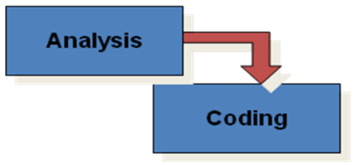
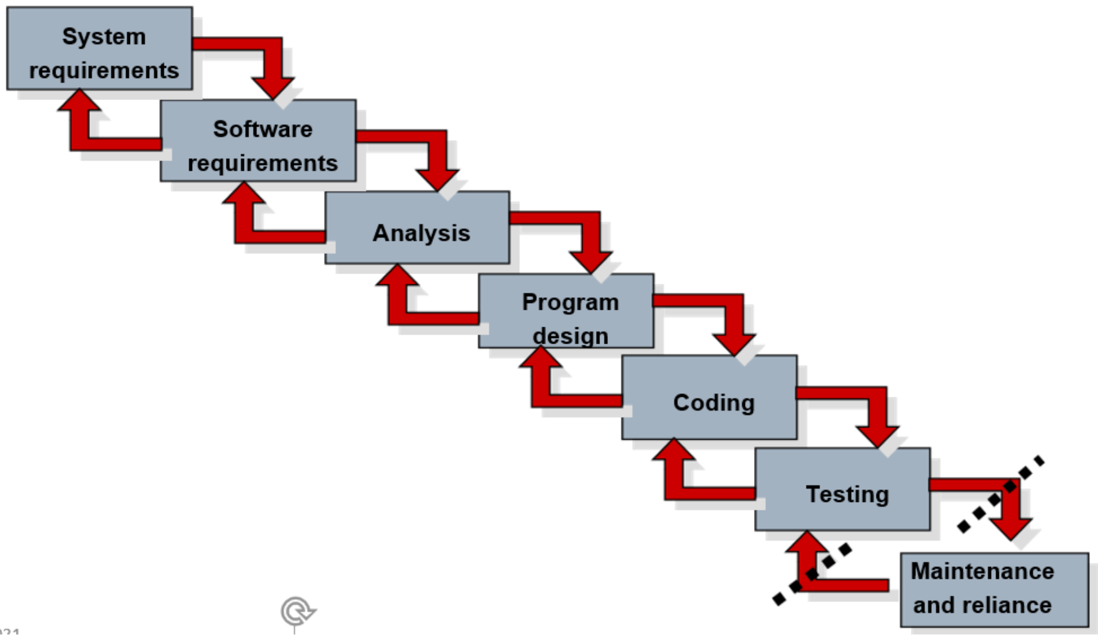
UNIT-1

1 Waterfall Model

- The **Waterfall Model** is the baseline process for most of the software projects.
- There are two essential steps common to the development of computer programs: analysis and coding.



- Both involve creative work that directly contributes to the usefulness of the end product. In order to manage and control all of the intellectual freedom associated with software development, one must introduce several other 'overhead' steps, including system requirements definition, software requirements definition, program design and testing.
- These are the phases of water¯fall model



;

Waterfall problem

- Inflexible partitioning of the project into distinct stages.
- This makes it difficult to respond to changing customer requirement. Therefore, this model is only appropriate when the requirements are well-understood.

Five necessary improvements for waterfall model are:-

1. **Program design comes first.**

   Insert a preliminary program design phase between the software requirements generation phase and the analysis phase. By this technique, the program designer assures that the software will not fail because of storage, timing, and data flux (continuous change). As analysis proceeds in the succeeding phase, the program designer must impose on the analyst the storage, timing, and operational constraints in such a way that he senses the consequences. If the total resources to be applied are insufficient or if the embryonic(in an early stage of development) operational design is wrong, it will be recognized at this early stage and the iteration with requirements and preliminary design can be redone before final design, coding, and test commences. How is this program design procedure implemented?

The following steps are required:

Begin the design process with program designers, not analysts or programmers. Design, define, and allocate the data processing modes even at the risk of being wrong. Allocate processing functions, design the database, allocate execution time, define interfaces and processing modes with the operating system, describe input and output processing, and define preliminary operating procedures. 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.

2. **Document the design.**

The amount of documentation required on most software programs is quite a lot, certainly much more than most programmers, analysts, or program designers are willing to do if left to their own devices.

Why do we need so much documentation?

- Each designer must communicate with interfacing designers, managers, and possibly customers.
- During early phases, the documentation is the design.
- The real monetary value of documentation is to support later modifications by a separate test team, a separate maintenance team, and operations personnel who are not software literate.

