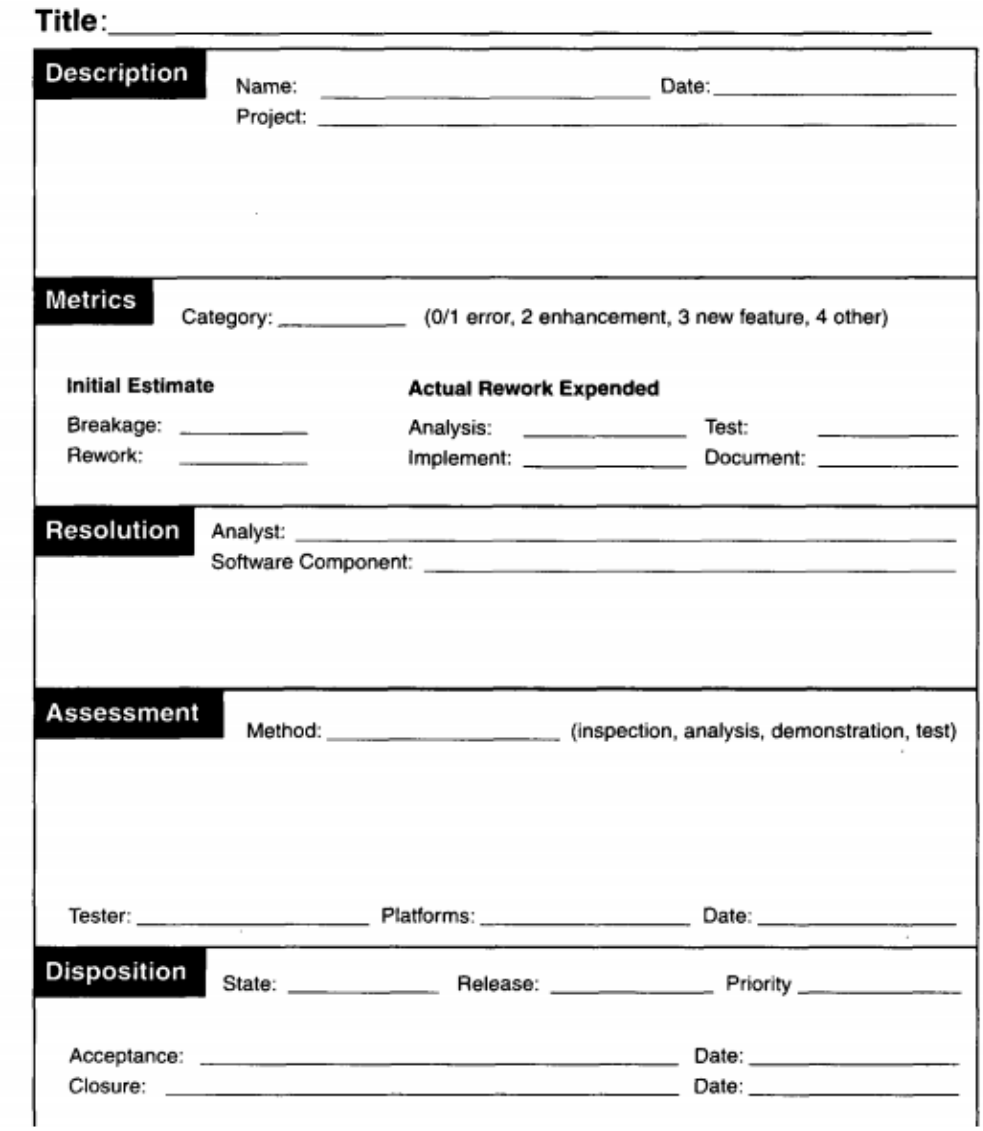
(iv) Resolution : It includes the name of the analyst who is responsible for the implementation of components changed and its description.

(v) Assessment : This describes the method of evaluation, its tester, platform and date of assessing the title.

(vi) Disposition : It specifies different set of states as proposed, accepted, rejected; archived, in process, in assessment and closed together with the specified date.

States are described as

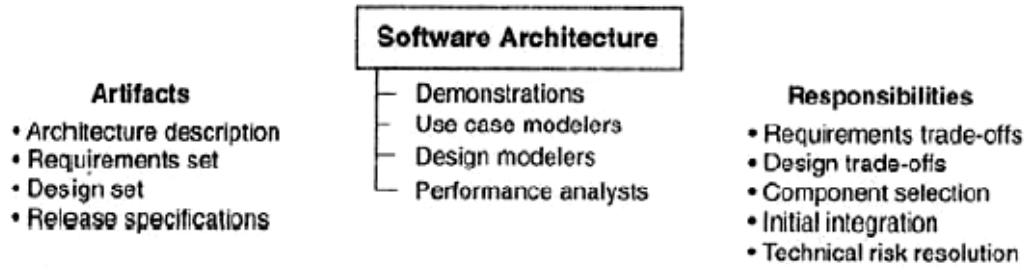
- Proposed: written, pending CCB review
- Accepted: CCB approved for resolution
- Rejected: closed with rationale as not a problem, duplicate, obsolete change resolved by another SCO
- Archived: accepted but postponed until a later release
- In progress: assigned and actively being resolved by the development organization.
- In assessment: resolved by development team, being assessed by a test team
- Closed: completely resolved, with the concurrence of all CCB members.



Q.4(d) Write a note on Software Architecture Team activities. [4]

Ans.:

- The software architecture team is responsible for the architecture.
- This responsibility encompasses the engineering necessary to specify a complete bill of materials for the software and the engineering necessary to make significant make/buy trade-offs so that all custom components are elaborated to the extent that construction/assembly costs are highly predictable.
- Figure 4 shows the focus of software architecture team activities over the project life cycle.



Life-Cycle Focus

Inception	Elaboration	Construction	Transition
Architecture prototyping Make/buy trade-offs Primary scenario definition Architecture evaluation criteria definition	Architecture baselining Primary scenario demonstration Make/buy trade-off baselining	Architecture maintenance Multiple-component issue resolution Performance tuning Quality improvements	Architecture maintenance Multiple-component issue resolution Performance tuning Quality improvements

Fig. : Software architecture team activities.

