14. ASSIGNMENT TOPICS WITH MATERIALS
There are five improvements to the basic waterfall model that would eliminate most of the development risks are as follows:

a) Complete program design before analysis and coding begin (program design comes first):
   - By this technique, the program designer gives surety that the software will not fail because of storage, timing, and data fluctuations.
   - Begin the design process with program designer, not the analyst or programmers.
   - Write an overview document that is understandable, informative, and current so that every worker on the project can gain an elemental understanding of the system.

b) Maintain current and complete documentation (Document the design):
   - It is necessary to provide a lot of documentation on most software programs.
   - Due to this, helps to support later modifications by a separate test team, a separate maintenance team, and operations personnel who are not software literate.

c) Do the job twice, if possible (Do it twice):
If a computer program is developed for the first time, arrange matters so that the version finally delivered to the customer for operational deployment is actually the second version insofar as critical design/operations are concerned.

“Do it N times” approach is the principle of modern-day iterative development.

d) Plan, control, and monitor testing:

- The biggest user of project resources is the test phase. This is the phase of greatest risk in terms of cost and schedule.
- In order to carry out proper testing the following things to be done:
  - Employ a team of test specialists who were not responsible for the original design.
  - Employ visual inspections to spot the obvious errors like dropped minus signs, missing factors of two, jumps to wrong addresses.
  - Test every logic phase.
  - Employ the final checkout on the target computer.

e) Involve the customer:

- It is important to involve the customer in a formal way so that he has committed himself at earlier points before final delivery by conducting some reviews such as,
  - Preliminary software review during preliminary program design step.
  - Critical software review during program design.
  - Final software acceptance review following testing.

**Topic 2: Top 10 principles of conventional software management performance?**

**CONVENTIONAL SOFTWARE MANAGEMENT PERFORMANCE**

Barry Boehm’s Top 10 “Industrial Software Metrics”:

1) Finding and fixing a software problem after delivery costs 100 times more than finding and fixing the problem in early design phases.

2) You can compress software development schedules 25% of nominal (small), but no more.

3) For every $1 you spend on development, you will spend $2 on maintenance.

4) Software development and maintenance costs are primarily a function of the number of source lines of code.
5) Variations among people account for the biggest difference in software productivity.

6) The overall ratio of software to hardware costs is still growing. In 1955 it was 15:85; in 1985, 85:15.

7) Only about 15% of software development effort is devoted to programming.

8) Software systems and products typically cost 3 times as much per SLOC as individual software programs. Software-system products cost 9 times as much.

9) Walkthroughs catch 60% of the errors.

10) 80% of the contribution comes from 20% of the contributors.

   - 80% of the engineering is consumed by 20% of the requirements.
   - 80% of the software cost is consumed by 20% of the components.
   - 80% of the errors are caused by 20% of the components.
   - 80% of the software scrap and rework is caused by 20% of the errors.
   - 80% of the resources are consumed by 20% of the components.
   - 80% of the engineering is accomplished by 20% of the tools.
   - 80% of the progress is made by 20% of the people.

**Topic 3: List the symptoms of the project which are in trouble frequently and explain late risk resolution?**

Projects destined for trouble frequently exhibit the following symptoms

6. Protracted integration and late design breakage
7. Late risk resolution
8. Requirements – driven functional decomposition
9. Adversarial stakeholder relationships
10. Focus on documents and review meetings

**Late Risk Resolution**

A serious issue associated with the waterfall life cycle was the lack of early risk resolution. The risk profile of a waterfall model is, It includes four distinct periods of risk exposure, where risk is defined as “the probability of missing a cost, schedule, feature, or quality goal”.
Evolution of Software Economics

Economics means System of interrelationship of money, industry and employment.

SOFTWARE ECONOMICS:

The cost of the software can be estimated by considering the following things as parameters to a function.

1) Size: Which is measured in terms of the number of Source Lines of Code or the number of function points required to develop the required functionality?

2) Process: Used to produce the end product, in particular the ability of the process is to avoid no value adding activities (rework, bureaucratic delays, and communications overhead).

3) Personnel: The capabilities of software engineering personnel, and particularly their experience with the computer science issues and the application domain issues of the project.
4) Environment: Which is made up of the tools and techniques available to support efficient software development and to automate the process

5) Quality: It includes its features, performance, reliability, and flexibility. The relationship among these parameters and estimated cost can be calculated by using,

\[
\text{Effort} = (\text{Personnel}) \times (\text{Environment}) \times (\text{Quality}) \times (\text{Size Process})
\]

One important aspect of software economics is that the relationship between effort and size exhibits a diseconomy of scale and is the result of the process exponent being greater than 1.0. Converse to most manufacturing processes, the more software you build, the more expensive it is per unit item.

There are three generations of basic technology advancement in tools, components, and processes are available.


2) Transition: 1980 and 1990, software engineering. Organizations used more-repeatable processes and off-the-shelf tools, and mostly (>70%) custom components built in higher level languages. Some of the components (<30%) were available as commercial products like, OS, DBMS, Networking and GUI.

3) Modern practices: 2000 and later, software production.

- 70% component-based,
- 30% custom

Conventional Transition Modern Practices

- Waterfall model - Process improvement - Iterative development
- Functional design - Encapsulation - based - Component-based
- Diseconomy of scale- Diseconomy of scale- ROI

Environments / tools: Custom Off-the-shelf, separate Off-the-shelf, Integrated

Size: - 100% custom 30% component-based 70% component-based
- 70% custom 30% custom
UNIT-II
Topic 1 : The principles of conventional software engineering?

1) Make quality #1: Quality must be quantified and mechanisms put into place to motivate its achievement.

2) High-quality software is possible: In order to improve the quality of the product we need to involving the customer, select the prototyping, simplifying design, conducting inspections, and hiring the best people.

3) Give products to customers early: No matter how hard you try to learn user’s 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.

4) Determine the problem before writing the requirements: Whenever a problem is raised most engineers provide a solution. Before we try to solve a problem, be sure to explore all the alternatives and don’t be blinded by the understandable solution.

5) Evaluate design alternatives: After the requirements are greed upon, we must examine a variety of architectures and algorithms and choose the one which is not used earlier.

6) Use an appropriate process model: For every project, there are so many prototypes (process models). So select the best one that is exactly suitable to our project.

7) Use different languages for different phases: Our industry’s main aim is to provide simple solutions to complex problems. In order to accomplish this goal choose different languages for different modules/phases if required.

8) Minimize intellectual distance: We have to design the structure of a software is as close as possible to the real-world structure.

9) Put techniques before tools: An un disciplined software engineer with a tool becomes a dangerous, undisciplined software engineer.

10) Get it right before you make it faster: It is very easy to make a working program run faster than it is to make a fast program work. Don’t worry about optimization during initial coding.

11) Inspect the code: Examine the detailed design and code is a much better way to find the errors than testing.

12) Good management is more important than good technology.
13) People are the key to success: Highly skilled people with appropriate experience, talent, and training are key. The right people with insufficient tools, languages, and process will succeed.

14) Follow with care: Everybody is doing something but does not make it right for you. It may be right, but you must carefully assess its applicability to your environment.

15) Take responsibility: When a bridge collapses we ask “what did the engineer do wrong?” Similarly if the software fails, we ask the same. So the fact is in every engineering discipline, the best methods can be used to produce poor results and the most out of date methods to produce stylish design.

16) Understand the customer’s priorities. It is possible the customer would tolerate 90% of the functionality delivered late if they could have 10% of it on time.

17) The more they see, the more they need. The more functionality (or performance) you provide a user, the more functionality (or performance) the user wants.

18) Plan to throw one away. One of the most important critical success factors is whether or not a product is entirely new. Such brand-new applications, architectures, interfaces, or algorithms rarely work the first time.

19) Design for change. The architectures, components, and specification techniques you use must accommodate change.

20) Design without documentation is not design. I have often heard software engineers say, “I have finished the design. All that is left is the documentation.”

21. Use tools, but be realistic. Software tools make their users more efficient.

22. Avoid tricks. Many programmers love to create programs with tricks- constructs that perform a function correctly, but in an obscure way. Show the world how smart you are by avoiding tricky code.

23. Encapsulate. Information-hiding is a simple, proven concept that results in software that is easier to test and much easier to maintain.

24. Use coupling and cohesion. Coupling and cohesion are the best ways to measure software’s inherent maintainability and adaptability.

25. Use the McCabe complexity measure. Although there are many metrics available to report the inherent complexity of software, none is as intuitive and easy to use as Tom McCabe’s.
26. Don’t test your own software. Software developers should never be the primary testers of their own software.

27. Analyze causes for errors. It is far more cost-effective to reduce the effect of an error by preventing it than it is to find and fix it. One way to do this is to analyze the causes of errors as they are detected.

28. Realize that software’s entropy increases. Any software system that undergoes continuous change will grow in complexity and become more and more disorganized.

29. People and time are not interchangeable. Measuring a project solely by person-months makes little sense.

30) Expert excellence. Your employees will do much better if you have high expectations for them.

**Topic 2 : the top five principles of modern software management?**

**principles of modern software management :**

1) Base the process on an architecture-first approach: (Central design element)
   - Design and integration first, then production and test

2) Establish an iterative life-cycle process: (The risk management element)
   - Risk control through ever-increasing function, performance, quality.

With today’s sophisticated systems, it is not possible to define the entire problem, design the entire solution, build the software, then test the end product in sequence. Instead, and iterative process that refines the problem understanding, an effective solution, and an effective plan over several iterations encourages balanced treatment of all stakeholder 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: (The technology element)

Moving from LOC mentally to component-based mentally 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 control element) - Metrics, trends, process instrumentation. The dynamics of iterative development, include concurrent workflows by different teams working on shared artifacts, necessitates objectively controlled baseline.

5) Enhance change freedom through tools that support round-trip engineering: (The automation element) - Complementary tools, integrated environment

Round-trip engineering is the environment support necessary to automate and synchronize engineering information in different formats. Change freedom is necessary in an iterative process.
Improving software economics

- It is not that much easy to improve the software economics but also difficult to measure and validate.
Software Project Management

- There are many aspects are there in order to improve the software economics they are, Size, Process, Personnel, Environment and quality.
- These parameters (aspects) are not independent they are dependent. For example, tools enable size reduction and process improvements, size-reduction approaches lead to process changes, and process improvements drive tool requirements.
- GUI technology is a good example of tools enabling a new and different process. GUI builder tools permitted engineering teams to construct an executable user interface faster and less cost.
- Two decades ago, teams developing a user interface would spend extensive time analyzing factors, screen layout, and screen dynamics. All this would done on paper. Where as by using GUI, the paper descriptions are not necessary.
- Along with these five basic parameters another important factor that has influenced software technology improvements across the board is the ever-increasing advances in hardware Performance.
Topic 4: Explain the “achieving required quality”?

Achieving required quality: Key elements that improve overall software quality include the following:

- Focusing on powerful requirements and critical use case early in the life
- Focusing on requirements completeness and traceability late in the life cycle
- Focusing throughout the life cycle on a balance between requirements evolution, design evolution, and plan evolution
● Using metrics and indicators to measure the progress and quality of an architecture as it evolves from high-level prototype into a fully biddable product

● Providing integrated life-cycle environments that support early and continuous configuration control, change management, rigorous design methods, document automation, and regression test automation

● Using visual modeling and higher level languages that support architectural control, abstraction, reliable programming, reuse, and self-documentation

● Early and continuous close look into performance issues through demonstration-based evaluations

In order to evaluate the performance the following sequence of events are necessary,

● Project inception

● Initial design review

● Mid-life-cycle design review

● Integration and test
Table 3-5. General quality improvements with a modern process

<table>
<thead>
<tr>
<th>QUALITY DRIVER</th>
<th>CONVENTIONAL PROCESS</th>
<th>MODERN ITERATIVE PROCESSES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements misunderstanding</td>
<td>Discovered late</td>
<td>Resolved early</td>
</tr>
<tr>
<td>Development risk</td>
<td>Unknown until late</td>
<td>Understood and resolved early</td>
</tr>
<tr>
<td>Commercial components</td>
<td>Mostly unavailable</td>
<td>Still a quality driver, but trade-offs must be resolved early in the life cycle</td>
</tr>
<tr>
<td>Change management</td>
<td>Late in the life cycle, chaotic and malignant</td>
<td>Early in the life cycle, straightforward and benign</td>
</tr>
<tr>
<td>Design errors</td>
<td>Discovered late</td>
<td>Resolved early</td>
</tr>
<tr>
<td>Automation</td>
<td>Mostly error-prone manual procedures</td>
<td>Mostly automated, error-free evolution of artifacts</td>
</tr>
<tr>
<td>Resource adequacy</td>
<td>Unpredictable</td>
<td>Predictable</td>
</tr>
<tr>
<td>Schedules</td>
<td>Overconstrained</td>
<td>Tunable to quality, performance, and technology</td>
</tr>
<tr>
<td>Target performance</td>
<td>Paper-based analysis or separate simulation</td>
<td>Executing prototypes, early performance feedback, quantitative understanding</td>
</tr>
<tr>
<td>Software process rigor</td>
<td>Document-based</td>
<td>Managed, measured, and tool-supported</td>
</tr>
</tbody>
</table>

Topic 5: Explain the peer inspections?

Peer inspections are frequently overhyped as the key aspect of a quality system. In my experience, peer reviews are valuable as secondary mechanisms, but they are rarely significant contributors to quality compared with the following primary quality mechanisms and indicators, which should be emphasized in the management process:

- Transitioning engineering information from one artifact set to another, thereby assessing the consistency, feasibility, understandability, and technology constraints inherent in the engineering artifacts
- Major milestone demonstrations that force the artifacts to be assessed against tangible criteria in the context of relevant use cases
● Environment tools (compilers, debuggers, analyzers, automated test suites) that ensure representation rigor, consistency, completeness, and change control
● Life-cycle testing for detailed insight into critical trade-offs, acceptance criteria, and requirements compliance
● Change management metrics for objective insight into multiple-perspective change trends and convergence or divergence from quality and progress goals

Although I believe that inspections are overemphasized, in certain cases they provide a significant return. One value of inspections is in the professional development of a team. It is generally useful to have the products of junior team members reviewed by senior mentors. Putting the products of amateurs into the hands of experts and vice versa is a good mechanism for accelerating the acquisition of knowledge and skill in new personnel. Gross blunders can be caught and feedback can be appropriately channeled, so that bad practices are not perpetuated. This is one of the best ways for junior software engineers to learn.

Inspections are also a good vehicle for holding authors accountable for quality products. All authors of software and documentation should have their products scrutinized as a natural by-product of the process. Therefore, the coverage of inspections should be across all authors rather than across all components. Junior authors need to have a random component inspected periodically, and they can learn by inspecting the products of senior authors. Varying levels of informal inspection are performed continuously when developers are reading or integrating software with another author's software, and during testing by independent test teams.

However, this "inspection" is much more tangibly focused on integrated and executable aspects of the overall system.

Significant or substantial design errors or architecture issues are rarely obvious from a superficial review unless the inspection is narrowly focused on a particular issue. And most inspections are superficial. Today's systems are highly complex, with innumerable components, concurrent execution, distributed resources, and other equally demanding dimensions of complexity. It would take human intellects similar to those of world-class chess players to comprehend the dynamic interactions within some simple software systems under some simple use cases.
Consequently, random human inspections tend to degenerate into comments on style and first-order semantic issues. They rarely result in the discovery of real performance bottlenecks, serious control issues (such as deadlocks, races, or resource contention), or architectural weaknesses (such as flaws in scalability, reliability, or interoperability). In all but trivial cases, architectural issues are exposed only through more rigorous engineering activities such as the following:

- Analysis, prototyping, or experimentation
- Constructing design models
- Committing the current state of the design model to an executable implementation
- Demonstrating the current implementation strengths and weaknesses in the context of critical subsets of the use cases and scenarios
- Incorporating lessons learned back into the models, use cases, implementations, and plans

Achieving architectural quality is inherent in an iterative process that evolves the artifact sets together in balance. The checkpoints along the way are numerous, including human review and inspections focused on critical issues. But these inspections are not the primary checkpoints. Early life-cycle artifacts are certainly more dependent on subjective human review than later ones are. Focusing a large percentage of a project's resources on human inspections is bad practice and only perpetuates the existence of low-value-added box checkers who have little impact on project success. Look at any successful software effort and ask the key designers, testers, or developers about the discriminators of their success. It is unlikely that any of them will cite meetings, inspections, or documents.

Quality assurance is everyone's responsibility and should be integral to almost all process activities instead of a separate discipline performed by quality assurance specialists. Evaluating and assessing the quality of the evolving engineering baselines should be the job of an engineering team that is independent of the architecture and development team. Their life-cycle assessment of the evolving artifacts would typically include change management, trend analysis, and testing, as well as inspection.
UNIT-III

Topic 1: phases of life cycle process?