3. **Do it twice.**

If a computer program is being 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. Note that this is simply the entire process done in miniature, to a time scale that is relatively small with respect to the overall effort. In the first version, the team must have a special broad competence where they can quickly sense trouble spots in the design, model them, model alternatives, forget the straightforward aspects of the design that aren't worth studying at this early point, and, finally, arrive at an error-free program.

4. **Plan, control, and monitor testing.**

Without question, the biggest user of project resources-manpower, computer time, and/or management judgment-is the test phase. This is the phase of greatest risk in terms of cost and schedule. It occurs at the latest point in the schedule, when backup alternatives are least available, if at all. The previous three recommendations were all aimed at uncovering and solving problems before entering the test phase.

However, even after doing these things, there is still a test phase and there are still important things to be done, including:

(1) employ a team of test specialists who were not responsible for the Operation Testing original design;

(2) employ visual inspections to spot the obvious errors like dropped minus signs, missing factors of two, jumps to wrong addresses (do not use the computer to detect this kind of thing, it is too expensive);
(3) test every logic path;
(4) employ the final checkout on the target computer.

5. **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. There are three points following requirements definition where the insight, judgment, and commitment of the customer can bolster the development effort. These include a "preliminary software review" following the preliminary program design step, a sequence of "critical software design reviews" during program design, and a "final software acceptance review".

The five necessary improvements to make the approach to work: -

1.Complete program design before analysis and coding begin.
2.Maintain current and complete documentation.
3.Do the job twice, if possible.
4.Plan,control,and monitor testing.
5.Involve the customer

The general troubles that conventional framework faces: -

1. Protracted Integration and late design breakage
2. Late Risk Resolution
3. Requirements-Driven Functional Decomposition
4. Adversarial Stakeholder Relationships
5. Focus on documents and Review Meetings

2 Conventional Software Management Performance Or Boehm’s top 10 list about conventional software management performance?

- Finding and fixing a software problem after delivery costs 100 times more than finding and fixing the problem in early design phases.
- You can compress software development schedules 25% of nominal, but no more.
- For every $1 you spend on development, you will spend $2 on maintenance.
- Software development and maintenance costs are primarily a function of the number of source lines of code.
- Variations among people account for the biggest differences in software productivity.

- The overall ratio of software to hardware costs is still growing. In 1955 it was 15:85; in 1985, 85:15.
- Only about 15% of software development effort is devoted to programming.
- Software Systems and products typically cost 3 times as much per SLOC as individual software programs.
- Software system products costs 9 times as much.
- Walkthroughs catch 60% of the errors.
- 80% of the contribution comes from 20% of contributors.

3 five components of software cost models

Most software cost models can be abstracted into a function of five basic parameters: size, process, personnel, environment, and required quality.

- The size of the end product (in human-generated components), which is typically quantified in terms of the number of source instructions or the number of function points required to develop the required functionality
- The process used to produce the end product, in particular the ability of the process to avoid nonvalue-adding activities (rework, bureaucratic delays, communications overhead)
- The capabilities of software engineering personnel, and particularly their experience with the computer science issues and the applications domain issues of the project
- The environment, which is made up of the tools and techniques available to support efficient software development and to automate the process
- The required quality of the product, including its features, performance, reliability, and adaptability

The relationships among these parameters and the estimated cost can be written as follows:

Effort = (Personnel) (Environment) (Quality) (Sizeprocess)

The three generations of software development are defined as follows:

- Conventional: 1960s and 1970s, craftsmanship. Organizations used custom tools, custom processes, and virtually all custom components built in primitive languages. Project performance was highly predictable in that cost, schedule, and quality objectives were almost always underachieved.
- Transition: 1980s and 1990s, software engineering. Organiz:1tions used more-repeatable processes and off the-shelf tools, and mostly (>70%) custom components built-in higher-level languages.
- Modern practices: 2000 and later, software production. This book's philosophy is rooted in the use of managed and measured processes, integrated automation environments, and mostly (70%) off-the-shelf components. Perhaps as few as 30% of the components need to be custom built Technology.

4 Team Effectiveness

Team effectiveness is the capacity of a group of people, usually with complementary skills, to work together to accomplish goals set out by an authority, members, or leaders of the team. Team effectiveness models help us understand the best management techniques to get optimal performance from our teams. Teamwork is much more important than the sum of the individuals. With software teams, a project manager needs to configure a balance of solid talent with highly skilled people in the Leverage positions.

Some maxims of team management include the following:

- A well-managed project can succeed with a nominal engineering team.
- A mismanaged project will almost never succeed, even with an expert team of engineers.
- A well-architected system can be built by a nominal team of software builders.
- A poorly architected system will flounder even with an expert team of builders.