- For any project, the skill of the software architecture team is crucial. It provides the framework for facilitating team communications, for achieving system-wide qualities, and for implementing the applications. With a good architecture team, an average development team can succeed. If the architecture is weak, even an expert development team of superstar programmers will probably not succeed.
- In most projects, the inception and elaboration phases will be dominated by two distinct teams: the software management team and the software architecture team.
- The software development and software assessment teams tend to engage in support roles while preparing for the full-scale production stage. By the time the construction phase is initiated, the architecture transitions into a maintenance mode and must be supported by a minimal level of effort to ensure that there is continuity of the engineering legacy.
- To succeed, the architecture team must include a fairly broad level of expertise, including the following:
  - Domain experience to produce an acceptable design view (architecturally significant elements of the design model) and use case view (architecturally significant elements of the use case model)
  - Software technology experience to produce an acceptable process view (concurrency and control thread relationships among the design, component, and deployment models), component view (structure of the implementation set), and deployment view (structure of the deployment set)
  - The architecture team is responsible for system-level quality, which includes attributes such as reliability, performance, and maintainability.
  - These attributes span multiple components and represent how well the components integrate to provide an effective solution. In this regard, the architecture team decides how most multiple-component design issues are resolved.

Q.5Attempt any TWO question of the following : [10]

Q.5(a) Explain the seven core metrics in managing a modern process. [5]

Ans.: Overview of the seven core metrics

Metric	Purpose	Perspectives
Work and progress	Iteration planning, plan vs. actuals, management indicator	SLOC, function points, object points, scenarios, test cases, SCOs
Budgeted cost and expenditures	Financial insight, plan vs. actuals, management indicator	Cost per month, full-time staff per month, percentage of budget expended
Staffing and team dynamics	Resource plan vs. actuals, hiring rate, attrition rate	People per months added, people per month leaving
Change traffic and stability	Iteration panning, management indicator of schedule convergence	SCOs opened vs. SCOs closed, by type (0, 1, 2, 3, 4), by release/component/subsystem
Breakage and modularity	Convergence, software scrap, quality indicator	Reworked SLOC per change, by type (0, 1, 2, 3, 4), by release/component/subsystem
Rework and adaptability	Convergence, software rework, quality indicator	Average hours per change, by type (0, 1, 2, 3, 4), by release/component/subsystem
MTBF and maturity	Test coverage/adequacy, robustness for use, quality indicator	Failure counts, test hours until failure, by release/component/subsystem

Q.5(b) Explain the four quality indicators' metrics in detail. [5]

Ans.: Quality Indicators

The four quality indicators are based primarily on the measurement of software change across evolving baselines of engineering data (such as design models and source code). These metrics are developed more fully in Appendix C.

Change Traffic and Stability

- Overall change traffic is one specific indicator of progress and quality. Change traffic is defined as the number of software change orders opened and closed over the life cycle. This metric can be collected by change type, by release, across all releases, by team, by components, by subsystem, and so forth. Coupled with the work and progress metrics, it provides insight into the stability of the software and its convergence toward stability (or divergence toward instability).
- Stability is defined as the relationship between opened versus closed SCOs. The change traffic relative to the release schedule provides insight into schedule predictability, which is the primary value of this metric and an indicator of how well the process is performing. The next three quality metrics focus more on the quality of the product.

Breakage and Modularity

- Breakage is defined as the average extent of change, which is the amount of software baseline that needs rework (in SLOC, function points, components, subsystems, files, etc.). Modularity is the average breakage trend over time. For a healthy project, the trend expectation is decreasing or stable.
- This indicator provides insight into the benign or malignant character of software change. In a mature iterative development process, earlier changes are expected to result in more scrap than later changes. Breakage trends that are increasing with time clearly indicate that product maintainability is suspect.

Rework and Adaptability

- Rework is defined as the average cost of change, which is the effort to analyze, resolve, and retest all changes to software baselines. Adaptability is defined as the rework trend over time. For a healthy project, the trend expectation is decreasing or stable.

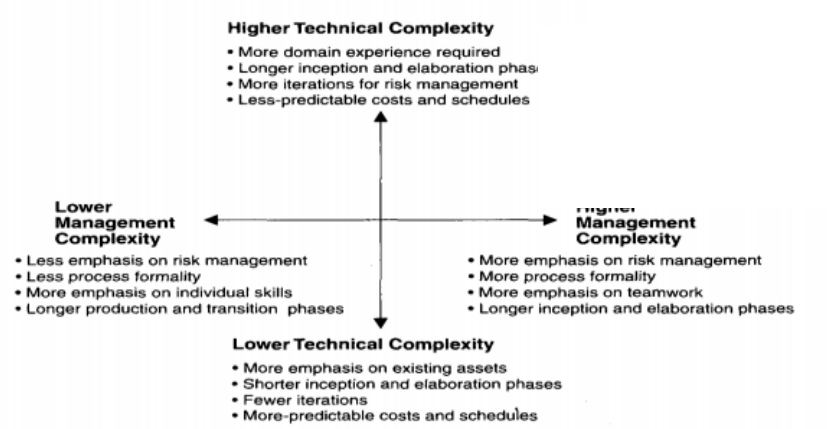
MTBF and Maturity

- MTBF is the average usage time between software faults. In rough terms, MTBF is computed by dividing the test hours by the number of type 0 and type 1 SCOs.
- Maturity is defined as the MTBF trend over time. Early insight into maturity requires that an effective test infrastructure be established. Conventional testing approaches for monolithic software programs focused on achieving complete test coverage of every line of code, every branch, and so forth. In today's distributed and componentized software systems, such complete test coverage is achievable only for discrete components. Systems of components are more efficiently tested by using statistical techniques.
- Consequently, the maturity metrics measure statistics over usage time rather than product coverage.

Q.5(c) Enlist the factors of tailoring a software process framework. Explain the scale factor in detail. [5]

Ans.: 1. Scale
2. Stakeholder cohesion and contention
3. Process flexibility or rigor
4. Process maturity
5. Architectural risk
6. Domain experience About Scale:

Perhaps the single most important factor in tailoring a software process framework to the specific needs of a project is the total scale of the software application. There are many ways to measure scale, including number of source lines of code, number of function points, number of use cases, and number of source lines of code, number of function points, number of use cases, and number of dollars. From a process tailoring perspective, the primary measure of scale is the size of the team. As the headcount increases, the importance of consistent interpersonal communications becomes paramount. Otherwise, the diseconomies of scale can have a serious impact on achievement of the project objectives.



Several size of the project:

(Explanation of different size of project---3 marks)

1. Trivial
2. Small
3. Medium-sized
4. Large
5. Huge

Trivial-sized projects require almost no management overhead (planning, communication, coordination, progress assessment, review, administration). There is little need to document the intermediate artifacts. Workflow is single-threaded. Performance is highly dependent on personnel skills.