Engineering and Production Stages:

To achieve economics of scale and higher return on investment, we must move toward a software manufacturing process which is determined by technological improvements in process automation and component based development.

There are two stages in the software development process:

1) The engineering stage: Less predictable but smaller teams doing design and production activities. This stage is decomposed into two distinct phases inception and elaboration.

2) The production stage: More predictable but larger teams doing construction, test, and deployment activities. This stage is also decomposed into two distinct phases construction and transition.

These four phases of lifecycle process are loosely mapped to the conceptual framework of the spiral model as shown in the following figure.

In the above figure the size of the spiral corresponds to the inactivity of the project with respect to the breadth and depth of the artifacts that have been developed.
This inertia manifests itself in maintaining artifact consistency, regression testing, documentation, quality analyses, and configuration control.

Increased inertia may have little, or at least very straightforward, impact on changing any given discrete component or activity.

However, the reaction time for accommodating major architectural changes, major requirements changes, major planning shifts, or major organizational perturbations clearly increases in subsequent phases.

**Topic 2: inception and elaboration phases?**

**Inception Phase:**
The main goal of this phase is to achieve agreement among stakeholders on the life-cycle objectives for the project.

The overriding goal of the inception phase is to achieve concurrence among all stakeholders on the lifecycle objectives for the project. The primary objectives of the inception phase include:

-- Establishing the project's software scope and boundary conditions, including an operational concept, acceptance criteria and what is intended to be in the product and what is not.
-- Discriminating the critical use cases of the system, the primary scenarios of operation that will drive the major design trade-offs.
-- Exhibiting, and maybe demonstrating, at least one candidate architecture against some of the primary scenarios
-- Estimating the overall cost and schedule for the entire project (and more detailed estimates for the elaboration phase that will immediately follow)
-- Estimating potential risks (the sources of unpredictability)

The essential activities of the inception phase are:

-- Formulating the scope of the project. This involves capturing the context and the most important requirements and constraints to such an extent that you can derive acceptance criteria for the end product.
-- Planning and preparing a business case. Evaluating alternatives for risk management, staffing, project plan, and cost/schedule/profitability trade-offs.
--Synthesizing a candidate architecture, evaluating trade-offs in design, and in make/buy/reuse, so that cost, schedule and resources can be estimated.

The outcome of the inception phase is:
--A vision document: a general vision of the core project's requirements, key features, main constraints.
--The use-case model survey (identifying all use cases that can be identified at this early stage).
--An initial glossary.
--An initial business case, which includes:
  --Business context.
  --Success criteria (revenue projection, market recognition, and so on).
  --Financial forecast.
--An initial risk assessment.
--A project plan, showing phases, and iterations.

**Elaboration Phase:**
- It is the most critical phase among the four phases.
- Depending upon the scope, size, risk, and freshness of the project, an executable architecture prototype is built in one or more iterations.

The primary objectives of the elaboration phase include:

i) Defining, validating and base lining the architecture as rapidly as practical
ii) Base lining the vision
iii) Base lining a high-fidelity plan for the construction phase
iv) Demonstrating that the baseline architecture will support this vision for reasonable cost in a reasonable time.

**The essential activities of the elaboration phase are:**
v) Elaborating the vision establishing a solid understanding of the most critical use cases that drive the architectural and planning decisions.
i) Elaborating the process and the infrastructure, the development environment. Putting in place the process, tools and automation support.
ii) Elaborating the architecture and selecting components. Potential components are evaluated and
the make/buy/reuse decisions sufficiently understood to determine the construction phase cost
and schedule with confidence. The selected architectural components are integrated and assessed
against the primary scenarios. Lessons learned from these activities may well result in a redesign
of the architecture, taking into consideration alternative designs or reconsideration of the
requirements.

**The outcome of the elaboration phase is:**
i) A use-case model (80% complete) - all use cases having been identified in the use-case model
survey, all actors having been identified, and most use-case descriptions (requirements capture)
having been developed.

ii) Supplementary requirements capturing the non-functional requirements and any requirements
that are not associated with a specific use case.

iii) An executable architecture and accompanying documentation - the Software Architecture
Document, including use-case descriptions (design) for a subset of use cases (use-case view), and
an updated glossary.

   A revised business case.
   A revised risk list.
   A development plan for the overall project, including the coarse-grained project plan,
   showing iterations and evaluation criteria for each iteration. A preliminary user manual
   (optional).

**Topic 3: Summarize the artifact sets in a diagrammatical manner and
management artifacts?**

**THE ARTIFACT SETS:** In order to manage the development of a complete software system, we
need to gather distinct collections of information and is organized into artifact sets.

i) Set represents a complete aspect of the system where as artifact represents interrelated
information that is developed and reviewed as a single entity.

ii) The artifacts of the process are organized into five sets:

6. Management
7. Requirements
8. Design
9. Implementation
10. Deployment

here the management artifacts capture the information that is necessary to synchronize stakeholder expectations. Where as the remaining four artifacts are captured rigorous notations that support automated analysis and browsing.

<table>
<thead>
<tr>
<th>Requirements Set</th>
<th>Design Set</th>
<th>Implementation Set</th>
<th>Deployment Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Vision document</td>
<td>1. Design model(s)</td>
<td>1. Source code baselines</td>
<td>1. Integrated product executable baselines</td>
</tr>
<tr>
<td>2. Requirements model(s)</td>
<td>2. Test model</td>
<td>2. Associated compile-time files</td>
<td>2. Associated run-time files</td>
</tr>
</tbody>
</table>

**MANAGEMENT ARTIFACTS:**

Development of WBS is dependent on product management style, organizational culture, custom performance, financial constraints and several project specific parameters.

i) The WBS is the architecture of project plan. It encapsulate change and evolve with appropriate level of details.

ii) A WBS is simply a hierarchy of elements that decomposes the project plan into discrete work task.

iii) A WBS provides the following information structure
iv) A delineation of all significant tasks.

v) A clear task decomposition for assignment of responsibilities.

vi) A framework for scheduling, debugging and expenditure tracking.

vii) Most systems have first level decomposition subsystem. subsystems are then decomposed into their components.

viii) Therefore WBS is a driving vehicle for budgeting and collecting cost.

ix) The structure of cost accountability is a serious project planning constraints.

**Business Case:**

The business case artifact provides all the information necessary to determine whether the project is worth investing in. It details the expected revenue, expected cost, technical and management plans, and backup data necessary to demonstrate the risks and realism of the plans.

In large contractual procurements, the business case may be implemented in a full-scale proposal with multiple volumes of information. In a small-scale endeavor for a commercial product, it may be implemented in a brief plan with an attached spreadsheet.

<table>
<thead>
<tr>
<th>I. Context (domain, market, scope)</th>
</tr>
</thead>
<tbody>
<tr>
<td>II. Technical approach</td>
</tr>
<tr>
<td>A. Feature set achievement plan</td>
</tr>
<tr>
<td>B. Quality achievement plan</td>
</tr>
<tr>
<td>C. Engineering trade-offs and technical risks</td>
</tr>
<tr>
<td>III. Management approach</td>
</tr>
<tr>
<td>A. Schedule and schedule risk assessment</td>
</tr>
<tr>
<td>B. Objective measures of success</td>
</tr>
<tr>
<td>IV. Evolutionary appendixes</td>
</tr>
<tr>
<td>A. Financial forecast</td>
</tr>
<tr>
<td>1. Cost estimate</td>
</tr>
<tr>
<td>2. Revenue estimate</td>
</tr>
<tr>
<td>3. Bases of estimates</td>
</tr>
</tbody>
</table>

**Figure 6-4. Typical business case outline**

**Release Specifications:**

The scope, plan, and objective evaluation criteria for each baseline release are derived from the vision statement as well as many other sources (make/buy analyses, risk management concerns, architectural considerations, shots in the dark, implementation) these artifacts are intended to
Software development plan:
It is the defining document for the project’s process. It must comply with the organization standards, evolve along with the design and requirements and be used consistently across all subordinate organizations doing software development.
Release descriptions:

These documents describe the results of each release, including performance against each of the evolution criteria in the corresponding release specifications. This document also includes a metrics summary that quantifies its quality in absolute and relative terms.

Status assessments:

This provides periodic snapshots of project health and status, including the software project manager’s risk assessment, quality indicators, and management indicators. This document provides the critical and mechanism for managing everyone’s expectations throughout the lifecycle, for addressing, communicating, and resolving management issues, technical issues, and project risks and for capturing project history.
Environment: An important emphasis of a modern approach is to define the development and maintenance environment as a first class artifact of the process. A robust integrated development environment must support automation of the development process.

Deployment: This can take many forms. Depending on the project it could include several document subsets for transitioning the product into operational status.

Topic 4: Different aspects of architecture in management perspective.

ARCHITECTURE: A MANAGEMENT PERSPECTIVE
i) The most critical and technical product of a software project is its architecture
ii) If a software development team is to be successful, the inter project communications, as captured in software architecture, must be accurate and precise.

From the management point of view, three different aspects of architecture

1. An architecture (the intangible design concept) is the design of software system it includes all engineering necessary to specify a complete bill of materials. Significant make or buy decisions are resolved, and all custom components are elaborated so that individual component costs and construction/assembly costs can be determined with confidence.

2. An architecture baseline (the tangible artifacts) is a slice of information across the engineering artifact sets sufficient to satisfy all stakeholders that the vision (function and quality) can be achieved within the parameters of the business case (cost, profit, time, technology, people).

3. An architectural description is an organized subset of information extracted from the design set model's. It explains how the intangible concept is realized in the tangible artifacts.

The number of views and level of detail in each view can vary widely. For example the architecture of the software architecture of a small development tool.

There is a close relationship between software architecture and the modern software development process because of the following reasons:

1. A stable software architecture is nothing but a project milestone where critical make/buy decisions should have been resolved. The life-cycle represents a transition from the engineering stage of a project to the production stage.

2. Architecture representation provide a basis for balancing the trade-offs between the problem space (requirements and constraints) and the solution space (the operational product).
3. The architecture and process encapsulate many of the important communications among individuals, teams, organizations, and stakeholders.

4. Poor architecture and immature process are often given as reasons for project failure.

5. In order to proper planning, a mature process, understanding the primary requirements and demonstrable architecture are important fundamentals.

6. Architecture development and process definition are the intellectual steps that map the problem to a solution without violating the constraints; they require human innovation and cannot be automated.
UNIT-IV

**Topic 1 : 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. An individual iteration's workflow, generally includes the following sequence:

i) 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

ii) Environment: evolving the software change order database to reflect all new baselines and changes to existing baselines for all product, test, and environment components

---

**Figure 8-2. The workflow of an iteration**

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 (previous versions).

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. As with any sequence of a software development workflow, many of the activities occur concurrently. For example, requirements analysis is not done all in one continuous lump; it intermingles with management, design, implementation, and so forth.

Iterations in the inception and elaboration phases focus on management, requirements, and design activities. Iterations in the construction phase focus on design, implementation, and assessment. Iterations in the transition phase focus on assessment and deployment. Figure 8-3 shows the emphasis on different activities across the life cycle.

These descriptions are pretty simplistic. In practice, the various sequences and overlaps among iterations become more complex. The terms iteration and increment deal with some of the pragmatic considerations. An iteration represents the state of the overall architecture and the complete deliverable system. An increment represents the current work in progress that will be combined with the preceding iteration to form the next iteration. Figure 8-4, an example of a simple development life cycle, illustrates the difference between iterations and increments. This example also illustrates a typical build sequence from the perspective of an abstract layered architecture.
Progress can be measured as the % of components under configuration control, the % of demonstrable use cases, etc.

**Figure 8-4.** A typical build sequence associated with a layered architecture
**Topic 2 : Management reviews that are conducted throughout the process.**

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

Major milestones. These system wide events are held at the end of each development phase. They provide visibility to system wide issues, synchronize the management and engineering perspectives, and verify that the aims of the phase have been achieved.

Minor milestones. These iteration-focused events are conducted to review the content of an iteration in detail and to authorize continued work.

Status assessments. These periodic events provide management with frequent and regular insight into the progress being made, illustrates a typical sequence of project checkpoints for a relatively large project.

<table>
<thead>
<tr>
<th>Inception</th>
<th>Elaboration</th>
<th>Construction</th>
<th>Transition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iteration 1</td>
<td>Iteration 2</td>
<td>Iteration 3</td>
<td>Iteration 4</td>
</tr>
</tbody>
</table>

**Figure 9-1. A typical sequence of life-cycle checkpoints**

**Topic 3 : the initial operational capability milestone and product release milestone in detail? 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. Acceptance testing can be done incrementally across multiple iterations or can be completed entirely during the transition phase is not necessarily the completion of the construction phase.
**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. Software quality metrics are reviewed to determine whether quality is sufficient for transition to the support organization.

**Topic 4 : Detail about WBS?**

**work breakdown structure**: WBS is simply a hierarchy of elements that decomposes the project plan into the discrete work tasks. A WBS provides the following information structure:

- A delineation of all significant work
- A clear task decomposition for assignment of responsibilities
- A framework for scheduling, budgeting, and expenditure tracking

Many parameters can drive the decomposition of work into discrete tasks: product subsystems, components, functions, organizational units, life-cycle phases, even geographies. Most systems have a first-level decomposition by subsystem. Subsystems are then decomposed into their components, one of which is typically the software.

**CONVENTIONAL WBS ISSUES**

Conventional work breakdown structures frequently suffer from three fundamental flaws.

- They are prematurely structured around the product design.
- They are prematurely decomposed, planned, and budgeted in either too much or too little detail.
- They are project-specific, and cross-project comparisons are usually difficult or impossible.

Conventional work breakdown structures are prematurely structured around the product design. Figure shows a typical conventional WBS that has been structured primarily around the subsystems of its product architecture, then further decomposed into the components of each subsystem. A WBS is the architecture for the financial plan.

Conventional work breakdown structures are prematurely decomposed, planned, and budgeted in either too little or too much detail. Large software projects tend to be over planned and small
Software Project Management

projects tend to be under planned. The basic problem with planning too much detail at the outset is that the detail does not evolve with the level of fidelity in the plan. Conventional work breakdown structures are project-specific, and cross-project comparisons are usually difficult or impossible. With no standard WBS structure, it is extremely difficult to compare plans, financial data, schedule data, organizational efficiencies, cost trends, productivity trends, or quality trends across multiple projects.
Management
System requirements and design
Subsystem 1
  Component 11
    Requirements
    Design
    Code
    Test
    Documentation
    . . . (similar structures for other components)
Component 1N
  Requirements
  Design
  Code
  Test
  Documentation
  . . . (similar structures for other subsystems)
Subsystem M
  Component M1
    Requirements
    Design
    Code
    Test
    Documentation
    . . . (similar structures for other components)
Component MN
  Requirements
  Design
  Code
  Test
  Documentation
Integration and test
  Test planning
  Test procedure preparation
  Testing
  Test reports
Other support areas
  Configuration control
  Quality assurance
  System administration

Figure 10-1. Conventional work breakdown structure, following the product hierarchy
EVOLUTIONARY WORK BREAKDOWN STRUCTURES:

An evolutionary WBS should organize the planning elements around the process framework rather than the product framework. The basic recommendation for the WBS is to organize the hierarchy as follows:

- First-level WBS elements are the workflows (management, environment, requirements, design, implementation, assessment, and deployment).
- Second-level elements are defined for each phase of the life cycle (inception, elaboration, construction, and transition).
- Third-level elements are defined for the focus of activities that produce the artifacts of each phase.

This recommended structure provides one example of how the elements of the process framework can be integrated into a plan. It provides a framework for estimating the costs and schedules of each element, allocating them across a project organization, and tracking expenditures.

The structure shown is intended to be merely a starting point. It needs to be tailored to the specifics of a project in many ways:

- Scale. Larger projects will have more levels and substructures.
- Organizational structure. Projects that include subcontractors or span multiple organizational entities may introduce constraints that necessitate different WBS allocations.
- Degree of custom development. Depending on the character of the project, there can be very different emphases in the requirements, design, and implementation workflows.
- Business context. Projects developing commercial products for delivery to a broad customer base may require much more elaborate substructures for the deployment element.
- Precedent experience. Very few projects start with a clean slate. Most of them are developed as new generations of a legacy system (with a mature WBS) or in the context of existing organizational standards (with preordained WBS expectations).

The WBS decomposes the character of the project and maps it to the life cycle, the budget,
and the personnel. Reviewing a WBS provides insight into the important attributes, priorities, and structure of the project plan.

Another important attribute of a good WBS is that the planning fidelity inherent in each element is commensurate with the current life-cycle phase and project state. One of the primary reasons for organizing the default WBS the way I have is to allow for planning elements that range from planning packages (rough budgets that are maintained as an estimate for future elaboration rather than being decomposed into detail) through fully planned activity networks (with a well-defined budget and continuous assessment of actual versus planned expenditures).