Improving Team Effectiveness

- The principle of top talent: *Use better and fewer people.*
- The principle of job matching: *Fit the task to the skills an 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 one another.*
- The principle of phase-out: *Keeping a misfit on the team doesn't benefit anyone.*

5 Principles of conventional software engineering performance

1. 1.Make quality #1. Quality must be quantified and mechanisms put into place to motivate its achievement
2. 2.High-quality software is possible. Techniques that have been demonstrated to increase quality include involving the customer, prototyping, simplifying design, conducting inspections, and hiring the best people
3. 3.Give products to customers early. No matter how hard you try to learn users' needs during the requirements phase, the most effective way to determine real needs is to give users a product and let them play with it
4. 4.Determine the problem before writing the requirements. When faced with what they believe is a problem, most engineers rush to offer a solution. Before you try to solve a problem, be sure to explore all the alternatives and don't be blinded by the obvious solution
5. 5.Evaluate design alternatives. After the requirements are agreed upon, you must examine a variety of architectures and algorithms. You certainly do not want to use" architecture" simply because it was used in the requirements specification.

6. 6.Use an appropriate process model. Each project must select a process that makes ·the most sense for that project on the basis of corporate culture, willingness to take risks, application area, volatility of requirements, and the extent to which requirements are well understood.
7. 7.Use different languages for different phases. Our industry's eternal thirst for simple solutions to complex problems has driven many to declare that the best development method is one that uses the same notation throughout the life cycle.
8. 8.Minimize intellectual distance. To minimize intellectual distance, the software's structure should be as close as possible to the real-world structure
9. 9.Put techniques before tools. An undisciplined software engineer with a tool becomes a dangerous, undisciplined software engineer
10. 10.Get it right before you make it faster. It is far easier to make a working program run faster than it is to make a fast program work. Don't worry about optimization during initial coding
11. 11.Inspect code. Inspecting the detailed design and code is a much better way to find errors than testing
12. 12.Good management is more important than good technology. Good management motivates people to do their best, but there are no universal "right" styles of management.
13. 13.People are the key to success. Highly skilled people with appropriate experience, talent, and training are key.
14. 14.Follow with care. Just because everybody is doing something does not make it right for you. It may be right, but you must carefully assess its applicability to your environment.
15. 15.Take responsibility. When a bridge collapses we ask, "What did the engineers do wrong?" Even when software fails, we rarely ask this. The fact is that in any engineering discipline, the best methods can be used to produce awful designs, and the most antiquated methods to produce elegant designs.
16. 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. 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. 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. 19. Design for change. The architectures, components, and specification techniques you use must Software Project Management 2 accommodate change.
20. 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. 21. Use tools, but be realistic. Software tools make their users more efficient.
22. 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. 23. Encapsulate. Information-hiding is a simple, proven concept that results in software that is easier to test and much easier to maintain.

24. 24. Use coupling and cohesion. Coupling and cohesion are the best ways to measure software's inherent maintainability and adaptability
25. 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. 26.Don't test your own software. Software developers should never be the primary testers of their own software.
27. 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. 28.Realize that software's entropy increases. Any software system that undergoes continuous change will grow in complexity and will become more and more disorganized
29. 29.People and time are not interchangeable. Measuring a project solely by person-months makes little sense
30. 30.Expect excellence. Your employees will do much better if you have high expectations for them.

UNIT-2

1 Primary objectives and essential activities of 4 phases of lifecycles

The Life cycle phases

The following are the two stages of the life-cycle:

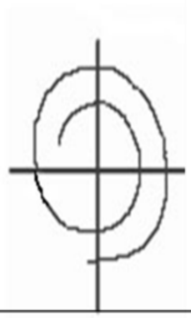
- The engineering stage – driven by smaller teams doing design and synthesis activities

- The production stage – driven by larger teams doing construction, test, and deployment activities

LIFE-CYCLE ASPECT	**ENGINEERING STAGE EMPHASIS**	**PRODUCTION STAGE EMPHASIS**
Risk reduction	Schedule, technical feasibility	Cost
Products	Architecture baseline	Product release baselines
Activities	Analysis, design, planning	Implementation, testing
Assessment	Demonstration, inspection, analysis	Testing
Economics	Resolving diseconomies of scale	Exploiting economics of scale
Management	Planning	Operations

The engineering stage is decomposed into two distinct phases, inception and elaboration, and the production stage into construction and transition. These four phases of the life-cycle process are loosely mapped to the conceptual framework of the spiral model.