Small projects (5 people) require very little management overhead, but team leadership toward a common objective is crucial. There is some need to communicate the intermediate artifacts among team members. Project milestones are easily planned, informally conducted, and easily changed. There is a small number of individual workflows. Performance depends primarily on personnel skills. Process maturity is relatively unimportant. Individual tools can have a considerable impact on performance.

Moderate-sized projects (25 people) require moderate management overhead, including a dedicated software project manager to synchronize team workflows and balance resources. Overhead workflows across all team leads are necessary for review, coordination, and assessment. There is a definite need to communicate the intermediate artifacts among teams. Project milestones are formally planned and conducted, and the impacts of changes are typically benign. There is a small number of concurrent team workflows, each with multiple individual workflows. Performance is highly dependent on the skills of key personnel, especially team leads. Process maturity is valuable. An environment can have a considerable impact on performance, but success can be achieved with certain key tools in place.

Large projects (125 people) require substantial management overhead, including a dedicated software project manager and several subproject managers to synchronize project-level and subproject-level workflows and to balance resources. There is significant expenditure in overhead workflows across all team leads for dissemination, review, coordination, and assessment. Intermediate artifacts are explicitly emphasized to communicate engineering results across many diverse teams. Project milestones are formally planned and conducted, and changes to milestone plans are expensive. Large numbers of concurrent team workflows are necessary, each with multiple individual workflows. Performance is highly dependent on the skills of key personnel, especially subproject managers and team leads. Project performance is dependent on average people, for two reasons:

1) There are numerous mundane jobs in any large project, especially in the overhead workflows.
2) The probability of recruiting, maintaining, and retaining a large number of exceptional people is small.

Huge projects (625 people) require substantial management overhead, including multiple software project managers and many subproject managers to synchronize project-level and subproject-level workflows and to balance resources. There is significant expenditure in overhead workflows across all team leads for dissemination, review, coordination, and assessment. Intermediate artifacts are explicitly emphasized to communicate engineering results across many diverse teams. Project milestones are very formally planned and conducted, and changes to milestone plans typically cause malignant replanning.

Q.5(d) Write a short note on pragmatic software metrics. [5]

Ans.:

- Measuring is useful, but it doesn't do any thinking for the decision makers. It only provides data to help them ask the right questions, understand the context, and make objective decisions. Because of the highly dynamic nature of software projects, these measures must be available at any time, tailorable to various subsets of the evolving product (release, version, component, class), and maintained so that trends can be assessed (first and second derivatives with respect to time).
- This situation has been achieved in practice only in projects where the metrics were maintained on-line as an automated by-product of the development/integration environment.

The basic characteristics of a good metric are as follows:

1) **It is considered meaningful by the customer, manager, and performer**. If any one of these stakeholders does not see the metric as meaningful, it will not be used. "The customer is always right" is a sales motto, not an engineering tenet. Customers come to software engineering

providers because the providers are more expert than they are at developing and managing software. Customers will accept metrics that are demonstrated to be meaningful to the developer.

2) **It demonstrates quantifiable correlation between process perturbations and business performance.** The only real organizational goals and objectives are financial: cost reduction, revenue increase, and margin increase.
3) **It is objective and unambiguously defined.** Objectivity should translate into some form of numeric representation (such as numbers, percentages, ratios) as opposed to textual representations (such as excellent, good, fair, poor). Ambiguity is minimized through well-understood units of measurement (such as staff-month, SLOC, change function point, class, scenario, requirement), which are surprisingly hard to define precisely in the software engineering world.
4) **It displays trends.** This is an important characteristic. Understanding the change in a metric's value with respect to time, subsequent projects, subsequent releases, and so forth is an extremely important perspective, especially for today's iterative development models. It is very rare that a given metric drives the appropriate action directly. More typically, a metric presents a perspective. It is up to the decision authority (manager, team, or other information processing entity) to interpret the metric and decide what action is necessary.
5) **It is a natural by-product of the process.** The metric does not introduce new artifacts or overhead activities; it is derived directly from the mainstream engineering and management workflows.
6) **It is supported by automation.** Experience has demonstrated that the most successful metrics are those that are collected and reported by automated tools, in part because software tools require rigorous definitions of the data they process. When metrics expose a problem, it is important to get underneath all the symptoms and diagnose it. Metrics usually display effects; the causes require synthesis of multiple perspectives and reasoning. For example, reasoning is still required to interpret the following situations correctly: A low number of change requests to a software baseline may mean that the software is mature and error-free, or it may mean that the test team is on vacation.
    - A software change order that has been open for a long time may mean that the problem was simple to diagnose and the solution required substantial rework, or it may mean that a problem was very time-consuming to diagnose and the solution required a simple change to a single line of code.
    - A large increase in personnel in a given month may cause progress to increase proportionally if they are trained people who are productive from the outset. It may cause progress to decelerate if they are untrained new hires who demand extensive support from productive people to get up to speed. Value judgments cannot be made by metrics; they must be left to smarter entities such as software project managers.

Q.6Attempt any TWO question of the following : [10]

Q.6(a) List the top 10 Software Management Principles. [5]

Ans.: 1. Base the process on an architecture-first approach.

An early focus on the architecture results in a solid foundation for the 20% of the stuff (requirements, components, use cases, risks, errors) that drives the overall success of the project. Getting the architecturally important components to be well understood and stable before worrying about the complete breadth and depth of the artifacts should result in scrap and rework rates that decrease or remain stable over the project life cycle.

2. Establish an iterative life-cycle process that confronts risk early.

A more dynamic planning framework supported by an iterative process results in better risk management and more predictable performance. Resolving the critical issues first results in a predictable construction phase with no surprises, as well as minimal exposure to sources of cost and schedule unpredictability.

3 **Transition design methods to emphasize component-based development.**
The complexity of a software effort is mostly a function of the number of human generated artifacts. Making the solution smaller reduces management complexity.

4. **Establish a change management environment**
The dynamics of iterative development, including concurrent workflows by different teams working on shared artifacts, necessitate highly controlled baselines.

5. **Enhance change freedom through tools that support round-trip engineering.**
Automation enables teams to spend more time on engineering and less time on overhead tasks.

6. **Capture design artifacts in rigorous, model-based notation.** An engineering notation for design enables complexity control, objective assessment, and automated analyses.

7. **Instrument the process for objective quality control and progress assessment.**
Progress and quality indicators are derived directly from the evolving artifacts, providing more-meaningful insight into trends and correlation with requirements.