**Topic 5 : Activities of software management team and software activities team.**

**Software management team activities :**

The software management team carries the burden of delivering win conditions to all stakeholders. The software management team is responsible for planning the effort and conducting the plan. The software management team takes ownership of all aspects of quality. It is responsible for attaining (reach / achieve) and maintaining balance among these aspects so that the overall solution is adequate for all stakeholders.
Software Management

Artifacts
• Business case
• Vision
• Software development plan
• Work breakdown structure
• Status assessments
• Requirements set

Systems engineering
Financial administration
Quality assurance

Responsibilities
• Resource commitments
• Personnel assignments
• Plans, priorities
• Stakeholder satisfaction
• Scope definition
• Risk management
• Project control

Life-Cycle Focus

<table>
<thead>
<tr>
<th>Inception</th>
<th>Elaboration</th>
<th>Construction</th>
<th>Transition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elaboration phase planning</td>
<td>Construction phase planning</td>
<td>Transition phase planning</td>
<td>Customer satisfaction</td>
</tr>
<tr>
<td>Team formulation</td>
<td>Full staff recruitment</td>
<td>Construction plan</td>
<td>Contract closure</td>
</tr>
<tr>
<td>Contract baselining</td>
<td>Risk resolution</td>
<td>optimization</td>
<td>Sales support</td>
</tr>
<tr>
<td>Architecture costs</td>
<td>Product acceptance criteria</td>
<td>Risk management</td>
<td>Next-generation planning</td>
</tr>
<tr>
<td></td>
<td>Construction costs</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Software Architecture Team activities

The software architecture team is responsible for the architecture. For any project, the skill of the software architecture team is crucial. With a good architecture team, an average development team can also succeed but if the architecture is weak, even an expert development team of superstar programmers may not succeed. In most projects, the inception and elaboration phases will be dominated by the software management team and the software architecture team.
### Software Architecture

**Artifacts**
- Architecture description
- Requirements set
- Design set
- Release specifications

**Responsibilities**
- Requirements trade-offs
- Design trade-offs
- Component selection
- Initial integration
- Technical risk resolution

### Life-Cycle Focus

<table>
<thead>
<tr>
<th>Inception</th>
<th>Elaboration</th>
<th>Construction</th>
<th>Transition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture prototyping</td>
<td>Architecture baselining</td>
<td>Architecture maintenance</td>
<td>Architecture maintenance</td>
</tr>
<tr>
<td>Make/buy trade-offs</td>
<td>Primary scenario demonstration</td>
<td>Multiple-component issue resolution</td>
<td>Multiple-component issue resolution</td>
</tr>
<tr>
<td>Primary scenario definition</td>
<td>Make/buy trade-off baselining</td>
<td>Performance tuning</td>
<td>Performance tuning</td>
</tr>
<tr>
<td>Architecture evaluation criteria definition</td>
<td></td>
<td>Quality improvements</td>
<td>Quality improvements</td>
</tr>
</tbody>
</table>
Software Project Management

UNIT-V

Topic 1: Seven core metrics?

Need for Software Metrics:
1. Software metrics are needed for calculating the cost and schedule of a software product with great accuracy.
2. Software metrics are required for making an accurate estimation of the progress.
3. The metrics are also required for understanding the quality of the software product.

Indicators:
An indicator is a metric or a group of metrics that provides an understanding of the software process or software product or a software project. A software engineer assembles measures and produce metrics from which the indicators can be derived. Two types of indicators are:
(i) Management indicators.
(ii) Quality indicators.

Management Indicators: The management indicators i.e., technical progress, financial status and staffing progress are used to determine whether a project is on budget and on schedule. The management indicators that indicate financial status are based on earned value system.

4. Work and progress (work performed over time)
5. Budgeted cost and expenditures (cost incurred over time)
6. Staffing and team dynamics (personnel changes over time)

Quality Indicators: The quality indicators are based on the measurement of the changes occurred in software.

5. Change traffic and stability (changes traffic over time)
6. Breakage and modularity (average breakage per change over time)
7. Rework and adaptability (average rework per change over time)
8. Mean time between failures (MTBF) and maturity (defect rate over time)

Seven Core Metrics of Software Project
Software Project Management

Software metrics instrument the activities and products of the software development/integration process. Metrics values provide an important perspective for managing the process. The most useful metrics are extracted directly from the evolving artifacts. There are seven core metrics that are used in managing a modern process.

| TABLE 13-1. 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, SCO |
| 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 month added, people per month leaving |
| Change traffic and stability | Iteration planning, 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 |

Topic 2: Top software management principles?

Principles of Software Management

1. Process must be based on architecture-first approach
2. Develop an iterative life-cycle process that identifies the risks at an early stage
3. After the design methods in-order to highlight components-based development.
4. Create a change management Environment
5. Improve change freedom with the help of automated tools that support round-trip engineering.
7. Process must be implemented or obtaining objective quality control and estimation of progress.
8. Implement a Demonstration-based Approach for Estimation of intermediately Artifacts
9. Develop a configuration process that should be economically scalable

**Topic 3: Next Generation Cost Models?**

Next generation software 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. The passionate overhype or under hype, depending on your perspective, makes it difficult to separate facts from exaggeration. 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. This section provides my perspective on how a software cost model should be structured to best support the estimation of a modern software process. There are cost models and techniques in the industry that can support subsets of this approach. My software cost model is all theory; I have 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. Figure 16-1 summarizes an hypothesized cost model for an architecture-first development process.
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. My reasoning is that the larger the system, the more opportunity there is to exploit automation and to reuse common processes, components, and architectures.
There are many solutions to any given problem, as illustrated in Figure 16-2, each with a different value proposition. Cost is a key discriminator among potential solutions. Cost estimates that are more accurate and more precise can be derived from specific solutions to problems.

I expect two major improvements in next-generation software cost estimation models:
1. Separation of the engineering stage from the production stage will force estimators to differentiate between architectural scale and implementation size. This will permit greater accuracy and more-honest precision in life-cycle estimates.
2. Rigorous design notations such as UML will offer an opportunity to define units of measure for scale that are more standardized and therefore can be automated and tracked. These measures can also be traced more straightforwardly into the costs of production.
16. Unit wise-Question bank
Two Marks Questions With Answers

1). What are the two basic steps of building a program?

Answer:

2). Justify the statement “walkthroughs catch 60% of the errors”?

Answer: This may be true. However given metric 1, walkthroughs are not catching the errors that matter and certainly are not catching them early enough in the lifecycle. All defects are not created equal. In general, walkthroughs and other forms of human inspection are good at catching surface problems and style issues.

3). What are the three common debates among developers and vendors of software cost estimation models and tools?

Answer:

- Which cost estimation model to use
- Whether to measure software size in source lines of code or function points
- Which constitutes a good estimate

4). Explain one improvement of waterfall model in detail?

Answer:

Complete program design before analysis and coding begin (program design comes first):

- By this technique, the program designer gives surety that the software will not fail because of storage, timing, and data fluctuations.
- Begin the design process with program designer, not the analyst or programmers.
- Write an overview document that is understandable, informative, and current so that every worker on the project can gain an elemental understanding of the system.

5). Give a brief note on prorated integration and late design breakage?
Software Project Management

Answer:
In the conventional model, the entire system was designed on paper, then implemented all at once, then integrated. Only at the end of this process was it possible to perform system testing to verify that the fundamental architecture was sound. Here the testing activities consume 40% or more life-cycle resources.

ACTIVITY COST
- Management 5%
- Requirements 5%
- Design 10%
- Code and unit testing 30%
- Integration and test 40%
- Deployment 5%
- Environment 5%

Symptoms of conventional waterfall process:
- Early paper designs and thorough briefings
- Commitment to code very late in cycle
- Integration nightmares due to unforeseen implementation and interface issues
- Heavy budget and schedule pressure
- Late ‘shoe-horning’ of non-optimal fixes with no time for redesign!!!
- A very fragile, un-maintainable product and almost always delivered late

Three Marks Questions With Answers
1. What is meant by ROI?

**Answer:**

A performance measure used to evaluate the efficiency of an investment or to compare the Efficiency of a number of different investments. To calculate ROI, the benefit (return) of an investment is divided by the cost of the investment; the result is expressed as a percentage or a ratio. The return on investment formula: Return on investment is a very popular metric because of its versatility and simplicity. That is, if an investment does not have a positive ROI, or if there are other opportunities with a higher ROI, then the investment should be not be undertaken.

2. Give a brief note on 2nd generation of software economics?

**Answer:**

Transition: 1980 and 1990, software engineering. Organizations used more-repeatable processes and off-the shelf tools, and mostly (>70%) custom components built in higher level languages. Some of the components (<30 %) were available as commercial products like OS, DBMS.

3. What are the attributes that constitutes a good estimate?

**Answer:**

A good estimate has the following attributes:

- 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.

4. Explain the waterfall model? discuss five necessary improvements for this approach?
There are five improvements to the basic waterfall model that would eliminate most of the development risks are as follows:

a) Complete program design before analysis and coding begin (program design comes first):
   - By this technique, the program designer gives surety that the software will not fail because of storage, timing, and data fluctuations.
   - Begin the design process with program designer, not the analyst or programmers.
   - Write an overview document that is understandable, informative, and current so that every worker on the project can gain an elemental understanding of the system.

b) Maintain current and complete documentation (Document the design):
   - It is necessary to provide a lot of documentation on most software programs.
   - Due to this, helps to support later modifications by a separate test team, a separate maintenance team, and operations personnel who are not software literate.

c) Do the job twice, if possible (Do it twice):
   - If a computer program is developed for the first time, arrange matters so that the version finally delivered to the customer for operational deployment is actually the second version insofar as critical design/operations are concerned.
   - “Do it N times” approach is the principle of modern-day iterative development.

d) Plan, control, and monitor testing:
Software Project Management

-The biggest user of project resources is the test phase. This is the phase of greatest risk in terms of cost and schedule.

-In order to carryout proper testing the following things to be done:
  - Employ a team of test specialists who were not responsible for the original design.
  - Employ visual inspections to spot the obvious errors like dropped minus signs, missing factors of two, jumps to wrong addresses.
    - Test every logic phase.
    - Employ the final checkout on the target computer.

e) Involve the customer:
  - It is important to involve the customer in a formal way so that he has committed himself at earlier points before final delivery by conducting some reviews such as,
    - Preliminary software review during preliminary program design step.
    - Critical software review during program design.
    - Final software acceptance review following testing.

5. What are the top 10 principles of conventional software management performance?

Answer:

Conventional Software Management Performance

Barry Boehm’s Top 10 “Industrial Software Metrics”:

a) Finding and fixing a software problem after delivery costs 100 times more than finding and fixing the problem in early design phases.

b) You can compress software development schedules 25% of nominal (small), but no more.

c) For every $1 you spend on development, you will spend $2 on maintenance.

d) Software development and maintenance costs are primarily a function of the number of source lines of code.

e) Variations among people account for the biggest difference in software productivity.

f) The overall ratio of software to hardware costs is still growing. In 1955 it was 15:85; in 1985, 85:15.

g) Only about 15% of software development effort is devoted to programming.
h) Software systems and products typically cost 3 times as much per SLOC as individual software programs. Software-system products cost 9 times as much.

i) Walkthroughs catch 60% of the errors.

j) 80% of the contribution comes from 20% of the contributors.
   -80% of the engineering is consumed by 20% of the requirements.
   -80% of the software cost is consumed by 20% of the components.
   -80% of the errors are caused by 20% of the components.
   -80% of the software scrap and rework is caused by 20% of the errors.
   -80% of the resources are consumed by 20% of the components.
   -80% of the engineering is accomplished by 20% of the tools.
   -80% of the progress is made by 20% of the people.

6. List the symptoms of the project which are in trouble frequently and explain late risk resolution?

   **Answer:**

   Projects destined for trouble frequently exhibit the following symptoms
   - Protracted integration and late design breakage
   - Late risk resolution
   - Requirements – driven functional decomposition
   - Adversarial stakeholder relationships
   - Focus on documents and review meetings

   **Late Risk Resolution**

   A serious issues associated with the waterfall life cycle was the lack of early risk resolution. The risk profile of a waterfall model is, It includes four distinct periods of risk exposure, where risk is defined as “the probability of missing a cost, schedule, feature, or quality goal”.

   K.Ramakrishna, Assistant Professor 117
Five Marks Questions With Answers

1). Explain in detail about adversarial stakeholder relationships?

Answer:
The conventional process tended to result in adversarial stakeholder relationships, in large part because of the difficulties of requirements specification and the exchange of information solely through paper documents that captured engineering information in ad hoc formats. The lack of rigorous notation resulted mostly in subjective reviews and opinionated exchanges of information.
2. Explain focus on documents and review meetings?

Answer:

Focus on Documents and Review Meetings: The conventional process focused on producing various documents that attempted to describe the software product, with insufficient focus on producing tangible increments of the products themselves. Major milestones were usually implemented as meetings defined solely in terms of specific documents. Contractors were driven to produce literally tons of paper to meet milestones and demonstrate progress to stakeholders, rather than spend their energy on tasks that would reduce risk and produce quality software. Typically, presenters and the audience reviewed the simple things that they understood rather than the complex and important issues. Most design reviews therefore resulted in low engineering value and high cost in terms of the effort and schedule involved in their preparation and conduct. They presented merely a facade of progress. Table 1-2 summarizes the results of a typical design review.
3. Explain in detail about requirement-driven functional decomposition?

Answer:

This approach depends on specifying requirements completely and unambiguously before other development activities begin. It naively treats all requirements as equally important, and depends on those requirements remaining constant over the software development life cycle. These conditions rarely occur in the real world. Specification of requirements is a difficult and important part of the software development process. As discussed in Appendix A, virtually every major software program suffers from severe difficulties in requirements specification. Moreover, the equal treatment of all requirements drains away substantial numbers of engineering hours from the driving requirements and wastes those efforts on paperwork associated with

<table>
<thead>
<tr>
<th>Results of conventional software</th>
<th>Project design reviews</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apparent results</td>
<td>Real results</td>
</tr>
<tr>
<td>Big briefing to a diverse audience</td>
<td>Only a small percentage of the audience understands the software. Briefings and documents expose few of the important assets and risks of complex software systems.</td>
</tr>
<tr>
<td>A design that appears to be compliant</td>
<td>There is no tangible evidence of compliance. Compliance with ambiguous requirements is of little value.</td>
</tr>
<tr>
<td>Coverage of requirements (typically Hundreds)</td>
<td>Few (tens) are design drivers. Dealing with all requirements dilutes the focus on the critical drivers.</td>
</tr>
<tr>
<td>A design considered &quot;innocent until Proven guilty&quot;</td>
<td>The design is always guilty. Design flaws are exposed later in the life cycle.</td>
</tr>
</tbody>
</table>
traceability, testability, logistics support, and so on—paperwork that is inevitably discarded later as the driving requirements and subsequent design understanding evolve.

As an example, consider a large-scale project such as CCPDS-R, presented as a case study in Appendix D, where the software requirements included 2,000 shalls. (A shall is a discrete requirement such as "the system shall tolerate all single-point hardware failures with no loss of critical capabilities.") Dealing adequately with the design drivers in such systems (typically only 20 to 50 of the shalls) is difficult when the contractual standards require that all 2,000 shalls be defined first and dealt with at every major milestone. The level of engineering effort that can be expended on the important design issues is significantly diluted by carrying around the excess baggage of more than 1,950 shalls and dealing with traceability, testability, documentation, and so on.

This decomposition was often very different from a decomposition based on object-oriented design and the use of existing components. The functional decomposition also became anchored in contracts, subcontracts, and work breakdown structures, often precluding a more architecture-driven approach. Figure 1-4 illustrates the result of requirements-driven approaches: a software structure that is organized around the requirements specification structure.

4. Explain pragmatic software cost estimation?

Answer:

One critical problem in software cost estimation is a lack of well-documented case studies of projects that used an iterative development approach. Although cost model vendors claim that their tools are suitable for estimating iterative development projects, few are based on empirical project databases with modern iterative development success stories. Furthermore, because the software industry has inconsistently defined metrics or atomic units of measure, the data from actual projects are highly suspect in terms of consistency and comparability. It is hard enough to collect a homogeneous set of project data within one organization; it is extremely difficult to homogenize data across different organizations with different processes, languages, domains, and so on. For example, the fundamental unit of size (a source line of code or a function point) can be, and is, counted differently across the industry. It is surprising that modern language standards (such as Ada 95 and Java) don't make a simple definition of a source line reportable by
the compiler. The exact definition of a function point or a SLOC is not very important, just as the exact length of a foot or a meter is equally arbitrary. It is simply important that everyone uses the same definition.