The size of the spiral model corresponds to the inertia of the project with respect to the breadth and depth of the artifacts that have been developed.

Engineering Stage		Production Stage	
Inception	Elaboration	Construction	Transition
			
Idea	Architecture	Beta Releases	Products

In most conventional life cycles, the phases are named after the primary activity within each phase: requirements analysis, design, coding, unit test, integration test, and system test. Conventional software development efforts emphasized a mostly sequential process, in which one activity was required to be complete before the next was begun.

Within an iterative process, each phase includes all the activities, in varying proportions.

1. **Inception Phase:**

   - Overriding goal of the inception phase is to achieve concurrence among stakeholders on the life- cycle objectives

   - Essential activities :
     - *Formulating the scope of the project* (capturing the requirements and operational concept in an information repository)

     - *Synthesizing the architecture* (design trade-offs, problem space ambiguities, and available solution-space assets are evaluated)

     - *Planning and preparing a business case* (alternatives for risk management, iteration planes, and cost/schedule/profitability trade- offs are evaluated)

2. **Elaboration Phase:**

   It is easy to argue that the elaboration phase is the most critical of the four phases. At the end of this phase, the "engineering "is considered complete and the project faces its reckoning. During the elaboration phase, an executable architecture prototype is built in one or more iterations, depending on the scope, size, risk and novelty of the project.

- Essential activities :

  - *Elaborating the vision* (establishing a high-fidelity understanding of the critical use cases that drive architectural or planning decisions)
  - *Elaborating the process and infrastructure* (establishing the construction process, the tools and process automation support
  - *Elaborating the architecture and selecting components* (lessons learned from these activities may result in redesign of the architecture

3. Construction Phase:

- During the construction phase :
  All remaining components and application features are integrated into the application all features are thoroughly tested

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

4. Transition Phase:

- The transition phase is entered when baseline is mature enough to be deployed in the end-user domain. This phase could include beta testing, conversion of operational databases, and training of users and maintainers.

- Essential activities :
  - *Synchronization and integration of concurrent construction into consistent deployment baselines*
  - *Deployment-specific engineering* (commercial packaging and production, field personnel training)
  - *Assessment of deployment baselines against the complete vision and acceptance criteria in the requirements set*

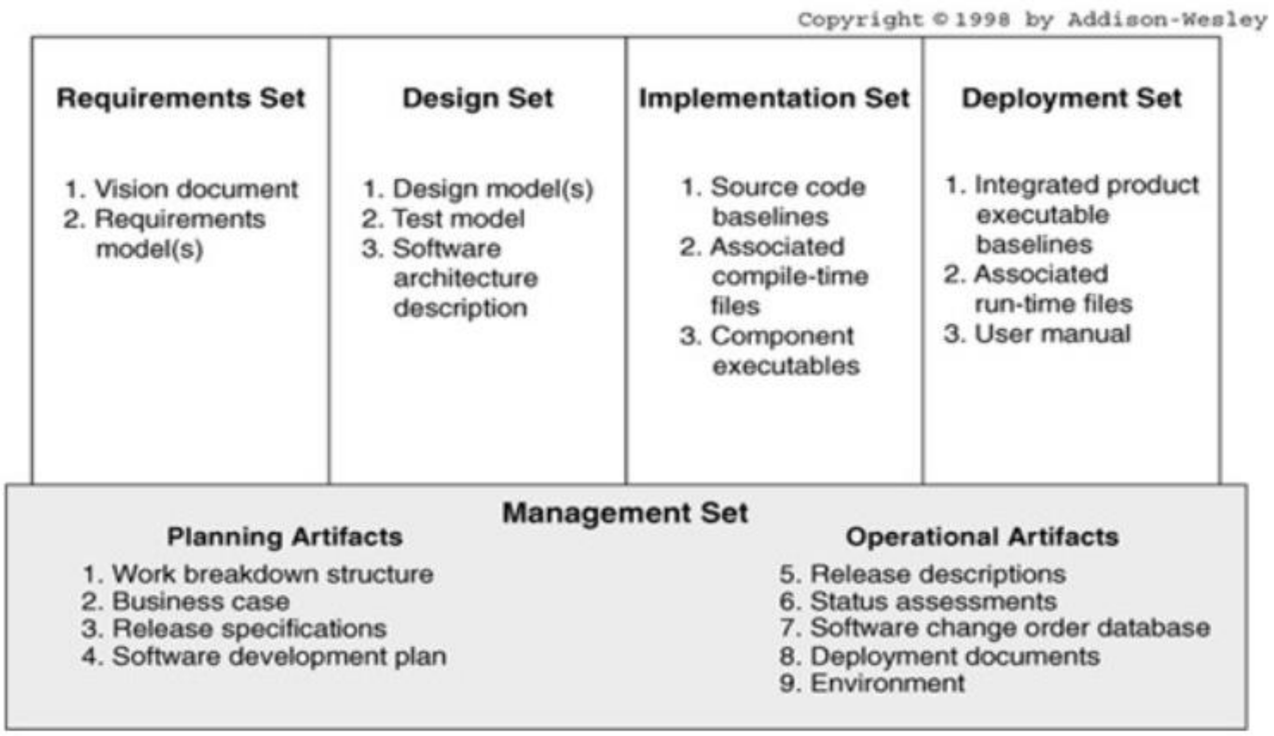
Evaluation Criteria:

- Is the user satisfied?
- Are actual resource expenditures versus planned expenditures acceptable?
- Each of the four phases consists of one or more iterations in which some technical capability is produced in demonstrable form and assessed against a set of the criteria.
- The transition from one phase to the nest maps more to a significant business decision than to the completion of specific software activity.

2 Artifact Sets.

- Artifact is highly associated and related to specific methods or processes of development.
- Methods or processes can be project plans, business cases, or risk assessments.
- Distinct gathering and collections of detailed information are generally organized and incorporated into artifact sets.
- Artifacts of the life-cycle of software are generally organized and divided into two sets i.e., Management set and Engineering set.
- These sets are further divided or partitioned by underlying language of set.

Overview of the Artifact sets



Management Set

- This set usually captures artifacts that are simply associated with planning and execution or running process.
- These artifacts generally make use of ad hoc notations. It also includes text, graphics or whatever representation is required or needed to simply capture "contracts" among all project personnel (such as project developers, project management, etc.), among different stakeholders (such as user, project manager, etc.), and even between stakeholders and project personnel.
- This set includes various artifacts such as work breakdown structure, business case, software development plan, deployment, Environment.

Requirements Set

- This set is primary engineering context simply used for evaluating other three artifact sets of engineering set and basis of test cases.
- Artifacts of this set are evaluated, checked, and measured through combination of following:
    - Analysis of consistency among present vision and requirements models.
    - Analysis of consistency with supplementary specifications of management set.
    - Analysis of consistency among requirement models.

Design Set

- Tools that are used in Visually modeling tools. To engineer design model, UML (Unified Modeling Language) notations are used.
- This sets simply contains many different levels of abstractions.
- Design model generally includes all structural and behavioral data or information to ascertain bill of material.
- These set artifacts mostly include test models, design models, software architecture descriptions.
- The design set is evaluated, assessed, and measured through a combination of the following:
    - Analysis of the internal consistency and quality of the design model
    - Analysis of consistency with the requirements models
    - Translation into implementation and deployment sets and notations to evaluate the consistency and completeness and the semantic balance between information in the sets
    - Analysis of changes between the current version of the design model and previous versions (scrap, rework, and defect elimination trends)
    - Subjective review of other dimensions of quality

Implementation Set

- The implementation set includes source code (programming language notations) that represents the tangible implementations of components (their form, interface, and dependency relationships).
- Implementation sets are human-readable formats that are evaluated, assessed, and measured through a combination of the following:
    - Analysis of consistency with the design models.
    - Translation into deployment set notations (for example, compilation and linking) to evaluate the consistency and completeness among artifact sets.
    - Assessment of component source or executable files against relevant evaluation criteria through inspection, analysis, demonstration, or testing

- Execution of stand-alone component test cases that automatically compare expected results with actual results.
- Analysis of changes between the current version of the implementation set and previous versions (scrap, rework, and defect elimination trends).
- Subjective review of other dimensions of quality.

Deployment Set

- The deployment set 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 the following:
    - 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 between the current version of the deployment set and previous versions
    - Subjective review of other dimensions of quality.

3 Principles of software architecture in modern software development process

- **Base the process on an architecture-first approach.** This requires that a demonstrable balance be achieved among the driving requirements, the architecturally significant design decisions, and the lifecycle plans before the resources are committed for full-scale development.
- **Establish an iterative life-cycle process that confronts risk early.** With today's sophisticated software systems, it is not possible to define the entire problem, design the entire solution, build the software, and then test the end product in sequence. Instead, an iterative process that refines the problem understanding, an effective solution, and an effective plan over several iterations encourages a balanced treatment of all stakeholder objectives. Major risks must be addressed early to increase predictability and avoid expensive downstream scrap and rework.
- **Transition design methods to emphasize component-based development.** Moving from a line-of-code mentality to a component-based mentality is necessary to reduce the amount of human-generated source code and custom development.