8. **Use a demonstration-based approach to assess intermediate artifacts.** Integration occurs early and continuously throughout the life cycle. Intermediate results are objective and tangible.

9. **Plan intermediate releases in groups of usage scenarios with evolving levels of detail.** Requirements, designs, and plans evolve in balance. Useful software releases are available early in the life cycle.

10. **Establish a configurable process that is economically scalable.** Methods, techniques, tools, and experience can be applied straightforwardly to a broad domain, providing improved return on investment across a line of business.

Q.6(b) Explain the term "Next generation Software economics". [5]

Ans.: Next-Generation Software Economics

Next-generation software economics is being practiced by some advanced software organizations.

A general structure is proposed for a cost estimation model that would be better suited to the process framework. This new approach would improve the accuracy and precision of software cost estimates, and would accommodate dramatic improvements in software economies of scale. Such improvements will be enabled by advances in software development environments.

Next-Generation Cost Models

- Software experts hold widely varying opinions about software economics and its manifestation in software cost estimation models: Source lines of code versus function points. Economy of scale versus diseconomy of scale. Productivity measures versus quality measures. Java versus C++. Object-oriented versus functionally oriented. Commercial components versus custom development.
- All these topics represent industry debates surrounded by high levels of rhetoric.
- Energetic disagreement is an indicator of an industry in flux, in which many competing technologies and techniques are maturing rapidly. One of the results, however, is a continuing inability to predict with precision the resources required for a given software endeavor.
- Accurate estimates are possible today, although honest estimates are imprecise. It will be difficult to improve empirical estimation models while the project data going into these models are noisy and highly uncorrelated, and are based on differing process and technology foundations.
- Some of today's popular software cost models are not well matched to an iterative software process focused on an architecture-first approach. Despite many advances by some vendors of software cost estimation tools in expanding their repertoire of up-to-date project experience data, many cost estimators are still using a conventional process experience base to estimate a modern project profile. There are cost models and techniques in the industry that can support subsets of this approach. The software cost model is all theory; with no empirical evidence to demonstrate that this approach will be more accurate than today's cost models. Even though most of the methods and technology necessary for a modern management process are available

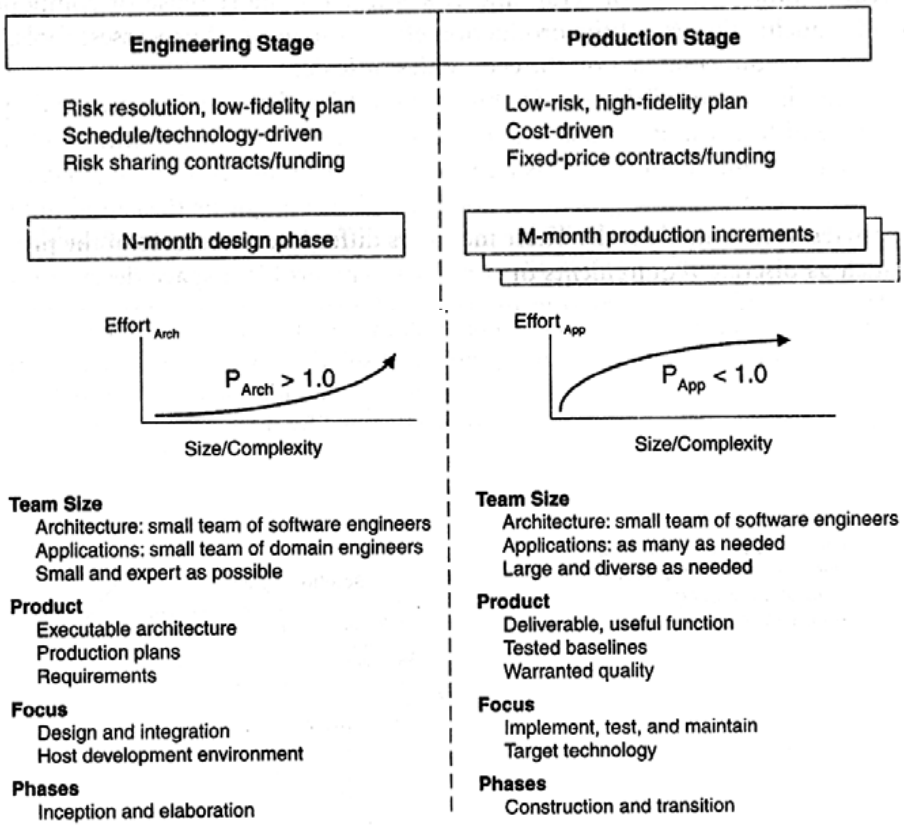
today, there are not enough relevant, completed projects to back up my assertions with objective evidence

- A next-generation software cost model should explicitly separate architectural engineering from application production, just as an architecture-first process does. The cost of designing, producing, testing, and maintaining the architecture baseline is a function of scale, quality, technology, process, and team skill. There should still be some diseconomy of scale (exponent greater than 1.0) in the architecture cost model because it is inherently driven by research and development-oriented concerns. When an organization achieves a stable architecture, the production costs should be an exponential function of size, quality, and complexity, with a much more stable range of process and personnel influence. The production stage cost model should reflect an economy of scale (exponent less than 1.0) similar to that of conventional economic models for bulk production of commodities.
- Next-generation software cost models should estimate large-scale architectures with economy of scale. This implies that the process exponent during the production stage will be less than 1.0. The larger the system, the more opportunity there is to exploit automation and to reuse common processes, components, and architectures.
- In the conventional process, the minimal level of automation that supported the overhead activities of planning, project control, and change management led to labor-intensive workflows and a diseconomy of scale. This lack of management automation was as true for multiple-project, line-of-business organizations as it was for individual projects. Next generation environments and infrastructures are moving to automate and standardize many of these management activities, thereby requiring a lower percentage of effort for overhead activities as scale increases.

$$\text{Effort} = F(T_{Arch}, S_{Arch}, Q_{Arch}, P_{Arch}) + F(T_{App}, S_{App}, Q_{App}, P_{App})$$

$$\text{Time} = F(P_{Arch}, \text{Effort}_{Arch}) + F(P_{App}, \text{Effort}_{App})$$

where:
T = technology parameter (environment automation support)
S = scale parameter (such as use cases, function points, source lines of code)
Q = quality parameter (such as portability, reliability, performance)
P = process parameter (such as maturity, domain experience)



Next Generation Cost Models

Q.6(c) What are the strategies to make error-free software? **[5]**

Ans.: The steps or strategies to make error-free software are as follows:

1. Continuous integration
2. Early risk resolution
3. Evolutionary requirements
4. Teamwork among stakeholders

Continuous Integration

- Iterative development produces the architecture first, allowing integration to occur as the verification activity of the design phase and enabling design flaws to be detected and resolved earlier in the life cycle.
- This approach avoids the big-bang integration at the end of a project by stressing continuous integration throughout' the project.
- Figure 1 illustrates the differences between the progress profile of a healthy modern project and that of a typical conventional project.
- The architecture-first approach forces integration into the design phase through the construction of demonstrations.
- The demonstrations do not eliminate the design breakage; instead, they make it happen in the engineering stage, when it can be resolved efficiently in the context of life-cycle goals.
- The downstream integration nightmare, late patches, and shoe-horned software fixes are avoided. The result is a more robust and maintainable design

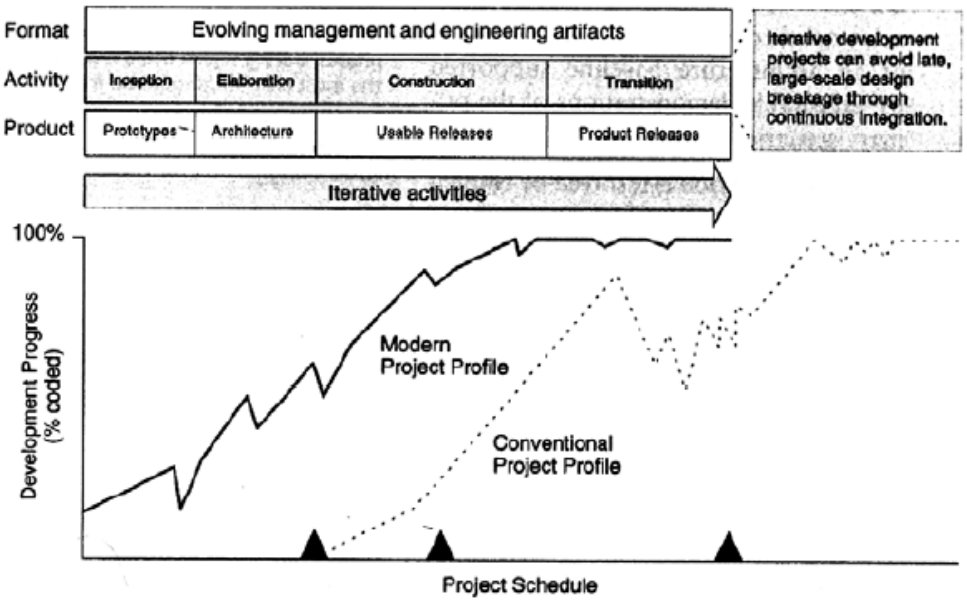


Fig.: Progress profile of a modern project

Table 1 : Differences in workflow cost allocations between a conventional process and a modern process.

Software Engineering Workflows	Conventional Process Expenditures	Modern Process Expenditures.
Management	5%	10%
Environment	5%	10%
Requirements	5%	10%
Design	10%	15%
Implementation	30%	25%
Assessment	40%	25%
Deployment	5%	5%
Total	100%	100%

- The continuous integration inherent in an iterative development process also enables better insight into quality trade-offs.
- System characteristics that are largely inherent in the architecture (performance, fault tolerance, maintainability) are tangible earlier in the process, when issues are still correctable without jeopardizing target costs and schedules.
- A recurring theme of successful iterative development projects is a cost profile very different from that experienced by conventional processes.
- Table 1 identifies the differences in a modern process profile from the perspective of the cost distribution among the various project workflows.
- This table is a simple combination of a typical conventional allocation and a default modern allocation.
- The primary discriminator of a successful modern process is inherent in the overall life-cycle expenditures for assessment and testing. Conventional projects, mired in inefficient integration and late discovery of substantial design issues, expend roughly 40% or more of their total resources in integration and test activities. Modern projects with a mature, iterative process deliver a product with only about 25% of the total budget consumed by these activities.

Early Risk Resolution

- The engineering stage of the life cycle (inception and elaboration phases) focuses on confronting the risks and resolving them before the big resource commitments of the production stage. Conventional projects usually do the easy stuff first, thereby demonstrating early progress.
- A modern process attacks the important 20% of the requirements, use cases, components, and risks. This is the essence of my most important principle: architecture first.
- Defining the architecture rarely includes simple steps for which visible progress can be achieved easily.
- The effect of the overall life-cycle philosophy on the 80/20 lessons learned over the past 30 years of software management experience provides a useful risk management perspective.
- 80% of the engineering is consumed by 20% of the requirements. Strive to understand the driving requirements completely before committing resources to fullscale development. Do not strive prematurely for high fidelity and full traceability of the requirements.
- 80% of the software cost is consumed by 20% of the components. Elaborate the cost-critical components first so that planning and control of cost drivers are well understood early in the life cycle.
- 80% of the errors are caused by 20% of the components. Elaborate the reliability critical components first so that assessment activities have enough time to achieve the necessary level of maturity.
- 80% of software scrap and rework is caused by 20% of the changes. Elaborate the change-critical components first so that broad-impact changes occur when the project is nimble.
- 80% of the resource consumption (execution time, disk space, memory) is consumed by 20% of the components. Elaborate the performance-critical components first so that engineering trade-offs with reliability, changeability, and cost-effectiveness can be resolved as early in the life cycle as possible.
- 80% of the progress is made by 20% of the people. Make sure that the initial team for planning the project and designing the architecture is of the highest quality. An adequate plan and adequate architecture can then succeed with an average construction team. An inadequate plan or inadequate architecture will probably not succeed, even with an expert construction team. Figure 2 compares the risk management profile of a modern project with profile for a typical conventional project.