There have been many long-standing debates among developers and vendors of software cost estimation models and tools. Three topics of these debates are of particular interest here:

- Which cost estimation model to use
- Whether to measure software size in source lines of code or function points
- What constitutes a good estimate

The predominant cost estimation process resistance from other stakeholders who are expecting to synchronize. So plans need to be as ambitious as can possibly be achieved.

Most real-world use of cost models is bottom-up (substantiating a target cost) rather than top-down (estimating the "should" cost). Illustrates the predominant practice: The software project manager defines the target cost of the software, then manipulates the parameters and sizing until the target cost can be justified. The rationale for the target cost may be to win a proposal, to solicit customer funding, to attain internal corporate funding, or to achieve some other goal.
5. Explain the evolution of software economics?

Answer:

SOFTWARE ECONOMICS:
The cost of the software can be estimated by considering the following things as parameters to a function.

1. Size: Which is measured in terms of the number of Source Lines of Code or the number of function points required to develop the required functionality?

2. Process: Used to produce the end product, in particular the ability of the process is to avoid non value adding activities (rework, bureaucratic delays, and communications overhead).

3. Personnel: The capabilities of software engineering personnel, and particularly their experience with the computer science issues and the application domain issues of the
4) Environment: Which is made up of the tools and techniques available to support efficient software development and to automate the process.

4. Quality: It includes its features, performance, reliability, and flexibility. The relationship among these parameters and estimated cost can be calculated by using,

\[
\text{Effort} = (\text{Personnel}) \times (\text{Environment}) \times (\text{Quality}) \times (\text{Size Process})
\]

One important aspect of software economics is that the relationship between effort and size exhibits a diseconomy of scale and is the result of the process exponent being greater than 1.0. Converse to most manufacturing processes, the more software you build, the more expensive it is per unit item.
OBJECTIVE TYPE QUESTIONS WITH ANSWERS

9. Two essential steps common to the development of computer programs are _______.

10. Name any symptom that a project frequently suffers with _________________.

11. Conventional software economics provides a benchmark of performance for _______.

12. Finding and fixing a software problem after delivery costs _________ more than finding and fixing the problem in early design phases.

13. Five basic parameters of software cost models are_____________________.

14. Period of first generation of software economics is _________________.

15. Name the third generation of software economics is _____________________.

16. Acronym of ROI_______________.

17. First stage of large scale system approach is _____________________.

18. Only about 15% of software development effort is devoted to _________________.

ANSWERS

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>analysis and coding</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>late risk resolution</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>conventional software management principles.</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>100 times</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>size, process, personnel, environment, and quality.</td>
<td>10</td>
</tr>
</tbody>
</table>
MULTIPLE CHOICE QUESTIONS

11. Which of the phase will come under waterfall model?
   a) System requirements  
   b) Deployment  
   c) Testing  
   d) All the above

12. Symptoms of project in trouble?
   a) Late risk resolution  
   b) Testing  
   c) Coding  
   d) Analysis

13. Walkthrough catch --- of the errors?
   a) 50%  
   b) 20%  
   c) 60%  
   d) 10%

14. 80% of the contribution comes from 20% of ---- ?
   a) Testers  
   b) Developers  
   c) Contributors  
   d) Analyzers

15. The period of transition generation in software economics?
   a) 1960s-1970s  
   b) 1980s-1990s  
   c) 2000 and on  
   d) 1950s-1960s

16. What is the environment used in modern practices?
   a) Custom  
   b) Separate
c) Off-the-shelf, Integrated

d) Off-the-shelf

17. Acronym of SLOC

a) Source lines of code
b) Source line of count
c) Source line on count
d) Simple line of code

18. Which is not the parameter of software cost models?

a) Size
b) Environment
c) Quality
d) None of the above

19. The process used in conventional model

a) Repeatable
b) Managed
c) Measured
d) Ad hoc

20. The relationship among the parameters of cost estimation models.

a) Effort = (Personnel) (Environment) (Quality) (Size_{Process})
b) Effort = (Environment) (Quality) (Size_{Process})
c) Effort = (Personnel) (Quality) (Size_{Process})
d) Effort = (Personnel) (Environment) (Size_{Process})

**ANSWERS**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>d</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>a</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>c</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>c</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>a</td>
<td>10</td>
</tr>
</tbody>
</table>
Two Marks Questions With Answers

1). List the five basic parameters of improving software economics?
Ans: -Reducing the size or complexity of what needs to be developed
-Improving the development process
-Using more-skilled personnel and better teams
-Using better environments
-Trading off or backing off on quality thresholds

2). List the five characteristics of a successful object-oriented project?
Ans-A ruthless focus on the development of a system that provides a well-understood collection of essential minimal characteristics.
-The existence of a culture that is centered on results, encourages communication and yet is not afraid at all.
-The effective use of object-oriented modeling
-The existence of a strong architectural vision.
-The application of a well managed iterative and incremental development life cycle.

3). Define meta-process and macro-process?
Ans -Meta process: an organizations policies, procedures and practices for pursuing a software intensive line of business. The focus of this process is on organizational economics, long-term strategies and software ROI.
-Macro process: a projects policies, procedures and practices for producing a complete software product within certain cost, schedule and quality constraints.

4). List the five staffing principles proposed by BHOEM?
Ans -The principle of top-talent: use better and fewer people.
-The principle of job matching: fit the tasks to the skills and motivation of the people available.
-The principle of career progression: an organization does best in the long run by helping its people to self actualize.
- The principle of team balance: select people who will complement and harmonize with each other.

- The principle of phase-out: keeping a misfit on the team doesn’t benefit anyone.

5). Justify the principle “analyze the cause for errors”.
Ans: This is far more cost-effective to reduce the effect of an error by preventing it than it is find and fix it. One way to do this is to analyze the causes of errors as they are detected.

**Three marks questions with answers**

1. Illustrate the top 5 principles of modern software management in a diagrammatical manner?
Ans:

![Diagram showing Waterfall and Iterative Process]

**Waterfall Process**
- Requirements first
- Custom development
- Change avoidance
- Ad hoc tools

**Iterative Process**
- Architecture first
- Component-based development
- Change management
- Round-trip engineering

**Architecture-first approach**
- Design and integration first, then production and test

**Iterative life-cycle process**
- Risk control through ever-increasing function, performance, quality

**Component-based development**
- Object-oriented methods, rigorous notations, visual modeling

**Change management environment**
- The control element
- Metrics, trends, process instrumentation

**Round-trip engineering**
- The automation element
- Complementary tools, integrated environments
2. Give the brief note on improving software economics in diagrammatical manner?

**Ans:**

<table>
<thead>
<tr>
<th>COST MODEL PARAMETERS</th>
<th>TRENDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size Abstraction and component-based development technologies</td>
<td>Higher order languages (C++, Ada 95, Java, Visual Basic, etc.) Object-oriented (analysis, design, programming) Reuse Commercial components</td>
</tr>
<tr>
<td>Process Methods and techniques</td>
<td>Iterative development Process maturity models Architecture-first development Acquisition reform</td>
</tr>
<tr>
<td>Personnel People factors</td>
<td>Training and personnel skill development Teamwork Win-win cultures</td>
</tr>
<tr>
<td>Environment Automation technologies and tools</td>
<td>Integrated tools (visual modeling, compiler, editor, debugger, change management, etc.) Open systems Hardware platform performance Automation of coding, documents, testing, analyses</td>
</tr>
<tr>
<td>Quality Performance, reliability, accuracy</td>
<td>Hardware platform performance Demonstration-based assessment Statistical quality control</td>
</tr>
</tbody>
</table>

3. What are the characteristics of reusable components?

**Ans** - They have an economic motivation for continued support.
- They take ownership of improving product quality, adding new features and transitioning to new technologies.
- They have a sufficiently broad customer base to be profitable.
4. Define roundtrip engineering, forward engineering and reverse engineering?

**Ans:**
- Round-trip engineering: is a term used to describe the key capability of environments that support iterative development.

- Forward engineering: is the automation of one engineering artifact from another, more abstract representation. Ex: compilers and linkers

- Reverse engineering: is the generation of modification of more abstract representation from an existing artifact. Ex: creating visual design model from a source code.

5. Summarize the advantages and disadvantages of commercial components vs custom software?

**Ans:**

<table>
<thead>
<tr>
<th>APPROACH</th>
<th>ADVANTAGES</th>
<th>DISADVANTAGES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commercial</td>
<td>Predictable license costs</td>
<td>Frequent upgrades</td>
</tr>
<tr>
<td>components</td>
<td>Broadly used, mature technology</td>
<td>Up-front license fees</td>
</tr>
<tr>
<td></td>
<td>Available now</td>
<td>Recurring maintenance fees</td>
</tr>
<tr>
<td></td>
<td>Dedicated support organization</td>
<td>Dependency on vendor</td>
</tr>
<tr>
<td></td>
<td>Hardware/software independence</td>
<td>Run-time efficiency sacrifices</td>
</tr>
<tr>
<td></td>
<td>Rich in functionality</td>
<td>Functionality constraints</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Integration not always trivial</td>
</tr>
<tr>
<td></td>
<td></td>
<td>No control over upgrades and maintenance</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Unnecessary features that consume extra resources</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Often inadequate reliability and stability</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Multiple-vendor incompatibilities</td>
</tr>
<tr>
<td>Custom</td>
<td>Complete change freedom</td>
<td>Expensive, unpredictable development</td>
</tr>
<tr>
<td>development</td>
<td>Small, often simplest implementations</td>
<td>Unpredictable availability date</td>
</tr>
<tr>
<td></td>
<td>Often better performance</td>
<td>Undefined maintenance model</td>
</tr>
<tr>
<td></td>
<td>Control of development and enhancement</td>
<td>Often immature and fragile</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Single-platform dependency</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Drain on expert resources</td>
</tr>
</tbody>
</table>
Five marks Questions with Answer

1). Explain the top 15 principles of conventional software engineering?

Ans: Make quality #1: Quality must be quantified and mechanisms put into place to motivate its achievement.

High-quality software is possible: In order to improve the quality of the product we need to involving the customer, select the prototyping, simplifying design, conducting inspections, and hiring the best people.

Give products to customers early: No matter how hard you try to learn user’s 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.

Determine the problem before writing the requirements: Whenever a problem is raised most engineers provide a solution. Before we try to solve a problem, be sure to explore all the alternatives and don’t be blinded by the understandable solution.

Evaluate design alternatives: After the requirements are greed upon, we must examine a variety of architectures and algorithms and choose the one which is not used earlier.

Use an appropriate process model: For every project, there are so many prototypes (process models). So select the best one that is exactly suitable to our project.

Use different languages for different phases: Our industry’s main aim is to provide simple solutions to complex problems. In order to accomplish this goal choose different languages for different modules/phases if required.

Minimize intellectual distance: We have to design the structure of a software is as close as possible to the real-world structure.

Put techniques before tools: An un disciplined software engineer with a tool becomes a dangerous, undisciplined software engineer.

Get it right before you make it faster: It is very easy to make a working program run faster than it is to make a fast program work. Don’t worry about optimization during initial coding.

Inspect the code: Examine the detailed design and code is a much better way to find the errors than testing.

Good management is more important than good technology

People are the key to success: Highly skilled people with appropriate experience, talent, and training are key. The right people with insufficient tools, languages, and process will succeed.
Follow with care: Everybody is doing something but does not make it right for you. It may be right, but you must carefully assess its applicability to your environment.

Take responsibility: When a bridge collapses we ask “what did the engineer do wrong?” Similarly if the software fails, we ask the same. So the fact is in every engineering discipline, the best methods can be used to produce poor results and the most out of date methods to produce stylish design.

2). Explain any 15 principles of conventional software engineering?

Ans: -Understand the customer’s priorities. It is possible the customer would tolerate 90% of the functionality delivered late if they could have 10% of it on time.

- The more they see, the more they need. The more functionality (or performance) you provide a user, the more functionality (or performance) the user wants.

- Plan to throw one away. One of the most important critical success factors is whether or not a product is entirely new. Such brand-new applications, architectures, interfaces, or algorithms rarely work the first time.

- Design for change. The architectures, components, and specification techniques you use must accommodate change.

- Design without documentation is not design. I have often heard software engineers say, “I have finished the design. All that is left is the documentation.”

- Use tools, but be realistic. Software tools make their users more efficient.

- Avoid tricks. Many programmers love to create programs with tricks - constructs that perform a function correctly, but in an obscure way. Show the world how smart you are by avoiding tricky code.

- Encapsulate. Information-hiding is a simple, proven concept that results in software that is easier to test and much easier to maintain.

- Use coupling and cohesion. Coupling and cohesion are the best ways to measure software’s inherent maintainability and adaptability.

- Use the McCabe complexity measure. Although there are many metrics available to report the inherent complexity of software, none is as intuitive and easy to use as Tom McCabe’s.
-Don’t test your own software. Software developers should never be the primary testers of their own software.

-Analyze causes for errors. It is far more cost-effective to reduce the effect of an error by preventing it than it is to find and fix it. One way to do this is to analyze the causes of errors as they are detected.

-Realize that software’s entropy increases. Any software system that undergoes continuous change will grow in complexity and become more and more disorganized.

-People and time are not interchangeable. Measuring a project solely by person-months makes little sense.

-Expert excellence. Your employees will do much better if you have high expectations for them.

3). Explain the top five principles of modern software management?
Ans : i)Base the process on an architecture-first approach: (Central design element)
   - Design and integration first, then production and test
ii)Establish an iterative life-cycle process: (The risk management element)
   - Risk control through ever-increasing function, performance, quality.

With today’s sophisticated systems, it is not possible to define the entire problem, design the entire solution, build the software, then test the end product in sequence. Instead, and iterative process that refines the problem understanding, an effective solution, and an effective plan over several iterations encourages balanced treatment of all stakeholder objectives.

Major risks must be addressed early to increase predictability and avoid expensive downstream scrap and rework.

1)Transition design methods to emphasize component-based development: (The technology element)

Moving from LOC mentally to component-based mentally 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.

2)Establish a change management environment: (The control element)
   - Metrics, trends, process instrumentation
The dynamics of iterative development, include concurrent workflows by different teams working on shared artifacts, necessitates objectively controlled baseline. 3) Enhance change freedom through tools that support round-trip engineering: (The automation element)
- Complementary tools, integrated environment
Round-trip engineering is the environment support necessary to automate and synchronize engineering information in different formats. Change freedom is necessary in an iterative process.
4). Summarize the important trends in improving software economics?

**Ans:** Improving software economics:

- It is not that much easy to improve the software economics but also difficult to measure and validate.
- There are many aspects are there in order to improve the software economics they are, Size, Process, Personnel, Environment and quality.
- These parameters (aspects) are not independent they are dependent. For example, tools enable size reduction and process improvements, size-reduction approaches lead to process changes, and process improvements drive tool requirements.
- GUI technology is a good example of tools enabling a new and different process. GUI builder tools permitted engineering teams to construct an executable user interface faster and less cost.
- Two decades ago, teams developing a user interface would spend extensive time analyzing factors, screen layout, and screen dynamics. All this would done on paper. Where as by using GUI, the paper descriptions are not necessary.
- Along with these five basic parameters another important factor that has influenced software technology improvements across the board is the ever-increasing advances in hardware Performance.

<table>
<thead>
<tr>
<th>COST MODEL PARAMETERS</th>
<th>TRENDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>Higher order languages (C++, Ada 95, Java, Visual Basic, etc.)</td>
</tr>
<tr>
<td>Abstraction and component-based development technologies</td>
<td>Object-oriented (analysis, design, programming)</td>
</tr>
<tr>
<td></td>
<td>Reuse</td>
</tr>
<tr>
<td></td>
<td>Commercial components</td>
</tr>
<tr>
<td>Process</td>
<td>Iterative development</td>
</tr>
<tr>
<td>Methods and techniques</td>
<td>Process maturity models</td>
</tr>
<tr>
<td></td>
<td>Architecture-first development</td>
</tr>
<tr>
<td></td>
<td>Acquisition reform</td>
</tr>
<tr>
<td>Personnel</td>
<td>Training and personnel skill development</td>
</tr>
<tr>
<td>People factors</td>
<td>Teamwork</td>
</tr>
<tr>
<td></td>
<td>Win-win cultures</td>
</tr>
<tr>
<td>Environment</td>
<td>Integrated tools (visual modeling, compiler, editor, debugger, change management, etc.)</td>
</tr>
<tr>
<td>Automation technologies and tools</td>
<td>Open systems</td>
</tr>
<tr>
<td></td>
<td>Hardware platform performance</td>
</tr>
<tr>
<td></td>
<td>Automation of coding, documents, testing, analyses</td>
</tr>
<tr>
<td>Quality</td>
<td>Hardware platform performance</td>
</tr>
<tr>
<td>Performance, reliability, accuracy</td>
<td>Demonstration-based assessment</td>
</tr>
<tr>
<td></td>
<td>Statistical quality control</td>
</tr>
</tbody>
</table>
5). Explain the “achieving required quality”?