- **Establish a change management environment.** The dynamics of iterative development, including concurrent workflows by different teams working on shared artifacts, necessitates objectively controlled baselines. Software Project Management 3 5.
- **Enhance change freedom through tools that support round-trip engineering.** Round-trip engineering is the environment support necessary to automate and synchronize engineering information in different formats(such as requirements specifications, design models, source code, executable code, test cases).
- **Capture design artifacts in rigorous, model-based notation.** A model based approach (such as UML) supports the evolution of semantically rich graphical and textual design notations.
- **Instrument the process for objective quality control and progress assessment.** Life-cycle assessment of the progress and the quality of all intermediate products must be integrated into the process.
- **Use a demonstration-based approach to assess intermediate artifacts.**
- **Plan intermediate releases in groups of usage scenarios with evolving levels of detail.** It is essential that the software management process drive toward early and continuous demonstrations within the operational context of the system, namely its use cases. Establish a configurable process that is economically scalable. No single process is suitable for all software developments.

4 Discuss the planning artifacts of a management set.

- The management set includes several artifacts that capture intermediate results and ancillary information necessary to document the product/process legacy, maintain the product, improve the product and improve the process.
- Some types of Management Artifacts:
    1. Business case
    2. Software Development Plan
    3. Work Breakdown structure
    4. Software change order Database
    5. Release Specifications
    6. Deployment
    7. Environment
- **Business Case:** The business case artifact provides all the information necessary to determine whether the project is worth investing in.
- **Software Development Plan:** The software development plan (SDP) elaborates the process framework into a fully detailed plan. Two indications of a useful SDP are periodic updating (it is not stagnant shelf ware) and understanding and acceptance by managers and practitioners alike.

- **Work breakdown structure (WBS):** It is the vehicle for budgeting and collecting costs. To monitor and control a project's financial performance, the software project manager must have insight into project costs and how they are expended.

- **Software Change Order Database:** Managing change is one of the fundamental primitives of an iterative development process. With greater change freedom, a project can iterate more productively.
- **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
- **Release Descriptions:** Release description documents describe the results of each release, including performance against each of the evaluation criteria in the corresponding release specification.
- **Status Assessments:** Status assessments provide periodic snapshots of project health and status, including the software project manager's risk assessment, quality indicators, and management indicators.
- **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:** Depending on the project, it could include several document subsets for transitioning the product into operational status.
- **Management Artifact Sequences:** In each phase of the life cycle, new artifacts are produced and previously developed artifacts are updated to incorporate lessons learned and to capture further depth and breadth of the solution.

5 Model based software architecture

INTRODUCTION:

- Software architecture is the central design problem of a complex software system in the same way an architecture is the software system design.
- The ultimate goal of the engineering stage is to converge on a stable architecture baseline.
- Architecture is not a paper document. It is a collection of information across all the engineering sets.
- Architectures are described by extracting the essential information from the design models.
- A model is a relatively independent abstraction of a system.
- A view is a subset of a model that abstracts a specific, relevant perspective.

ARCHITECTURE: A 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 inter project communications, as captured in software architecture, must be accurate and precise.

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

- 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.
- 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, and people).
- 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.