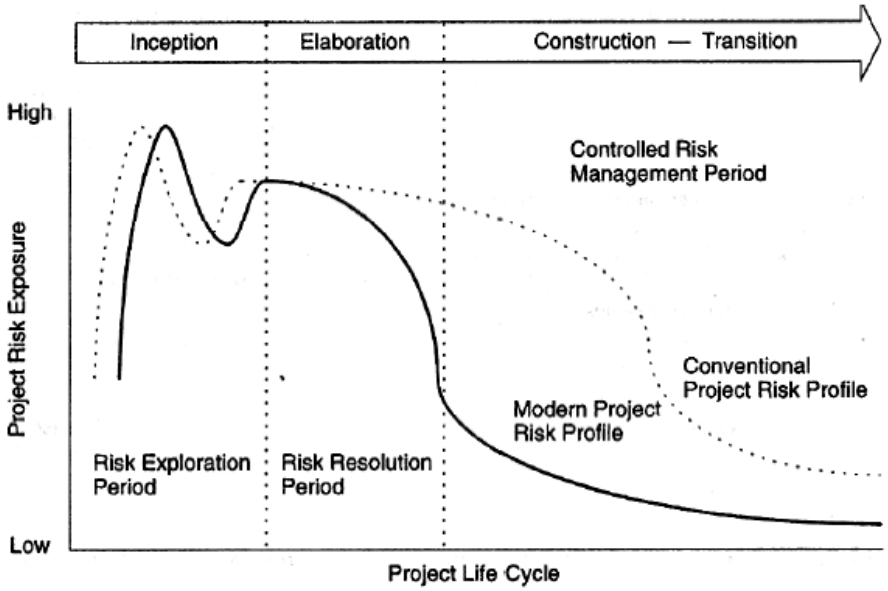


Fig.: Risk profile of a typical modern poject across its life cycle.

Evolutionary Requirements

- Conventional approaches decomposed system requirements into subsystem requirements, subsystem requirements into component requirements, and component requirements into unit requirements.
- The organization of requirements was structured so that traceability was simple. With an early life-cycle emphasis on requirements first, design second, then complete traceability between requirements and design components, the natural tendency was for the design structure to evolve into an organization that closely paralleled the structure of the requirements organization.
- It was no surprise that functional decomposition of the problem space led to a functional decomposition of the solution space.
- Most modern architectures that use commercial components, legacy components, distributed resources, and object-oriented methods are not trivially traced to the requirements they satisfy.
- There are now complex relationships between requirements statements and design elements, including 1 to 1, many to 1, 1 to many, conditional, time-based, and state-based.
- Top-level system requirements are retained as the vision, but lower level requirements are captured in evaluation criteria attached to each intermediate release. These artifacts, illustrated in Figure 3, are intended to evolve along with the process, with more and more fidelity as the life cycle progresses and requirements understanding matures. This is a fundamental difference from conventional requirements management approaches, in which this fidelity was pursued far too early in the life cycle.

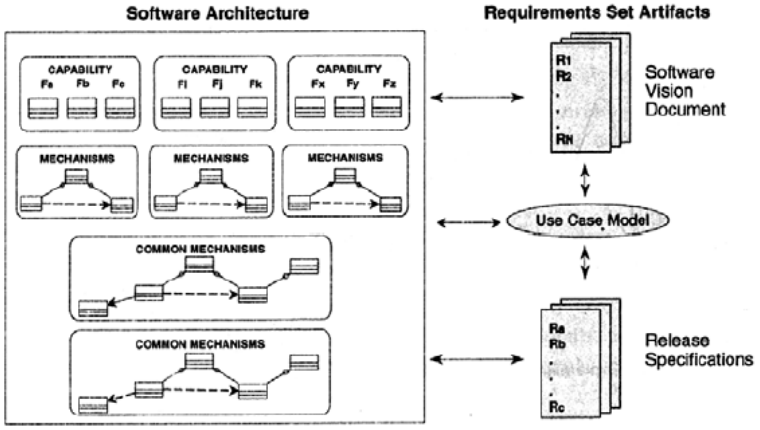


Fig.: Organization of software components resulting from a modern process.

- Many aspects of the classic development process cause stakeholder relationships to degenerate into mutual distrust, making it difficult to balance requirements, product features, and plans. A more iterative process, with more-effective working relationships between stakeholders, allows trade-offs to be based on a more objective understanding by everyone.
- This process requires that customers, users, and monitors have both applications and software expertise, remain focused on the delivery of a usable system (rather than on blindly enforcing standards and contract terms), and be willing to allow the contractor to make a profit with good performance. It also requires a development organization that is focused on achieving customer satisfaction and high product quality in a profitable manner.
- The transition from the exchange of mostly paper artifacts to demonstration of intermediate results is one of the crucial mechanisms for promoting teamwork among stakeholders. Major milestones provide tangible results and feedback from a usage point

Teamwork Among Stakeholders

- Many aspects of the classic development process cause stakeholder relationships to degenerate into mutual distrust, making it difficult to balance requirements, product features, and plans.
- A more iterative process, with more-effective working relationships between stakeholders, allows trade-offs to be based on a more objective understanding by everyone.
- This process requires that customers, users, and monitors have both applications and software expertise, remain focused on the delivery of a usable system (rather than on blindly enforcing standards and contract terms), and be willing to allow the contractor to make a profit with good performance.
- It also requires a development organization that is focused on achieving customer satisfaction and high product quality in a profitable manner.
- The transition from the exchange of mostly paper artifacts to demonstration of intermediate results is one of the crucial mechanisms for promoting teamwork among stakeholders. Major milestones provide tangible results and feedback from a usage point of view.

Q.6(d) Write a note on a Modern Project Profiles. [5]

Ans.: Modern Project Profiles

A modern process framework exploits several critical approaches for resolving these issues:

1) **Protracted integration and late design breakage** are resolved by forcing integration into the engineering stage. This is achieved through continuous integration of an architecture baseline supported by executable demonstrations of the primary scenarios.
2) **Late risk resolution** is resolved by emphasizing an architecture-first approach, in which the high-leverage elements of the system are elaborated early in the life cycle.
3) **The analysis paralysis of a requirements-driven functional decomposition** is avoided by organizing lower level specifications along the content of releases rather than along the product decomposition (by subsystem, by component, etc.).
4) **Adversarial stakeholder relationships** are avoided by providing much more tangible and objective results throughout the life cycle.
5) **The conventional focus on documents and review meetings** is replaced by a focus on demonstrable results and well-defined sets of artifacts, with more-rigorous notations and extensive automation supporting a paperless environment.

Q.7Attempt any THREE question of the following : [15]

Q.7(a) What are the attributes of a good estimate? [5]

Ans.:

- It is conceived and supported by the project manager, architecture team, development team, and test team accountable for performing the work.
- It is accepted by all stakeholders as ambitious but realizable.
- It is based on a well-defined software cost model with a credible basis.
- It is based on a database of relevant project experience that includes similar processes, similar technologies, similar environments, similar quality requirements and similar people.
- It is defined in enough detail so that its key risk areas are understood and the probability of success is objectively assessed.