Ans: Achieving required quality: Key elements that improve overall software quality include the following:

- Focusing on powerful requirements and critical use case early in the life
- Focusing on requirements completeness and traceability late in the life cycle
- Focusing throughout the life cycle on a balance between requirements evolution, design evolution, and plan evolution
- Using metrics and indicators to measure the progress and quality of an architecture as it evolves from high-level prototype into a fully biddable product
- Providing integrated life-cycle environments that support early and continuous configuration control, change management, rigorous design methods, document automation, and regression test automation
- Using visual modeling and higher level languages that support architectural control, abstraction, reliable programming, reuse, and self-documentation
  - Early and continuous close look into performance issues through demonstration-based evaluations

**In order to evaluate the performance the following sequence of events are necessary,**

- Project inception
- Initial design review
- Mid-life-cycle design review
- Integration and test
6). Explain the peer inspections?

**Ans**: Peer inspections are frequently overhyped as the key aspect of a quality system. In my experience, peer reviews are valuable as secondary mechanisms, but they are rarely significant contributors to quality compared with the following primary quality mechanisms and indicators, which should be emphasized in the management process:

- Transitioning engineering information from one artifact set to another, thereby assessing the consistency, feasibility, understandability, and technology constraints inherent in the engineering artifacts
- Major milestone demonstrations that force the artifacts to be assessed against tangible criteria in the context of relevant use cases
- Environment tools (compilers, debuggers, analyzers, automated test suites) that ensure representation rigor, consistency, completeness, and change control
- Life-cycle testing for detailed insight into critical trade-offs, acceptance criteria, and requirements compliance
Software Project Management

- Change management metrics for objective insight into multiple-perspective change trends and convergence or divergence from quality and progress goals
- Although I believe that inspections are overemphasized, in certain cases they provide a significant return. One value of inspections is in the professional development of a team. It is generally useful to have the products of junior team members reviewed by senior mentors. Putting the products of amateurs into the hands of experts and vice versa is a good mechanism for accelerating the acquisition of knowledge and skill in new personnel. Gross blunders can be caught and feedback can be appropriately channeled, so that bad practices are not perpetuated. This is one of the best ways for junior software engineers to learn.
- Inspections are also a good vehicle for holding authors accountable for quality products. All authors of software and documentation should have their products scrutinized as a natural by-product of the process. Therefore, the coverage of inspections should be across all authors rather than across all components. Junior authors need to have a random component inspected periodically, and they can learn by inspecting the products of senior authors. Varying levels of informal inspection are performed continuously when developers are reading or integrating software with another author's software, and during testing by independent test teams. However, this "inspection" is much more tangibly focused on integrated and executable aspects of the overall system.
- Significant or substantial design errors or architecture issues are rarely obvious from a superficial review unless the inspection is narrowly focused on a particular issue. And most inspections are superficial. Today's systems are highly complex, with innumerable components, concurrent execution, distributed resources, and other equally demanding dimensions of complexity. It would take human intellects similar to those of world-class chess players to comprehend the dynamic interactions within some simple software systems under some simple use cases. Consequently, random human inspections tend to degenerate into comments on style and first-order semantic issues. They rarely result in the discovery of real performance bottlenecks, serious control issues (such as deadlocks, races, or resource contention), or architectural weaknesses (such as flaws in scalability, reliability, or interoperability). In all but
trivial cases, architectural issues are exposed only through more rigorous engineering activities such as the following:

- Analysis, prototyping, or experimentation
- Constructing design models
- Committing the current state of the design model to an executable implementation
- Demonstrating the current implementation strengths and weaknesses in the context of critical subsets of the use cases and scenarios
- Incorporating lessons learned back into the models, use cases, implementations, and plans

Achieving architectural quality is inherent in an iterative process that evolves the artifact sets together in balance. The checkpoints along the way are numerous, including human review and inspections focused on critical issues. But these inspections are not the primary checkpoints. Early life-cycle artifacts are certainly more dependent on subjective human review than later ones are. Focusing a large percentage of a project's resources on human inspections is bad practice and only perpetuates the existence of low-value-added box checkers who have little impact on project success. Look at any successful software effort and ask the key designers, testers, or developers about the discriminators of their success. It is unlikely that any of them will cite meetings, inspections, or documents.

Quality assurance is everyone's responsibility and should be integral to almost all process activities instead of a separate discipline performed by quality assurance specialists. Evaluating and assessing the quality of the evolving engineering baselines should be the job of an engineering team that is independent of the architecture and development team.
OBJECTIVE TYPE QUESTIONS

1. Modern software processes are in nature__________________.
2. The principle of top talent:_____________________________
3. _________________________is a overloaded term.
4. The subject of micro process is _________________
5. The time scales of meta process is _________________
6. One advantage of commercial components is _________________
7. One disadvantage of custom development is _________________
8. Acronym of UFPs is _________________
9. Trend of personnel parameter is _________________
10. Which technique is applied to ensure the continued evolution of legacy systems ______

ANSWERS

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>iterative</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>use better and fewer people.</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>Process</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>iteration.</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>6 to 12 months.</td>
<td>10</td>
</tr>
</tbody>
</table>
MULTIPLE CHOICE QUESTIONS

1. Which of the following will come under trends of quality?
   a) Teamwork
   b) Win-win cultures
   c) Statistical quality control
   d) Commercial components

2. Reducing the ----- of what needs to be developed?
   a) Size
   b) Process
   c) Personnel          d) Environments

3. Which of the following will come under process perspectives?
   a) Meta process
   b) Micro process
   c) Macro process
   d) All the above

4. Using more-skilled ---- and better teams?
   a) Size
   b) Process
   c) Personnel
   d) Environments

5. Which of the following will come under trends of personnel?
   a) Team work
   a) Open systems
   b) Hardware platform performance
   c) Quality

6. Time scale of macro process?
   a) 6 months
   b) 1 month
   c) 1 to many years      d) 12 months

K.Ramakrishna, Assistant Professor 142
7. Which will come under the quality driver?
   a) Automation
   b) Testing
   c) Coding
   d) Analysis

8. Principles of modern software management are?
   a) Architecture first approach
   b) Iterative life-cycle process
   c) Component based development
   d) All the above

9. Which is the staffing principle proposed by boehm?
   a) Principle of top talent
   b) Principle of talent
   c) Principle of job
   d) Principle of matching

10. Which of the following are the skills of project manager?
    a) Hiring skills
    b) Customer-interface skills
    c) Selling skill
    d) All the above

**ANSWERS**

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>c</td>
<td>6</td>
<td>c</td>
</tr>
<tr>
<td>2</td>
<td>a</td>
<td>7</td>
<td>a</td>
</tr>
<tr>
<td>3</td>
<td>d</td>
<td>8</td>
<td>d</td>
</tr>
<tr>
<td>4</td>
<td>c</td>
<td>9</td>
<td>a</td>
</tr>
<tr>
<td>5</td>
<td>a</td>
<td>10</td>
<td>d</td>
</tr>
</tbody>
</table>
Two marks Questions with Answers

(1). What are the primary objectives of construction phase?
Ans - 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.

(2). What are the essential activities of transition phase?
Ans - Synchronization and integration of concurrent construction increments into consistent deployment baselines.
   - Deployment-specific engineering
   - Assessment of deployment baselines against the complete vision and acceptance criteria in the requirements set.

(3). Summarize the artifact sets in a diagrammatical manner?
Ans:

```
<table>
<thead>
<tr>
<th>Requirements Set</th>
<th>Design Set</th>
<th>Implementation Set</th>
<th>Deployment Set</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Planning Artifacts</th>
<th>Management Set</th>
<th>Operational Artifacts</th>
</tr>
</thead>
</table>
```
(4). Explain the requirement set?

**Ans**: The requirements set is the primary engineering context for evaluating the other three engineering artifact sets and is the basis for test cases. Requirement artifacts are evaluated, assessed, and measured through a combination of:

- Analysis of consistency with the release specifications of the mgmt set.
- Analysis of consistency between the vision and the requirement models.
- Mapping against the design, implementation, and deployment sets to evaluate the consistency and completeness and the semantic balance between information in the different sets.
- Analysis of changes between the current version of the artifacts and previous versions.
- Subjective review of other dimensions of quality

(5). Explain deployment set?

**Ans**: It includes user deliverables and machine language notations, executable software, and the build scripts, installation scripts, and executable target-specific data necessary to use the product in its target environment. Deployment sets are evaluated, assessed, and measured through a combination of:

- Testing against the usage scenarios and quality attributes defined in the requirements set to evaluate the consistency and completeness and the semantic balance between information in the two sets.
- Testing the partitioning, replication, and allocation strategies in mapping components of the implementation set to physical resources of the deployment system.
- Testing against the defined usage scenarios in the user manual such as installation, user-oriented dynamic reconfiguration, mainstream usage, and anomaly management.
- Analysis of changes b/w the current version of the deployment set and previous versions.
- Subjective review of other dimensions of quality

**Three marks questions with answers**

1. Give a brief note on vision document?

**Ans**: 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.

(2) Give a brief note on architecture description?

**Ans:** The architecture description provides an organized view of the software architecture under development. It is extracted largely from the design model and includes views of the design, implementation, and deployment sets sufficient to understand how the operational concept of the requirements set will be achieved. The breadth of the architecture description will vary from project to project depending on many factors. The architecture can be described using a subset of the design model or as an abstraction of the design model with supplementary material, or a combination of both. As examples of these two forms of descriptions, consider the architecture of this book:

- A subset form could be satisfied by the table of contents. This description of the architecture of the book is directly derivable from the book itself.
- An abstraction form could be satisfied by a "Cliffs Notes" treatment. (Cliffs Notes are condensed versions of classic books used as study guides by some college students.) This format is an abstraction that is developed separately and includes supplementary material that is not directly derivable from the evolving product.

3. Give a brief note on software user manual?

**Ans:** The software user manual provides the user with the reference documentation necessary to support the delivered software. Although content is highly variable across application domains, the user manual should include installation procedures, usage procedures and guidance, operational constraints, and a user interface description, at a minimum. For software products with a user interface, this manual should be developed early in the life cycle because it is a necessary mechanism for communicating and stabilizing an important subset of requirements. The user manual should be written by members of the test team, who are more likely to
understand the user's perspective than the development team. If the test team is responsible for
the manual, it can be generated in parallel with development and can be evolved early as a
tangible and reliable.

4. List the issues in pragmatic artifacts?

Ans: - People want to review information but don't understand the language of the artifact.
   - People want to review information but don't have access to the tools.
   - Human-readable engineering artifacts should use rigorous notations that are complete,
     consistent, and used in a self-documenting manner.
   - Useful documentation is self-defining:
     - Paper is tangible; electronic artifacts are too easy to change.

5. What is artifact set?

Ans: In order to manage the development of a complete software system, we need to gather
distinct collections of information and is organized into artifact sets. Set represents a complete
aspect of the system where as artifact represents interrelated information that is developed and
reviewed as a single entity. The artifacts of the process are organized into five sets:
   - Management
   - Requirements
   - Design
   - Implementation
   - Deployment

Five marks questions with answers

1. Explain the phases of life cycle process?

Ans: Engineering and Production stages: To achieve economies of scale and higher return on
investment, we must move toward a software manufacturing process which is determined by
technological improvements in process automation and component based development.

There are two stages in the software development process

The engineering stage: Less predictable but smaller teams doing design and production
activities. This stage is decomposed into two distinct phases: inception and elaboration.
The production stage: More predictable but larger teams doing construction, test, and deployment activities. This stage is also decomposed into two distinct phases’ construction and transition.

These four phases of lifecycle process are loosely mapped to the conceptual framework of the spiral model is as shown in the following figure.

In the above figure the size of the spiral corresponds to the inactivity of the project with respect to the breadth and depth of the artifacts that have been developed.

- This inertia manifests itself in maintaining artifact consistency, regression testing, documentation, quality analyses, and configuration control.
- Increased inertia may have little, or at least very straightforward, impact on changing any given discrete component or activity.
- However, the reaction time for accommodating major architectural changes, major requirements changes, major planning shifts, or major organizational perturbations clearly increases in subsequent phases.

2. Explain in detail about inception phase?

Ans: Inception Phase: The main goal of this phase is to achieve agreement among stakeholders on the life-cycle objectives for the project.
The overriding goal of the inception phase is to achieve concurrence among all stakeholders on the lifecycle objectives for the project. The primary objectives of the inception phase include:

- Establishing the project's software scope and boundary conditions, including an operational concept, acceptance criteria and what is intended to be in the product and what is not.
- Discriminating the critical use cases of the system, the primary scenarios of operation that will drive the major design trade-offs.
- Exhibiting, and maybe demonstrating, at least one candidate architecture against some of the primary scenarios.
- Estimating the overall cost and schedule for the entire project (and more detailed estimates for the elaboration phase that will immediately follow)
- Estimating potential risks (the sources of unpredictability)

**The essential activities of the inception phase are:**

- Formulating the scope of the project. This involves capturing the context and the most important requirements and constraints to such an extent that you can derive acceptance criteria for the end product.
- Planning and preparing a business case. Evaluating alternatives for risk management, staffing, project plan, and cost/schedule/profitability trade-offs.
- Synthesizing a candidate architecture, evaluating trade-offs in design, and in make/buy/reuse, so that cost, schedule and resources can be estimated.

The outcome of the inception phase is:

- A vision document: a general vision of the core project's requirements, key features, main constraints.
- The use-case model survey (identifying all use cases that can be identified at this early stage).
- An initial glossary.
- An initial business case, which includes:
  - Business context.
  - Success criteria (revenue projection, market recognition, and so on).
Financial forecast.
- An initial risk assessment.
- A project plan, showing phases, and iterations.

3. Explain in detail about elaboration phase?

**Ans : ELABORATION PHASE:** It is the most critical phase among the four phases. Depending upon the scope, size, risk, and freshness of the project, an executable architecture prototype is built in one or more iterations.

The primary objectives of the elaboration phase include:

- Defining, validating and base lining the architecture as rapidly as practical
- Base lining the vision
- Base lining a high-fidelity plan for the construction phase
- Demonstrating that the baseline architecture will support this vision for reasonable cost in a reasonable time.

**The essential activities of the elaboration phase are:**

- Elaborating the vision establishing a solid understanding of the most critical use cases that drive the architectural and planning decisions.
- Elaborating the process and the infrastructure, the development environment. Putting in place the process, tools and automation support.
- Elaborating the architecture and selecting components. Potential components are evaluated and the make/buy/reuse decisions sufficiently understood to determine the construction phase cost and schedule with confidence. The selected architectural components are integrated and assessed against the primary scenarios. Lessons learned from these activities may well result in a redesign of the architecture, taking into consideration alternative designs or reconsideration of the requirements.

**The outcome of the elaboration phase is:**

- A use-case model (80% complete) - all use cases having been identified in the use-case model survey, all actors having been identified, and most use-case descriptions (requirements capture) having been developed.
Software Project Management

- Supplementary requirements capturing the non functional requirements and any requirements that are not associated with a specific use case.
- An executable architecture and accompanying documentation - the Software Architecture Document, including use-case descriptions (design) for a subset of use cases (use-case view), and an updated glossary.
- A revised business case.
- A revised risk list.
- A development plan for the overall project, including the coarse-grained project plan, showing iterations and evaluation criteria for each iteration.
- A preliminary user manual (optional).

4. Explain in detail about construction phase?

**Ans:** 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
Software Project Management

-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?

5. Explain in detail about transition phase?

Ans: The transition phase is entered when a baseline is mature enough to be deployed in the end-user domain. This typically requires that a usable subset of the system has been achieved with acceptable quality levels and user documentation so that transition to the user will provide positive results.

This phase could include any of the following activities:

1. Beta testing to validate the new system against user expectations
2. Beta testing and parallel operation relative to a legacy system it is replacing
3. Conversion of operational databases
4. Training of users and maintainers

The transition phase concludes when the deployment baseline has achieved the complete vision. For some projects, this life-cycle end point may coincide with the life-cycle starting point for the next version of the product. For others, it may coincide with a complete delivery of the information sets to a third party responsible for operation, maintenance, and enhancement.

The transition phase focuses on the activities required to place the software into the hands of the users. Typically, this phase includes several iterations, including beta releases, general availability releases, and bug-fix and enhancement releases. Considerable effort is expended in
developing user-oriented documentation, training users, supporting users in their initial product use, and reacting to user feedback. (At this point in the life cycle, user feedback should be confined mostly to product tuning, configuring, installing, and usability issues.)