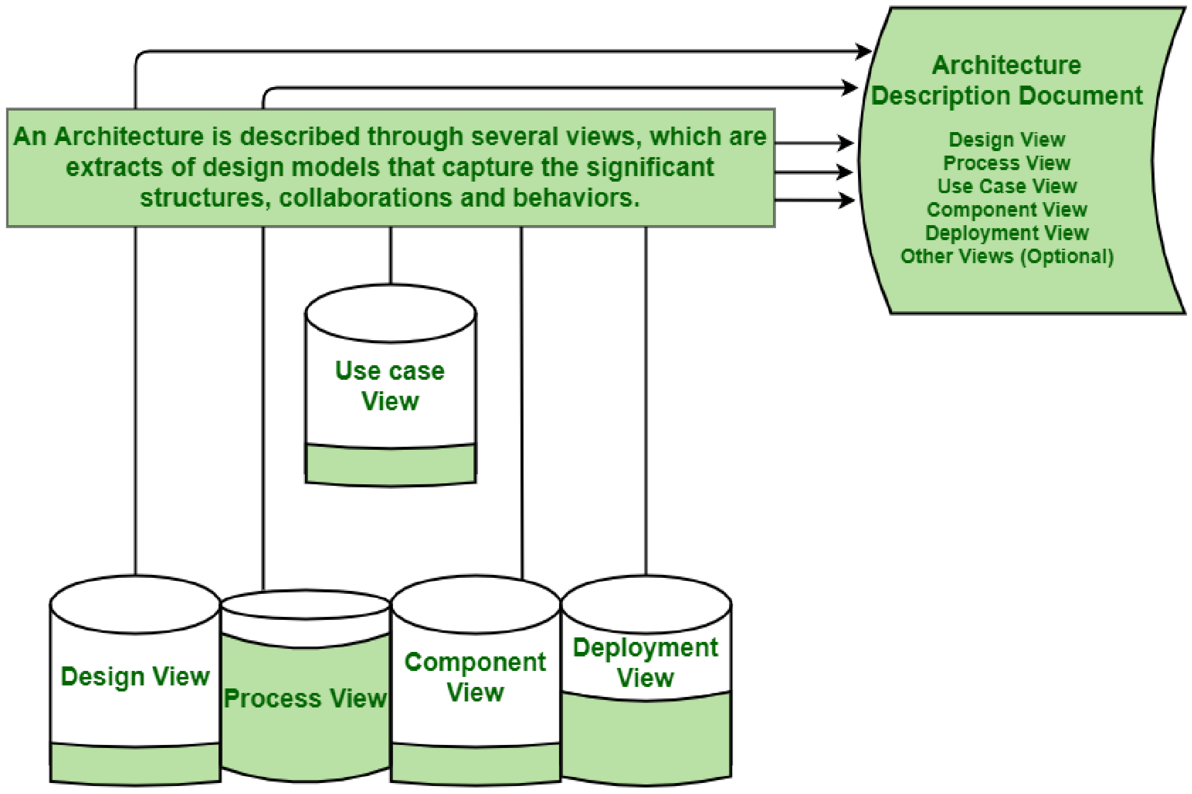
ARCHITECTURE: A TECHNICAL PERSPECTIVE

An architecture framework is defined in terms of views that are abstractions of the UML models in the design set. The design model includes the full breadth and depth of information.

An architecture view is an abstraction of the design model, it contains only the architecturally significant information.

Most real world systems require four views: design, process, component, and deployment. The purposes of these views are as follows:

- Design: describes architecturally significant structures and functions of the design model.
- Process: describes concurrency and control thread relationships among the design components, and deployment views.
- Component: describes the structure of the implementation set.
- Deployment: describes the structure of the deployment set.


An Architecture is described through several views, which are extracts of design models that capture the significant structures, collaborations and behaviors.
Architecture Description Document
Design View
Process View
Use Case View
Component View
Deployment View
Other Views (Optional)
Use case View
Design View
Process View
Component View
Deployment View

A model that draws on the foundation of architecture developed at Rational Software Corporation and particularly on Philippe, collaborations and behvaiors.

- ➢ The use case view describes how the system's critical use cases are realized by elements of the design model. It is modeled statically using case diagrams, and dynamically using any of the UML behavioral diagrams. The design view addresses the basic structure and the functionality of the solution.

- The process view addresses the run-time collaboration issues involved in executing the architecture on a distributed deployment model, including the logical software network topology, interprocess communicationand state management.
- The component view describes the architecturally significant elements of the implementation set and addresses the software source code realization of the system from perspective of the project's integrators and developers.
- The deployment view addresses the executable realization of the system, including the allocation of logical processes in the distribution view to physical resources of the deployment network.

Generally an architecture baseline should include the following:

- Requirements: critical use cases, system level quality objectives, and priority relationship among features and qualities
- Design: names, attributes, structures, behaviors, groupings and relationships of significant classes and components
- Implementation: source component inventory and bill of materials (number, name, purpose, cost) of all primitive components
- Deployment: executable components sufficient to demonstrate the critical use cases and the risk associated with achieving the system qualities