Q.7(b) Explain the Construction Phase of a life cycle. [5]

Ans.: Construction Phase

- During the construction phase, all remaining components and application features are integrated into the application, and all features are thoroughly tested.
- Newly developed software is integrated where required. The construction phase represents a production process, in which emphasis is placed on managing resources and controlling operations to optimize costs, schedules, and quality.
- In this sense, the management mindset undergoes a transition from the development of intellectual property during inception and elaboration activities to the development of deployable products during construction and transition activities.
- Many projects are large enough that parallel construction increments can be spawned.
- These parallel activities can significantly accelerate the availability of deployable releases; they can also increase the complexity of resource management and synchronization of workflows and teams. A robust architecture is highly correlated with an understandable plan. In other words, one of the critical qualities of any architecture is its ease of construction. This is one reason that the balanced development of the architecture and the plan is stressed during the elaboration phase.

Primary Objectives

- Minimizing development costs by optimizing resources and avoiding unnecessary scrap and rework Achieving adequate quality as rapidly as practical
- Achieving useful versions (alpha, beta, and other test releases) as rapidly as practical.

Essential Activities

- Resource management, control, and process optimization
- Complete component development and testing against evaluation criteria
- Assessment of product releases against acceptance criteria of the vision

Primary Evaluation Criteria

- Is this product baseline mature enough to be deployed in the user community? (Existing defects are not obstacles to achieving the purpose of the next release.)
- Is this product baseline stable enough to be deployed in the user community? (Pending changes are not obstacles to achieving the purpose of the next release.)
- Are the stakeholders ready for transition to the user community?
- Are actual resource expenditures versus planned expenditures acceptable?

Q.7(c) How do you classify the joint management reviews? Explain all its types. [5]

Ans.: Three types of joint management reviews are conducted throughout the process:

1) **Major milestones:** These systemwide events are held at the end of each development phase. They provide visibility to systemwide issues, synchronize the management and engineering perspectives, and verify that the aims of the phase have been achieved.
2) **Minor milestones:** These iteration-focused events are conducted to review the content of an iteration in detail and to authorize continued work.
3) **Status assessments:** These periodic events provide management with frequent and regular insight into the progress being made. Each of the four phases—inception, elaboration, construction, and transition— consists of one or more iterations and concludes with a major milestone when a planned technical capability is produced in demonstrable form. An iteration represents a cycle of activities for which there is a well-defined intermediate result—a minor milestone—captured with two artifacts: a release specification (the evaluation criteria and plan) and a release description (the results). Major milestones at the end of each phase use formal, stakeholder-approved evaluation criteria and release descriptions; minor milestones use informal, development-team-controlled versions of these artifacts.

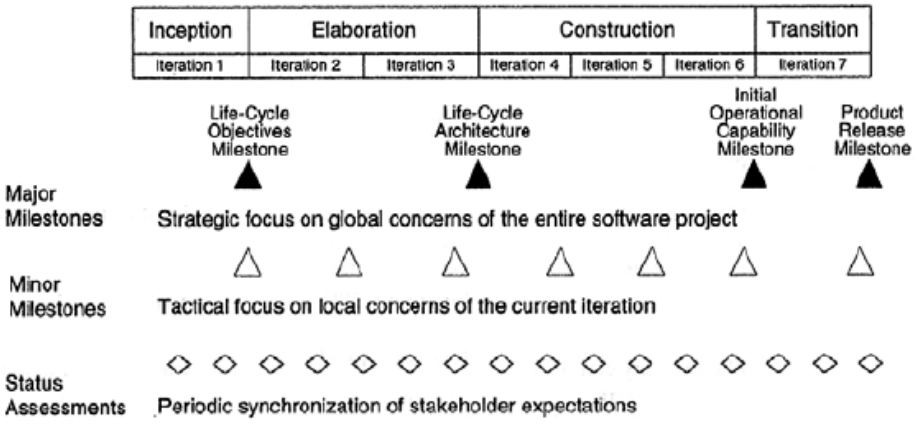


Fig. : A typical sequence of life-cycle checkpoints

Major Milestones

The four major milestones occur at the transition points between life-cycle phases. They can be used in many different process models, including the conventional waterfall model. Inan iterative model, the major milestones are used to achieve concurrence among all stakeholders on the current state of the project. Different stakeholders have very different concerns:

- Customers: schedule and budget estimates, feasibility, risk assessment, requirements understanding, progress, product line compatibility
- Users: consistency with requirements and usage scenarios, potential for accommodating growth, quality attributes
- Architects and systems engineers: product line compatibility, requirements changes, trade-off analyses, completeness and consistency, balance among risk, quality, and usability
- Developers: sufficiency of requirements detail and usage scenario descriptions, frameworks for component selection or development, resolution of development risk, product line compatibility, sufficiency of the development environment
- Maintainers: sufficiency of product and documentation artifacts, understandability, interoperability with existing systems, sufficiency of maintenance environment
- Others: possibly many other perspectives by stakeholders such as regulatory agencies, independent verification and validation contractors, venture capital investors, subcontractors, associate contractors, and sales and marketing teams

Minor Milestones

For most iterations, which have a one-month to six-month duration, only two minor milestones are needed: the iteration readiness review and the iteration assessment review.

- Iteration Readiness Review. This informal milestone is conducted at the start of each iteration to review the detailed iteration plan and the evaluation criteria that have been allocated to this iteration .
- Iteration Assessment Review. This informal milestone is conducted at the end of each iteration to assess the degree to which the iteration achieved its objectives and satisfied its evaluation criteria, to review iteration results, to review qualification test results (if part of the iteration), to determine the amount of rework to be done, and to review the impact of the iteration results on the plan for subsequent iterations.

The format and content of these minor milestones tend to be highly dependent on the project and the organizational culture. Figure identifies the various minor milestones to be considered when a project is being planned.