**Primary Objectives**
- Achieving user self-supportability
- Achieving stakeholder concurrence that deployment baselines are complete and consistent with the evaluation criteria of the vision
- Achieving final product baselines as rapidly and cost-effectively as practical

**Essential Activities**
- Synchronization and integration of concurrent construction increments into consistent deployment baselines
- Deployment-specific engineering (cutover, commercial packaging and production, sales rollout kit development, field personnel training)
- Assessment of deployment baselines against the complete vision and acceptance criteria in the requirements set

**Evaluation Criteria**
- Are actual resource expenditures versus planned expenditures acceptable?

6. What are the different aspects of architecture in management perspective?

**Ans:** Architecture in management perspective: The most critical and technical product of a software project is its architecture

- If a software development team is to be successful, the interproject communications, as captured in software architecture, must be accurate and precise.

**From the management point of view, three different aspects of architecture**

1) An architecture (the intangible design concept) is the design of software system it includes all engineering necessary to specify a complete bill of materials. Significant make or buy decisions

2) are resolved, and all custom components are elaborated so that individual component costs and construction/assembly costs can be determined with confidence.
3) An architecture baseline (the tangible artifacts) is a slice of information across the engineering artifact sets sufficient to satisfy all stakeholders that the vision (function and quality) can be achieved within the parameters of the business case (cost, profit, time, technology, people).

6) An architectural description is an organized subset of information extracted from the design set model's. It explains how the intangible concept is realized in the tangible artifacts.

The number of views and level of detail in each view can vary widely. For example the architecture of the software architecture of a small development tool.

There is a close relationship between software architecture and the modern software development process because of the following reasons:

1) A stable software architecture is nothing but a project milestone where critical make/buy decisions should have been resolved. The life-cycle represents a transition from the engineering stage of a project to the production stage.

2) Architecture representation provide a basis for balancing the trade-offs between the problem space (requirements and constraints) and the solution space (the operational product).

3) The architecture and process encapsulate many of the important communications among individuals, teams, organizations, and stakeholders.

4) Poor architecture and immature process are often given as reasons for project failure.

5) In order to proper planning, a mature process, understanding the primary requirements and demonstrable architecture are important fundamentals.

6) Architecture development and process definition are the intellectual steps that map the problem to a solution without violating the constraints; they require human innovation and cannot be automated.
OBJECTIVE TYPE QUESTIONS

1. The two stages of life cycle are ______________
2. The engineering stage is decomposed into ______________ stages.
3. Output of the transition phase is ______________
4. The ______________ capture the information necessary to synchronize stakeholder expectations.
5. Name any two planning artifacts ______________
6. Requirement set consists of ______________
7. The different aspects of management perspective are ______________
8. ______________ is the software system design.
9. The goal of engineering stage is to converge on a ______________
10. The artifacts of the process are organized into ______________ sets.

ANSWERS

<table>
<thead>
<tr>
<th></th>
<th>engineering and production stage.</th>
<th></th>
<th>vision document, requirement models</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>inception and elaboration</td>
<td>6</td>
<td>architecture, architecture baseline, and architecture description.</td>
</tr>
<tr>
<td>2</td>
<td>products</td>
<td>7</td>
<td>Architecture</td>
</tr>
<tr>
<td>3</td>
<td>management artifacts</td>
<td>8</td>
<td>stable architecture baseline</td>
</tr>
<tr>
<td>5</td>
<td>WBS, software development plan</td>
<td>9</td>
<td>Five</td>
</tr>
</tbody>
</table>
MULTIPLE CHOICE QUESTIONS

1. The ------ stage driven by less predictable but smaller teams doing design and synthesis activities?
   a) Inception
   b) Elaboration
   c) Construction
   d) Engineering

2. The production stage comprises of
   a) Inception
   b) Elaboration
   c) Construction and transition
   d) Inception and elaboration

3. What is the essential activity of inception phase?
   a) Synthesizing the architecture
   b) Elaborating the vision
   c) Resource management
   d) Deployment specific engineering

4. ---- is a relatively independent abstraction of a system?
   a) Set
   b) Artifact
   c) Model
   d) Architecture

5. What is the essential activity of transition phase?
   a) Synthesizing the architecture
   b) Elaborating the vision
   c) Resource management
   d) Deployment specific engineering

6. Poor architectures and -------- are often given as reason for project failures
   a) Mature process
b) Immature process  
c) Baseline  
d) Architecture description  

7. What is the essential activity of elaboration phase?  
a) Synthesizing the architecture  
b) Elaborating the vision  
c) Resource management  
d) Deployment specific engineering  

8. The ultimate goal of the engineering stage is to cover a stable ----  
a) Descriptive baseline  
b) Architecture baseline  
c) Baseline  
d) Engineering baseline  

9. An architecture baseline is not a ----  
a) Document  
b) Soft copy  
c) Paper document  
d) Design  

10. Which of the following is the engineering artifacts?  
a) Architecture description  
b) Software user manual  
c) Vision document  
d) All the above  

ANSWERS
Two marks questions with answers

1. Define major milestones?
   **Ans:** These system wide events are held at the end of each development phase. They provide visibility to system wide issues, synchronize the management and engineering perspectives and verify that the aims of the phase have been achieved.

2. Define minor milestones?
   **Ans:** These iteration-focused events are conducted to review the content of iteration in detail and to authorize continued work. Minor milestones capture two artifacts: a release specification and a release description. 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.

3. What are the seven top-level workflows?
   **Ans:** The term workflow means a thread of cohesive and mostly sequential activities.
   --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.
Software Project Management

--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.

4. Define status assessments?

Ans: These periodic events provide management with frequent and regular insight into the progress being made. These are management reviews conducted at regular intervals (monthly, quarterly) to address progress and quality of project and maintain open communication among all stakeholders. The main objective of these assessments is to synchronize all stakeholders expectations and also serve as project snapshots.

5. Define WBS?

Ans: Work breakdown structure is the “architecture” of the project plan and also an architecture for financial plan. - A project is said to be in success, if we maintain good work breakdown structure and its synchronization with the process frame work. A WBS is simply a hierarchy of elements that decomposes the project plan into discrete work tasks and it provides:

--A pictorial description of all significant work.
--A clear task decomposition for assignment of responsibilities.
--A framework for scheduling, budgeting, and expenditure tracking.

Three Marks Questions With Answers

1. List the management indicators?

Ans: Management Indicators The management indicators i.e., technical progress, financial status and staffing progress are used to determine whether a project is on budget and on schedule. The management indicators that indicate financial status are based on earned value system.

→ Work and progress
→ Budgeted cost and expenditures
Software Project Management

→ Staffing and team dynamics

2. List the quality indicators?
   Ans: Quality Indicators The quality indicators are based on the measurement of the changes occurred in software.
   → change traffic and stability
   → Breakage and modularity
   → Rework and adaptability
   → Mean time between failures and maturity

3. What are the main features of default organization?
   Ans: → Responsibility for process definition & maintenance is specific to a cohesive line of business.
   → Responsibility for process automation is an organizational role & is equal in importance to the process definition role.
   → Organizational role may be fulfilled by a single individual or several different teams.

4. What is round trip engineering?
   Ans: Raising the environment activities to a first-class workflow is critical. The environment is the tangible picture of the project’s process, methods, and notations for producing the artifacts.

5. What is project review authority?
   Ans: Project Review Authority (PRA) The PRA is the single individual responsible for ensuring that a software project complies with all organizational & business unit software policies, practices & standards A software Project Manager is responsible for meeting the requirements of a contract or some other project compliance standard.

Five Marks Questions With Answers

1. Explain in detail about iteration workflows?
   Ans: 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. An
individual iteration's workflow, illustrated in Figure 8-2, 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 (previous versions)

- **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

As with any sequence of a software development workflow, many of the activities occur concurrently. For example, requirements analysis is not done all in one continuous lump; it intermingles with management, design, implementation, and so forth.

Iterations in the inception and elaboration phases focus on management, requirements, and design activities. Iterations in the construction phase focus on design, implementation, and assessment. Iterations in the transition phase focus on assessment and deployment. Figure 8-3 shows the emphasis on different activities across the life cycle.

These descriptions are pretty simplistic. In practice, the various sequences and overlaps among iterations become more complex. The terms iteration and increment deal with some of the pragmatic considerations. An iteration represents the state of the overall architecture and the complete deliverable system. An increment represents the current work in progress that will be combined with the preceding iteration to form the next iteration. Figure 8-4, an example of a simple development life cycle, illustrates the difference between iterations and increments. This example also illustrates a typical build sequence from the perspective of an abstract layered architecture.
Progress can be measured as the % of components under configuration control, the % of demonstrable use cases, etc.

Application-specific components
- Iteration 1: 6, 3
- Iteration 2: 1
- Iteration 3: 5
- Iteration 4: 4
- Increment 4: 5
- Iteration 5: 2
- Increment 5: 5
- Iteration 6: 6

Domain-specific components
- Iteration 1: 1
- Iteration 2: 6, 6
- Iteration 3: 4
- Increment 4: 3
- Iteration 5: 2
- Increment 5: 6
- Iteration 6: 3
- Increment 6: 3

Middleware and common mechanism components
- Iteration 1: 5
- Iteration 2: 3
- Iteration 3: 4
- Increment 4: 3
- Iteration 5: 2
- Increment 5: 1
- Iteration 6: 4

Operating system and networking components
- Iteration 1: 1
- Iteration 2: 2
- Iteration 3: 1
- Increment 4: 1
- Iteration 5: 3
- Increment 5: 3
- Iteration 6: 2
- Increment 6: 3

Iteration 7 adds no new components, only upgrades, fixes, and enhancements.

Iterations 1, 2, and 3 include architecturally significant components.
2. What are the management reviews that are conducted throughout the process?

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

- **Major milestones.** These system wide events are held at the end of each development phase. They provide visibility to system wide issues, synchronize the management and engineering perspectives, and verify that the aims of the phase have been achieved.

- **Minor milestones.** These iteration-focused events are conducted to review the content of an iteration in detail and to authorize continued work.

- **Status assessments.** These periodic events provide management with frequent and regular insight into the progress being made.

Figure illustrates a typical sequence of project checkpoints for a relatively large project.

![Project Checkpoints Diagram](image)

3. Explain the initial operational capability milestone and product release milestone in detail?

**Ans: 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. Acceptance testing can be done incrementally across multiple iterations or can be completed entirely during the transition phase is not necessarily the completion of the construction phase.

**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. Software quality metrics are reviewed to determine whether quality is sufficient for transition to the support organization.

4. Explain in detail about conventional WBS?

**Ans:** work breakdown structure

A WBS is simply a hierarchy of elements that decomposes the project plan into the discrete work tasks. A WBS provides the following information structure:

- A delineation of all significant work
- A clear task decomposition for assignment of responsibilities
- A framework for scheduling, budgeting, and expenditure tracking

Many parameters can drive the decomposition of work into discrete tasks: product subsystems, components, functions, organizational units, life-cycle phases, even geographies. Most systems have a first-level decomposition by subsystem. Subsystems are then decomposed into their components, one of which is typically the software.

**CONVENTIONAL WBS ISSUES**

Conventional work breakdown structures frequently suffer from three fundamental flaws.

- They are prematurely structured around the product design.
- They are prematurely decomposed, planned, and budgeted in either too much or too little detail.
- They are project-specific, and cross-project comparisons are usually difficult or impossible.

Conventional work breakdown structures are prematurely structured around the product design. Figure shows a typical conventional WBS that has been structured primarily around the subsystems of its product architecture, then further decomposed into the components of each subsystem. A WBS is the architecture for the financial plan.

Conventional work breakdown structures are prematurely decomposed, planned, and budgeted in either too little or too much detail. Large software projects tend to be over planned and small projects tend to be under planned. The basic problem with planning too
much detail at the outset is that the detail does not evolve with the level of fidelity in the plan.

Conventional work breakdown structures are project-specific, and cross-project comparisons are usually difficult or impossible. With no standard WBS structure, it is extremely difficult to compare plans, financial data, schedule data, organizational efficiencies, cost trends, productivity trends, or quality trends across multiple projects.

5. Explain in detail about evolutionary work breakdown structures?

**Ans: EVOLUTIONARY WORK BREAKDOWN STRUCTURES**

An evolutionary WBS should organize the planning elements around the process framework rather than the product framework. The basic recommendation for the WBS is to organize the hierarchy as follows:

```
Management
  System requirements and design
  Subsystem 1
    Component 11
      Requirements
      Design
      Code
      Test
      Documentation
      ... (similar structures for other components)
    Component 1N
      Requirements
      Design
      Code
      Test
      Documentation
      ...(similar structures for other subsystems)
  Subsystem M
    Component M1
      Requirements
      Design
      Code
      Test
      Documentation
      ...(similar structures for other components)
    Component MN
      Requirements
      Design
      Code
      Test
      Documentation
Integration and test
  Test planning
  Test procedure preparation
  Testing
  Test reports
Other support areas
  Configuration control
  Quality assurance
  System administration
```
First-level WBS elements are the workflows (management, environment, requirements, design, implementation, assessment, and deployment).

Second-level elements are defined for each phase of the life cycle (inception, elaboration, construction, and transition).

Third-level elements are defined for the focus of activities that produce the artifacts of each phase.

This recommended structure provides one example of how the elements of the process framework can be integrated into a plan. It provides a framework for estimating the costs and schedules of each element, allocating them across a project organization, and tracking expenditures.

The structure shown is intended to be merely a starting point. It needs to be tailored to the specifics of a project in many ways.

Scale. Larger projects will have more levels and substructures.

Organizational structure. Projects that include subcontractors or span multiple organizational entities may introduce constraints that necessitate different WBS allocations.

Degree of custom development. Depending on the character of the project, there can be very different emphases in the requirements, design, and implementation workflows.

Business context. Projects developing commercial products for delivery to a broad customer base may require much more elaborate substructures for the deployment element.

Precedent experience. Very few projects start with a clean slate. Most of them are developed as new generations of a legacy system (with a mature WBS) or in the context of existing organizational standards (with preordained WBS expectations).

The WBS decomposes the character of the project and maps it to the life cycle, the budget, and the personnel. Reviewing a WBS provides insight into the important attributes, priorities, and structure of the project plan.

Another important attribute of a good WBS is that the planning fidelity inherent in each element is commensurate with the current life-cycle phase and project state. One of the
primary reasons for organizing the default WBS the way I have is to allow for planning elements that range from planning packages (rough budgets that are maintained as an estimate for future elaboration rather than being decomposed into detail) through fully planned activity networks (with a well-defined budget and continuous assessment of actual versus planned expenditures).

Five Marks Questions with Answers

1. Explain the activities of software management team?

   **Ans: Software management team activities**
   
   The software management team carries the burden of delivering win conditions to all stakeholders. The software management team is responsible for planning the effort and conducting the plan. The software management team takes ownership of all aspects of quality. It is responsible for attaining (reach / achieve) and maintaining balance among these aspects so that the overall solution is adequate for all stakeholders.

   ![Software Management Diagram]

2. Explain the activities of software architecture team activities?

   **Ans: Software Architecture Team activities**

   The software architecture team is responsible for the architecture. For any project, the skill of the software architecture team is crucial. With a good architecture team, an average development team can also succeed but if the architecture is weak, even an expert
Software Project Management

development team of superstar programmers may not succeed. In most projects, the inception and elaboration phases will be dominated by the software management team and the software architecture team.

Software Architecture

Artifacts
- Architecture description
- Requirements set
- Design set
- Release specifications

Responsibilities
- Requirements trade-offs
- Design trade-offs
- Component selection
- Initial integration
- Technical risk resolution

Life-Cycle Focus

<table>
<thead>
<tr>
<th>Inception</th>
<th>Elaboration</th>
<th>Construction</th>
<th>Transition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture prototyping</td>
<td>Architecture baselining</td>
<td>Architecture maintenance</td>
<td>Architecture maintenance</td>
</tr>
<tr>
<td>Make/buy trade-offs</td>
<td>Primary scenario demonstration</td>
<td>Multiple-component issue resolution</td>
<td>Multiple-component issue resolution</td>
</tr>
<tr>
<td>Primary scenario definition</td>
<td>Make/buy trade-off baselining</td>
<td>Performance tuning</td>
<td>Performance tuning</td>
</tr>
<tr>
<td>Architecture evaluation criteria definition</td>
<td></td>
<td>Quality improvements</td>
<td>Quality improvements</td>
</tr>
</tbody>
</table>

3. Explain in detail about periodic status assessments?

Ans:

- Managing risks requires continuous attention to all the interacting activities of a software development effort. 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. The paramount objective of these assessments is to ensure that the expectations of all stakeholders (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
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.

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

4. Explain evolution of organizations?

Ans:

- The project organization represents the architecture of the team and needs to evolve consistent with the project plan captured in the breakdown structure. Figure 11-7 illustrates how the team's center of gravity shifts over the life cycle, with about 50% of
the staff assigned to one set of activities in each phase. A different set of activities is emphasized in each phase, as follows:

- Inception team: an organization focused on planning, with enough support from the other teams to ensure that the plans represent a consensus of all perspectives.

Software project team evolution over the life cycle:

- Elaboration team: an architecture-focused organization in which the driving forces of the project reside in the software architecture team and are supported by the software development and software assessment teams as necessary to achieve a stable architecture baseline.

- Construction team: a fairly balanced organization in which most of the activity resides in the software development and software assessment teams.

- Transition team: a customer-focused organization in which usage feedback drives the deployment activities.

- It is equally important to elaborate the details of subteams, responsibilities, and work.
Software Project Management

packages, but not until the planning details in the WBS are stable. Defining all the details of lower level team structures prematurely can result in serious downstream inefficiencies.

5. Explain cost and schedule estimation process ?

Ans:

Project plans need to be derived from two perspectives. The first is a forward-looking, top-down approach. It starts with an understanding of the general requirements and constraints, derives a macro-level budget and schedule, then decomposes these elements into lower level budgets and intermediate milestones. From this perspective, the following planning sequence would occur:

- The software project manager (and others) develops a characterization of the overall size, process, environment, people, and quality required for the project.
- A macro-level estimate of the total effort and schedule is developed using a software cost estimation model.
- The software project manager partitions the estimate for the effort into a top-level WBS using guidelines. The project manager also partitions the schedule into major milestone dates and partitions the effort into a staffing profile using guidelines. Now there is a project-level plan. These sorts of estimates tend to ignore many detailed project-specific parameters.
- At this point, subproject managers are given the responsibility for decomposing each of the WBS elements into lower levels using their top-level allocation, staffing profile, and major milestone dates as constraints.

The second perspective is a backward-looking, bottom-up approach. You start with the end in mind, analyze the micro-level budgets and schedules, then sum all these elements into the higher level budgets and intermediate milestones. This approach tends to define and populate the WBS from the lowest levels upward. From this perspective, the following planning sequence would occur:

- The lowest level WBS elements are elaborated into detailed tasks, for which budgets and schedules are estimated by the responsible WBS element manager. These estimates tend
to incorporate the project-specific parameters in an exaggerated way.

- Estimates are combined and integrated into higher level budgets and milestones. The biases of individual estimators need to be homogenized so that there is a consistent basis of negotiation.

- Comparisons are made with the top-down budgets and schedule milestones. Gross differences are assessed and adjustments are made in order to converge on agreement between the top-down and the bottom-up estimates.

Milestone scheduling or budget allocation through top-down estimating tends to exaggerate the project management biases and usually results in an overly optimistic plan. Bottom-up estimates usually exaggerate the performer biases and result in an overly pessimistic plan. Iteration is necessary, using the results of one approach to validate and refine the results of the other approach, thereby evolving the plan through multiple versions. This process instills ownership of the plan in all levels of management.

These two planning approaches should be used together, in balance, throughout the life cycle of the project. During the engineering stage, the top-down perspective will dominate because there is usually not enough depth of understanding nor stability in the detailed task sequences to perform credible bottom-up planning. During the production stage, there should be enough precedent experience and planning fidelity that the bottom-up planning perspective will dominate. By then, the top-down approach should be well tuned to the project-specific parameters, so it should be used more as a global assessment technique.
OBJECTIVE TYPE QUESTIONS

1. The term ________ is used to mean thread cohesive and mostly sequential activities.
2. The _______________ is usually the event that transitions the project from elaboration phase into the construction phase.
3. ___ are crucial for focusing continuous attention on the evolving health of the project.
4. The work breakdown structure is the ____________ of project plan.
5. Cost and schedule budgets should be estimated using ____ and _____ techniques.
6. A default WBS consistent with the process framework is phases ____________
7. The __________develops a characterization of the overall size, process, environment.
8. Which is the type of iteration comes under elaboration phase _______________
9. One iteration in inception phase is an ______________
10. Two iterations in construction phase is________________

ANSWERS

<table>
<thead>
<tr>
<th></th>
<th>Workflow</th>
<th>6</th>
<th>workflows and artifacts.</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>major milestone</td>
<td>7</td>
<td>software project manager</td>
</tr>
<tr>
<td>3</td>
<td>Periodic status assessments</td>
<td>8</td>
<td>architecture iterations.</td>
</tr>
<tr>
<td>4</td>
<td>architecture</td>
<td>9</td>
<td>architecture prototype</td>
</tr>
<tr>
<td>5</td>
<td>macro analysis and micro analysis</td>
<td>10</td>
<td>alpha and beta releases.</td>
</tr>
</tbody>
</table>
MULTIPLE CHOICE QUESTIONS

11. The management workflow is concerned with three disciplines
   a) Planning
   b) Project control
   c) Organization
   d) All the above

12. Which of the following is the artifact of assessment workflow?
   a) User manual
   b) Business case
   c) Design set
   d) Vision

13. Programming the components and evolving the implementation and deployment artifacts
   b) Management workflow
   b) Implementation workflow
   c) Assessment workflow
   d) Deployment workflow

14. Which of the following is the artifact of design workflow?
   a) Business case
   b) Architecture description
   c) Vision
   d) Status assessments

15. Which of the following will come under major milestones?
   a) Product release milestone
   b) Iteration readiness review
   c) Iteration assessment review
   d) None of the above

16. Transitioning the end products to the user
   a) Management workflow
b) Implementation workflow

c) Assessment workflow

d) Deployment workflow

17. Which of the following is the artifact of environment workflow?

a) Business case
b) Design set
c) Software change order database
d) User manual

18. Which of the following will come under minor milestones?

a) Iteration readiness review
b) Product release milestone
c) Iteration assessment review
d) a and c

19. Acronym of PRA

a) Process required analysis
b) Project review authority
c) Project review access
d) None of the above

20. What is the environment tool of assessment workflow?

a) Editor
b) Compiler
c) Visual modeling
d) Defect tracking

ANSWERS

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>D</td>
<td>6</td>
<td>D</td>
</tr>
<tr>
<td>2</td>
<td>A</td>
<td>7</td>
<td>C</td>
</tr>
<tr>
<td>3</td>
<td>B</td>
<td>8</td>
<td>D</td>
</tr>
<tr>
<td>4</td>
<td>B</td>
<td>9</td>
<td>B</td>
</tr>
<tr>
<td>5</td>
<td>A</td>
<td>10</td>
<td>D</td>
</tr>
</tbody>
</table>
UNIT-5

Two marks questions with answers

1. What is the balanced application of modern principles to achieve economic results?
   
   Ans:

2. What are the two major improvements in next generation software cost estimation models?
   
   Ans:
   - Separation of the engineering stage from the production stage will force estimators to differentiate between architectural scale and implementation size. This will permit greater accuracy and more-honest precision in life-cycle estimates.
   - Rigorous design notations such as UML will offer an opportunity to define units of measure for scale that are more standardized and therefore can be automated and
Software Project Management

tracked. These measures can also be traced more straightforwardly into the costs of production.

3. List the cultural shifts?

Ans:
- The lower-level managers and the middle level managers should participate in the project development
- Tangible design and requirements
- Assertive Demonstrations are prioritized
- The performance of the project can be determined earlier in the life cycle.
- Earlier increments will be adolescent
- Artifacts tend to be insignificant at the early stages but proves to be the most significant in the later stages
- Identifying and Resolving of real issues is done in a systematic order
- Everyone should focus on quality assurance
- Performance issues crop up earlier in the projects life cycle
- Automation must be done with appropriate investments
- Good software organizations should have good profit margins.

4. What are the characteristics of modern iterative development process framework?

Ans:
- Continuous round-trip engineering from requirements to test at evolving levels of abstraction
- Achieving high fidelity understanding of the drivers as early as practical.
- Evolving the artifacts in breadth and depth based on risk management priorities.
- Postponing completeness and consistency analysis until later in the lifecycle.

5. How to resolve requirements-driven functional decomposition?

Ans:
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.
Three mark questions with answers

6. What is the priorities for tailoring the process framework?

Ans:

2. What are the results of major milestones in a modern process?

Ans:
3. How to resolve late risk resolution?

**Ans:**

By emphasizing an architecture – first approach in which the high-leverage elements of the system are elaborated early in the lifecycle.

1. Define breakage and modularity?

   Breakage is defined as the average extent of change which is the amount of software baseline that needs a rework.

   Modularity is the average breakage trend over time. For a healthy project the trend expectation is decreasing or stable.

2. Define rework and adaptability?

**Ans:**

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.

five mark questions with answers

1. Explain the seven core metrics?
**Software Metrics:**

1. Software metrics are needed for calculating the cost and schedule of a software product with great accuracy.
2. Software metrics are required for making an accurate estimation of the progress.
3. The metrics are also required for understanding the quality of the software product.

**Indicators:**

An indicator is a metric or a group of metrics that provides an understanding of the software process or software product or a software project. A software engineer assembles measures and produce metrics from which the indicators can be derived. Two types of indicators are:

(i) Management indicators.
(ii) Quality indicators.

Management Indicators

The management indicators i.e., technical progress, financial status and staffing progress are used to determine whether a project is on budget and on schedule. The management indicators that indicate financial status are based on earned value system.

1. Work and progress (work performed over time)
2. Budgeted cost and expenditures (cost incurred over time)
3. Staffing and team dynamics (personnel changes over time)

Quality Indicators

The quality indicators are based on the measurement of the changes occurred in software.

1. Change traffic and stability (changes traffic over time)
2. Breakage and modularity (average breakage per change over time)
3. Rework and adaptability (average rework per change over time)
4. Mean time between failures (MTBF) and maturity (defect rate over time)

**Seven Core Metrics of Software Project**

Software metrics instrument the activities and products of the software development/integration process. Metrics values provide an important perspective for managing the
process. The most useful metrics are extracted directly from the evolving artifacts. There are seven core metrics that are used in managing a modern process.

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

2. Explain the stakeholder cohesion and contention?

Ans:
The degree of cooperation and coordination among stakeholders (buyers, developers, users, subcontractors, and maintainers, among others) can significantly drive the specifics of how a process is defined. This process parameter can range from cohesive to adversarial. Cohesive teams have common goals, complementary skills, and close communications. Adversarial teams have conflicting goals, competing or incomplete skills, and less-than-open communications.

A product that is funded, developed, marketed, and sold by the same organization can be set up with a common goal (for example, profitability). A small, collocated organization can be established that has a cohesive skill base and excellent day-to-day communications among team members.

It is much more difficult to set up a large contractual effort without some contention across teams. A development contractor rarely has all the necessary software or domain expertise and frequently must team with multiple subcontractors, who have competing profit goals. Funding authorities and users want to minimize cost, maximize the feature set, and accelerate time to

K. Ramakrishna, Assistant Professor 182
market, while development contractors want to maximize profitability. Large teams are almost impossible to collocate, and synchronizing stakeholder expectations is challenging. All these factors tend to degrade team cohesion and must be managed continuously. Table summarizes key differences in the process primitives for varying levels of stakeholder cohesion.

<table>
<thead>
<tr>
<th>PROCESS PRIMITIVE</th>
<th>FEW STAKEHOLDERS, COHESIVE TEAMS</th>
<th>MULTIPLE STAKEHOLDERS, ADVERSARIAL RELATIONSHIPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Life-cycle phases</td>
<td>Weak boundaries between phases</td>
<td>Well-defined phase transitions to synchronize progress among concurrent activities</td>
</tr>
<tr>
<td>Artifacts</td>
<td>Fewer and less detailed management artifacts required</td>
<td>Management artifacts paramount, especially the business case, vision, and status assessment</td>
</tr>
<tr>
<td>Workflow effort allocations</td>
<td>Less overhead in assessment</td>
<td>High assessment overhead to ensure stakeholder concurrence</td>
</tr>
<tr>
<td>Checkpoints</td>
<td>Many informal events</td>
<td>3 or 4 formal events</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Many informal technical walkthroughs necessary to synchronize technical decisions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Synchronization among stakeholder teams, which can impede progress for weeks</td>
</tr>
<tr>
<td>Management discipline</td>
<td>Informal planning, project control, and organization</td>
<td>Formal planning, project control, and organization</td>
</tr>
<tr>
<td>Automation discipline</td>
<td>(insignificant)</td>
<td>On-line stakeholder environments necessary</td>
</tr>
</tbody>
</table>

2. Explain the process discriminates?

Answ:

In tailoring the management process to a specific domain or project, there are two dimensions of discriminating factors: technical complexity and management complexity. Illustrates these two dimensions of process variability and shows some example project applications. The formality of reviews, the quality control of artifacts, the priorities of concerns, and numerous other process instantiation parameters are governed by the point a project occupies in these two dimensions.
A process framework is not a project-specific process implementation with a well-defined recipe for success. Judgment must be injected, and the methods, techniques, culture, formality, and organization must be tailored to the specific domain to achieve a process implementation that can succeed. The following discussion about the major differences among project processes is organized around six process parameters: the size of the project and the five parameters that affect the process exponent, and hence economies of scale, in COCOMO II. These are some of the critical dimensions that a software project manager must consider when tailoring a process framework to create a practical process implementation.

3. Explain top software management principles?

Ans:

**Principles of Software Management**

- Process must be based on architecture-first approach
- Develop an iterative life-cycle process that identifies the risks at an early stage
After the design methods in-order to highlight components-based development.

Create a change management Environment

Improve change freedom with the help of automated tools that support round-trip engineering.

Process must be implemented or obtaining objective quality control and estimation of progress.

Implement a Demonstration-based Approach for Estimation of intermediately Artifacts

Develop a configuration process that should be economically scalable

4. Explain in detail about process flexibility or rigor?

**Ans:**

The degree of rigor, formality, and change freedom inherent in a specific project's "contract" (vision document, business case, and development plan) will have a substantial impact on the implementation of the project's process. For very loose contracts such as building a commercial product within a business unit of a software company (such as a Microsoft application or a Rational Software Corporation development tool), management complexity is minimal. In these sorts of development processes, feature set, time to market, budget, and quality can all be freely traded off and changed with very little overhead. For example, if a company wanted to eliminate a few features in a product under development to capture market share from the competition by accelerating the product release, it would be feasible to make this decision in less than a week. The entire coordination effort might involve only the development manager, marketing manager, and business unit manager coordinating some key commitments.

On the other hand, for a very rigorous contract, it could take many months to authorize a change in a release schedule. For example, to avoid a large custom development effort, it might be desirable to incorporate a new commercial product into the overall design of a next-generation air traffic control system. This sort of change would require coordination among the development contractor, funding agency, users (perhaps the air traffic controllers' union and major airlines), certification agencies (such as the Federal Aviation Administration), associate contractors for interfacing systems, and others. Large-scale, catastrophic cost-of-
failure systems have extensive contractual rigor and require significantly different management approaches. Table summarizes key differences in the process primitives for varying levels of process flexibility.

<table>
<thead>
<tr>
<th>PROCESS PRIMITIVE</th>
<th>FLEXIBLE PROCESS</th>
<th>INFLEXIBLE PROCESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Life-cycle phases</td>
<td>Tolerant of cavalier phase commitments</td>
<td>More credible basis required for inception phase commitments</td>
</tr>
<tr>
<td>Artifacts</td>
<td>Changeable business case and vision</td>
<td>Carefully controlled changes to business case and vision</td>
</tr>
<tr>
<td>Workflow effort allocations</td>
<td>(insignificant)</td>
<td>Increased levels of management and assessment workflows</td>
</tr>
<tr>
<td>Checkpoints</td>
<td>Many informal events for maintaining technical consistency</td>
<td>3 or 4 formal events</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Synchronization among stakeholder teams, which can impede progress for days or weeks</td>
</tr>
<tr>
<td>Management discipline</td>
<td>(insignificant)</td>
<td>More fidelity required for planning and project control</td>
</tr>
<tr>
<td>Automation discipline</td>
<td>(insignificant)</td>
<td>(insignificant)</td>
</tr>
</tbody>
</table>

5. Explain in detail about process maturity?

**Ans:**

The process maturity level of the development organization, as defined by the Software Engineering Institute's Capability Maturity Model is another key driver of management complexity. Managing a mature process (level 3 or higher) is far simpler than managing an immature process (levels 1 and 2). Organizations with a mature process typically have a high level of precedent experience in developing software and a high level of existing process collateral that enables predictable planning and execution of the process. This sort of collateral includes well-defined methods, process automation tools, trained personnel, planning metrics, artifact templates, and workflow templates. Tailoring a mature
Organization's process for a specific project is generally a straightforward task. Table summarizes key differences in the process primitives for varying levels of process maturity.