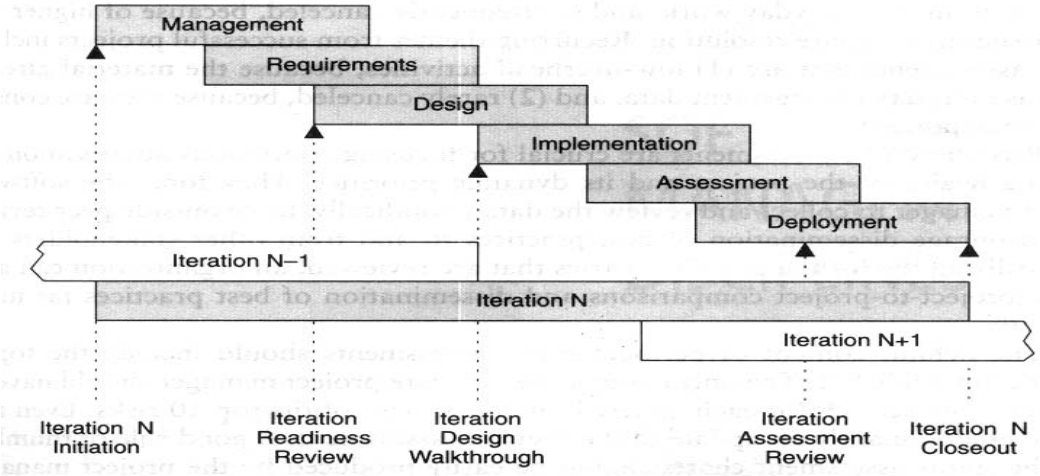


Fig. : Typical minor milestones in the life cycle of an iteration

Periodic Status Assessments

Periodic status assessments are management reviews conducted at regular intervals (monthly, quarterly) to address progress and quality indicators, ensure continuous attention to project dynamics, and maintain open communications among all stakeholders.

Periodic status assessments serve as project snapshots. While the period may vary, the recurring event forces the project history to be captured and documented. Status assessments provide the following:

- A mechanism for openly addressing, communicating, and resolving management issues, technical issues, and project risks
- Objective data derived directly from on-going activities and evolving product configurations
- A mechanism for disseminating process, progress, quality trends, practices, and experience information to and from all stakeholders in an open forum

  Periodic status assessments are crucial for focusing continuous attention on the evolving health of the project and its dynamic priorities. They force the software project manager to collect and review the data periodically, force outside peer review, and encourage dissemination of best practices to and from other stakeholders.

The default content of periodic status assessments should include the topics identified in Table .

Table : Default content of status assessment reviews

TOPIC	CONTENT
Personnel	Staffing plan vs. actuals
	Attritions, additions
Financial trends	Expenditure plan vs. actuals for the previous, current, and next major milestones
	Revenue forecasts
Top 10 risks	Issues and criticality resolution plans
	Quantification (cost, time, quality) of exposure
Technical progress	Configuration baseline schedules for major milestones
	Software management metrics and indicators
	Current change trends
	Test and quality assessments
Major milestone plans and results	Plan, schedule, and risks for the next major milestone
	Pass/fail results for all acceptance criteria
Total product scope	Total size, growth, and acceptance criteria perturbations

Q.7(d) Explain the three types of project environment. Write about four important environment disciplines that are critical to the management context and the success of a modern iterative development process **[5]**

Ans.: The project environment artifacts evolve through three discrete states: the prototyping environment, the development environment, and the maintenance environment.

1) **The prototyping environment** includes an architecture tested for prototyping project architectures to evaluate trade-offs during the inception and elaboration phases of the life cycle. This informal configuration of tools should be capable of supporting the following activities:
   - Performance trade-offs and technical risk analyses
   - Make/buy trade-offs and feasibility studies for commercial products
   - Fault tolerance/dynamic reconfiguration trade-offs
   - Analysis of the risks associated with transitioning to full-scale implementation
   - Development of test scenarios, tools, and instrumentation suitable for analyzing the requirements

2) **The development environment** should include a full suite of development tools needed to support the various process workflows and to support round-trip engineering to the maximum extent possible.

3) **The maintenance environment** should typically coincide with a mature version of the development environment. In some cases, the maintenance environment may be a subset of the development environment delivered as one of the project's end products.

There are four important environment disciplines that are critical to the management context and the success of a modern iterative development process:

1) Tools must be integrated to maintain consistency and traceability. Round-trip engineering is the term used to describe this key requirement for environments that support iterative development.
2) Change management must be automated and enforced to manage multiple iterations and to enable change freedom. Change is the fundamental primitive of iterative development.
3) Organizational infrastructures enable project environments to be derived from a common base of processes and tools. A common infrastructure promotes inter project consistency, reuse of training, reuse of lessons learned, and other strategic improvements to the organization's metaprocess.
4) Extending automation support for stakeholder environments enables further support for paperless exchange of information and more effective review of engineering artifacts.

Q.7(e) What are the basic characteristics of Good Metrics? **[5]**

Ans.: The basic characteristics of a good metric are as follows:

1) It is considered meaningful by the customer, manager, and performer. If any one of these stakeholders does not see the metric as meaningful, it will not be used. "The customer is always right" is a sales motto, not an engineering tenet. Customers come to software engineering providers because the providers are more expert than they are at developing and managing software. Customers will accept metrics that are demonstrated to be meaningful to the developer.
2) It demonstrates quantifiable correlation between process perturbations and business performance. The only real organizational goals and objectives are financial: cost reduction, revenue increase, and margin increase.
3) It is objective and unambiguously defined. Objectivity should translate into some form of numeric representation (such as numbers, percentages, ratios) as opposed to textual representations (such as excellent, good, fair, poor). Ambiguity is minimized through well-understood units of measurement (such as staff-month, SLOC, change, function point, class,

scenario, requirement), which are surprisingly hard to define precisely in the software engineering world.

4) It displays trends. This is an important characteristic. Understanding the change in a metric's value with respect to time, subsequent projects, subsequent releases, and so forth is an extremely important perspective, especially for today's iterative development models. It is very rare that a given metric drives the appropriate action directly. More typically, a metric presents a perspective. It is up to the decision authority (manager, team, or other information processing entity) to interpret the metric and decide what action is necessary.
5) It is a natural by-product of the process. The metric does not introduce new artifacts or overhead activities; it is derived directly from the mainstream engineering and management workflows.
6) It is supported by automation. Experience has demonstrated that the most successful metrics are those that are collected and reported by automated tools, in part because software tools require rigorous definitions of the data they process.

Q.7(f) How was the conventional software process characterized and how is a modern iterative development process framework characterized? [5]

Ans.: The conventional software process was characterized by the following:

- Sequentially transitioning from requirements to design to code to test.
- Achieving 100% completeness of each artifact at each life-cycle stage.
- Treating all requirements, artifacts, components, and so forth, as equals.
- Achieving high-fidelity traceability among all artifacts at each stage in the life cycle.

A modern iterative development process framework is characterized by the following:

- Continuous round-trip engineering from requirements to test at evolving levels of abstraction
- Achieving high-fidelity understanding of the drivers (the 20%) as early as practical
- Evolving the artifacts in breadth and depth based on risk management priorities
- Postponing completeness and consistency analyses until later in the life cycle.

❑ ❑ ❑ ❑ ❑