<table>
<thead>
<tr>
<th>PROCESS PRIMITIVE</th>
<th>MATURE, LEVEL 3 OR 4 ORGANIZATION</th>
<th>LEVEL 1 ORGANIZATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Life-cycle phases</td>
<td>Well-established criteria for phase transitions</td>
<td>(insignificant)</td>
</tr>
<tr>
<td>Artifacts</td>
<td>Well-established format, content, and production methods</td>
<td>Free-form</td>
</tr>
<tr>
<td>Workflow effort allocations</td>
<td>Well-established basis</td>
<td>No basis</td>
</tr>
<tr>
<td>Checkpoints</td>
<td>Well-defined combination of formal and informal events</td>
<td>(insignificant)</td>
</tr>
<tr>
<td>Management discipline</td>
<td>Predictable planning</td>
<td>Informal planning and project control</td>
</tr>
<tr>
<td></td>
<td>Objective status assessments</td>
<td></td>
</tr>
<tr>
<td>Automation discipline</td>
<td>Requires high levels of automation for round-trip engineering, change management, and process instrumentation</td>
<td>Little automation or disconnected islands of automation</td>
</tr>
</tbody>
</table>

6. Explain in detail about architectural risk?

**Ans:**
The degree of technical feasibility demonstrated before commitment to full-scale production is an important dimension of defining a specific project's process. There are many sources of architectural risk. Some of the most important and recurring sources are system performance (resource utilization, response time, throughput, accuracy), robustness to change (addition of new features, incorporation of new technology, adaptation to dynamic operational conditions), and system reliability (predictable behavior, fault tolerance). The degree to which these risks can be eliminated before construction begins can have dramatic ramifications in the process tailoring. Table summarizes key differences in the process primitives for varying levels of architectural risk.
7. Explain the next generation cost models?

Ans:

Next generation software 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. The passionate overhype or under hype, depending on your perspective, makes it difficult to separate facts from exaggeration. 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. This section provides my perspective on how a software cost model should be structured to best support the estimation of a modern software process. There are cost models and techniques in the industry that can support subsets of this approach. My software cost model is all theory; I have 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. Figure summarizes an hypothesized cost model for an architecture-first development process.
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. My reasoning is that the larger the system, the more opportunity there is to exploit automation and to reuse common processes, components, and architectures.
Software Project Management

There are many solutions to any given problem, as illustrated in Figure, each with a different value proposition. Cost is a key discriminator among potential solutions. Cost estimates that are more accurate and more precise can be derived from specific solutions to problems.

I expect two major improvements in next-generation software cost estimation models:

1. Separation of the engineering stage from the production stage will force estimators to differentiate between architectural scale and implementation size. This will permit greater accuracy and more-honest precision in life-cycle estimates.

2. Rigorous design notations such as UML will offer an opportunity to define units of measure for scale that are more standardized and therefore can be automated and tracked. These measures can also be traced more straightforwardly into the costs of production.

8. Explain the concept of early risk resolution?

   Ans:

   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.
80% of the engineering is consumed by 20% of the requirements. Strive to understand the driving requirements completely before committing resources to full-scale 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.
1). Explain the software management best practices?

   Ans:

   The nine best practices are described next, with my commentary on how they resonate with the process framework, management disciplines, and top 10 principles that I have recommended. (Quotations are presented in italics.)

**1. Formal risk management.**

   Using an iterative process that confronts risk is more or less what this is saying.

**2. Agreement on interfaces.**

   While we may use different words, this is exactly the same intent as my architecture-first principle. Getting the architecture baselined forces the project to gain agreement on the various external interfaces and the important internal interfaces, all of which are inherent in the architecture.

**3. Formal inspections.**

   Ans:

   The assessment workflow throughout the life cycle, along with the other engineering workflows, must balance several different defect removal strategies. The least important strategy, in terms of breadth, should be formal inspection, because of its high costs in human
resources and its low defect discovery rate for the critical architectural defects that span multiple components and temporal complexity.

4. Metric-based scheduling and management.

Ans :
This important principle is directly related to my model-based notation and objective quality control principles. Without rigorous notations for artifacts, the measurement of progress and quality degenerates into subjective estimates.

5. Binary quality gates at the inch-pebble level.

Ans :
This practice is easy to misinterpret. Too many projects have taken exactly this approach early in the life cycle and have laid out a highly detailed plan at great expense. Three months later, when some of the requirements change or the architecture changes, a large percentage of the detailed planning must be rebaselined. A better approach would be to maintain fidelity of the plan commensurate with an understanding of the requirements and the architecture. Rather than inch pebbles, I recommend establishing milestones in the engineering stage followed by inch pebbles in the production stage. This is the primary message behind my evolving levels of detail principle.

6. Program wide visibility of progress versus plan.

Ans :
This practice—namely, open communications among project team members—is obviously necessary. None of my principles traces directly to this practice. It seems so obvious, I let it go without saying.

7. Defect tracking against quality targets.

Ans :
This important principle is directly related to my architecture-first and objective quality control principles. The make-or-break defects and quality targets are architectural. Getting a handle on these qualities early and tracking their trends are requirements for success.
8. Configuration management.

Ans:
The Airlie Software Council emphasized configuration management as key to controlling complexity and tracking changes to all artifacts. It also recognized that automation is important because of the volume and dynamics of modern; large-scale projects, which make manual methods cost-prohibitive and error-prone. The same reasoning is behind my change management principle.


Ans:
This is another management principle that seems so obvious, I let it go without saying.

There is significant overlap and commonality of spirit between my top principles and the Airlie Software Council's best practices. However, I think the Council omitted some important principles: configurability and component-based, model-based, demonstration-based development. This omission is surprising, because my rationale for including component-based and model-based principles was to reduce the complexity of development. This is exactly the stated purpose of the Airlie Software Council. The demonstration-based principle is in my top 10 primarily to force integration to occur continuously throughout the life cycle and to promote better stakeholder relationships through a more meaningful medium of communications. Because the Airlie Software Council was focused on a particular domain—namely, large-scale, nationally important systems—configurability was less important.

The two Airlie Software Council practices I would not have included are inspections and binary quality gates at the inch-pebble level. Although they are useful, they are overemphasized in practice, and there are other important principles that should have been included.
**OBJECTIVE TYPE QUESTIONS**

1. __________ values provide an important perspective for managing the process.  
   __________ is defined as the average extent of change, which is the amount of software base line that needs rework.

2. __________ is defined as the rework trend over time.

3. __________ is defined as the MTBF trend over time.

4. __________ focus on demonstrated results.

5. __________ is the average breakage trend over time.

6. Acronym of SPCP ____________

7. 80% of the engineering is consumed by 20% of the ____________

8. Work and progress means ________________

9. Acronym of MTBF ____________________________

** ANSWERS**

<table>
<thead>
<tr>
<th></th>
<th>Metrics</th>
<th></th>
<th></th>
<th>Modularity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Breakage</td>
<td></td>
<td>7</td>
<td>software project control panel.</td>
</tr>
<tr>
<td>2</td>
<td>Adaptability</td>
<td>8</td>
<td></td>
<td>requirements.</td>
</tr>
<tr>
<td>3</td>
<td>Maturity</td>
<td>9</td>
<td></td>
<td>work performed over time.</td>
</tr>
<tr>
<td>4</td>
<td>Major milestones</td>
<td>10</td>
<td></td>
<td>means time between failures</td>
</tr>
</tbody>
</table>
MULTIPLE CHOICE QUESTIONS

1. Which of the following is a management indicator?
   a) Work
   b) Progress
   c) Staffing
   d) Work and progress

2. Which of the following is the quality indicator?
   a) Staffing
   b) Work and progress
   c) Breakage
   d) Breakage and modularity

3. Acronym of SPCP
   a) Software project control panel
   b) Software project common panel
   c) Software project control procedure
   d) Simple project control panel

4. What is defect rate over time?
   a) Rework
   b) MTBF
   c) Maturity
   d) Adaptability

5. What is called as average cost of change?
   a) Rework
   b) MTBF
   c) Maturity
   d) Adaptability

6. Which is the process discriminates among these?
   a) Scale
   b) Reuse
7. Which of the following are process primitives?
   a) Life-cycle phases
   b) Artifacts
   c) Checkpoints
   d) All the above

8. 80% of the engineering is consumed by 20% of the
   a) Analysis
   b) Requirement
   c) Deployment
   d) Testing

9. Plan intermediate releases in groups of usage scenarios with
   a) Evolving levels of detail
   b) Model based notation
   c) Configurable process
   d) None of the above

10. Requirements and designs are fluid and -----
    a) Intangible
    b) Tangible
    c) Immature d) Mature

**ANSWERS**

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>D</td>
<td>6</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>D</td>
<td>7</td>
<td>D</td>
</tr>
<tr>
<td>3</td>
<td>A</td>
<td>8</td>
<td>B</td>
</tr>
<tr>
<td>4</td>
<td>C</td>
<td>9</td>
<td>A</td>
</tr>
<tr>
<td>5</td>
<td>C</td>
<td>10</td>
<td>B</td>
</tr>
</tbody>
</table>
17. beyond syllabus topic with materials

Beyond Syllabus Topics with Material

COCOMO Model

Specific Instructional Objectives

At the end of this lesson the student would be able to:

- Differentiate among organic, semidetached and embedded software projects.
- Explain basic COCOMO.
- Differentiate between basic COCOMO model and intermediate COCOMO model.
- Explain the complete COCOMO model.

Organic, Semidetached and Embedded software projects

Boehm postulated that any software development project can be classified into one of the following three categories based on the development complexity: organic, semidetached, and embedded. In order to classify a product into the identified categories, Boehm not only considered the characteristics of the product but also those of the development team and development environment. Roughly speaking, these three product classes correspond to application, utility and system programs, respectively. Normally, data processing programs are considered to be application programs. Compilers, linkers, etc., are utility programs. Operating systems and real-time system programs, etc. are system programs. System programs interact directly with the hardware and typically involve meeting timing constraints and concurrent processing.

Boehm’s [1981] definition of organic, semidetached, and embedded systems are elaborated below.

Organic: A development project can be considered of organic type, if the project deals with developing a well understood application program, the size of the development team is reasonably small, and the team members are experienced in developing similar types of projects.

Semidetached: A development project can be considered of semidetached type, if the
Software Project Management

development consists of a mixture of experienced and inexperienced staff. Team members may have limited experience on related systems but may be unfamiliar with some aspects of the system being developed.

**Embedded:** A development project is considered to be of embedded type, if the software being developed is strongly coupled to complex hardware, or if the stringent regulations on the operational procedures exist.

**COCOMO**

COCOMO (Constructive Cost Estimation Model) was proposed by Boehm [1981]. According to Boehm, software cost estimation should be done through three stages: Basic COCOMO, Intermediate COCOMO, and Complete COCOMO.

**Basic COCOMO Model**

The basic COCOMO model gives an approximate estimate of the project parameters. The basic COCOMO estimation model is given by the following expressions:

- \( \text{Effort} = a_1 \times (\text{KLOC})^2 \text{PM} \)
- \( \text{Tdev} = b_1 \times (\text{Effort})^2 \text{Months} \)

Where

- KLOC is the estimated size of the software product expressed in Kilo Lines of Code,
- \( a_1, a_2, b_1, b_2 \) are constants for each category of software products,
- Tdev is the estimated time to develop the software, expressed in months,
- Effort is the total effort required to develop the software product, expressed in person months (PMs).

The effort estimation is expressed in units of person-months (PM). It is the area under the person-month plot (as shown in fig. 11.3). It should be carefully noted that an effort of 100 PM does not imply that 100 persons should work for 1 month nor does it imply that 1 person should be employed for 100 months, but it denotes the area under the person-month curve (as shown in fig.).
According to Boehm, every line of source text should be calculated as one LOC irrespective of the actual number of instructions on that line. Thus, if a single instruction spans several lines (say n lines), it is considered to be nLOC. The values of a1, a2, b1, b2 for different categories of products (i.e. organic, semidetached, and embedded) as given by Boehm [1981] are summarized below. He derived the above expressions by examining historical data collected from a large number of actual projects.

Estimation of development effort

For the three classes of software products, the formulas for estimating the effort based on the code size are shown below:

Organic : \( \text{Effort} = 2.4(KLOC)^{1.05} \text{ PM} \)

Semi-detached : \( \text{Effort} = 3.0(KLOC)^{1.12} \text{ PM} \)

Embedded : \( \text{Effort} = 3.6(KLOC)^{1.20} \text{ PM} \)

Estimation of development time

For the three classes of software products, the formulas for estimating the development time based on the effort are given below:
some insight into the basic COCOMO model can be obtained by plotting the estimated characteristics for different software sizes. Fig. 11.4 shows a plot of estimated effort versus product size. From fig. 11.4, we can observe that the effort is somewhat superlinear in the size of the software product. Thus, the effort required to develop a product increases very rapidly with project size.

The development time versus the product size in KLOC is plotted in fig. 11.5. From fig. 11.5, it can be observed that the development time is a sublinear function of the size of the product, i.e. when the size of the product increases by two times, the time to develop the product does not double but rises moderately. This can be explained by the fact that for larger products, a larger number of activities which can be carried out concurrently can be identified. The parallel activities can be carried out simultaneously by the engineers. This reduces the time to complete the project. Further, from fig. 11.5, it can be observed that the development time is roughly the same for all the three categories of products. For example, a 60 KLOC program can be developed in approximately 18 months, regardless of whether it is of organic, semidetached, or embedded type.
From the effort estimation, the project cost can be obtained by multiplying the required effort by the manpower cost per month. But, implicit in this project cost computation is the assumption that the entire project cost is incurred on account of the manpower cost alone. In addition to manpower cost, a project would incur costs due to hardware and software required for the project and the company overheads for administration, office space, etc.

It is important to note that the effort and the duration estimations obtained using the COCOMO model are called as nominal effort estimate and nominal duration estimate. The term nominal implies that if anyone tries to complete the project in a time shorter than the estimated duration, then the cost will increase drastically. But, if anyone completes the project
over a longer period of time than the estimated, then there is almost no decrease in the estimated cost value.

**Example:**

Assume that the size of an organic type software product has been estimated to be 32,000 lines of source code. Assume that the average salary of software engineers be Rs. 15,000/- per month. Determine the effort required to develop the software product and the nominal development time.

From the basic COCOMO estimation formula for organic software:

\[
\text{Effort} = 2.4 \times (32)^{1.05} = 91 \text{ PM}
\]

\[
\text{Nominal development time} = 2.5 \times (91)^{0.38} = 14 \text{ months}
\]

\[
\text{Cost required to develop the product} = 14 \times 15,000 = \text{Rs. 210,000/-}
\]

Intermediate COCOMO model

The basic COCOMO model assumes that effort and development time are functions of the product size alone. However, a host of other project parameters besides the product size affect the effort required to develop the product as well as the development time. Therefore, in order to obtain an accurate estimation of the effort and project duration, the effect of all relevant parameters must be taken into account. The intermediate COCOMO model recognizes this fact and refines the initial estimate obtained using the basic COCOMO expressions by using a set of 15 cost drivers (multipliers) based on various attributes of software development. For example, if modern programming practices are used, the initial estimates are scaled downward by multiplication with a cost driver having a value less than 1. If there are stringent reliability requirements on the software product, this initial estimate is scaled upward. Boehm requires the project manager to rate these 15 different parameters for a particular project on a scale of one to three. Then, depending on these ratings, he suggests appropriate cost driver values which should be multiplied with the initial estimate obtained using the basic COCOMO. In general, the cost drivers can be classified as being attributes of the following items:

**Product:** The characteristics of the product that are considered include the inherent complexity of the product, reliability requirements of the product, etc.
Computer: Characteristics of the computer that are considered include the execution speed required, storage space required etc.

Personnel: The attributes of development personnel that are considered include the experience level of personnel, programming capability, analysis capability, etc.

Development Environment: Development environment attributes capture the development facilities available to the developers. An important parameter that is considered is the sophistication of the automation (CASE) tools used for software development.

Complete COCOMO model

A major shortcoming of both the basic and intermediate COCOMO models is that they consider a software product as a single homogeneous entity. However, most large systems are made up several smaller sub-systems. These sub-systems may have widely different characteristics. For example, some sub-systems may be considered as organic type, some semidetached, and some embedded. Not only that the inherent development complexity of the subsystems may be different, but also for some subsystems the reliability requirements may be high, for some the development team might have no previous experience of similar development, and so on. The complete COCOMO model considers these differences in characteristics of the subsystems and estimates the effort and development time as the sum of the estimates for the individual subsystems. The cost of each subsystem is estimated separately. This approach reduces the margin of error in the final estimate.

The following development project can be considered as an example application of the complete COCOMO model. A distributed Management Information System (MIS) product for an organization having offices at several places across the country can have the following sub-components:

- Database part
- Graphical User Interface (GUI) part
- Communication part

Of these, the communication part can be considered as embedded software. The database part could be semi-detached software, and the GUI part organic software. The costs for these three components can be estimated separately, and summed up to give the overall cost of the system.