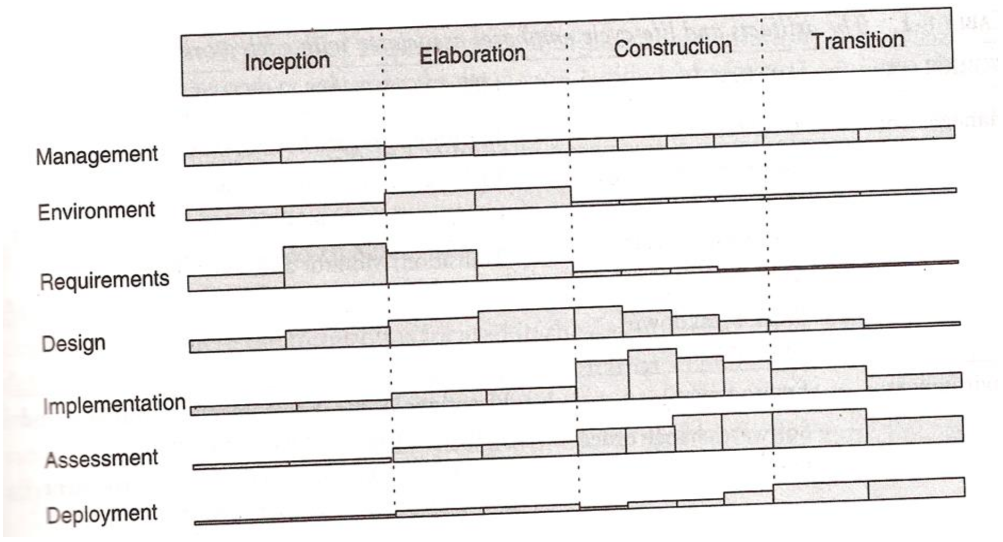
UNIT-3

1. SEVEN SOFTWARE PROCESS WORKFLOWS

1. A : **Workflow –**
   In general workflow refers to the series of sequential tasks those are performed to achieve certain goal. Each workflow step is defined by three parameters i.e input, transformation, and output. In workflow process a series of actions are performed to achieve a business outcome.
2. **Software Process Workflows –**
   Software process is the set of related activities those are performed to get a software product as an outcome and there the software process workflows leads the software developments in a linear way by performing series of sequential tasks.

There are top 7 Software Process Workflows in Software Project Management –

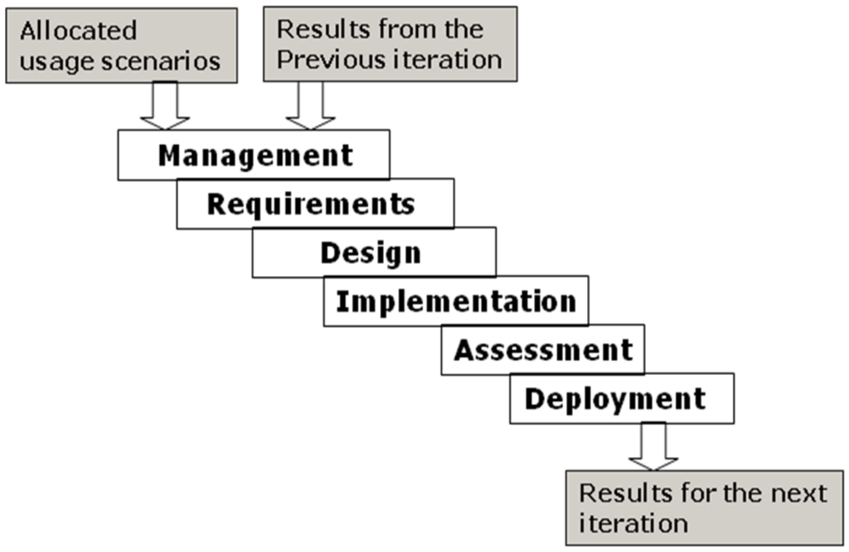
1. **Management workflow –**
   Some of the essential steps of controlling process are carried out in management workflow. The artifacts include Software Development Plan (SDP), business case, vision etc.
   Ensuring win win condition for stake holders in terms of developing, executing and implementing software project.
2. **Environment workflow –**
   Automating the process to coordinate and integrate tools and people with process through workflow which in terms reduces human errors and enables faster development with faster resource allocation and response to the issues. Evolving the maintenance environment for maintaining and updating software.
3. **Requirements workflow –**
   Analyzing the problem space for identifying/understanding the problems and finding a solution. Evolving the requirement artifacts for example use cases, requirements and design documents/specifications which helps describe the function, architecture, and design of software.
4. **Design workflow –**
   Modelling the software is done to express the software design where Software modeling will address the entire software design. Evolving the architecture and design artifacts.
5. **Implementation workflow –**
   In this workflow the implementation of the designs and architectures are done by programming the components. Along with this implantation and deployment artifacts are evolved.
6. **Assessment workflow –**
   Assessing the trends in process. Product quality assessment is carried here by analyzing quality attributes of the product and defect management of the product.
7. **Deployment workflow –**
   In this workflow the process of delivering the end products to the user is carried or preparing the software application/product to run and operate in a specific environment is done.



2 WORKFLOW ITERATIONS

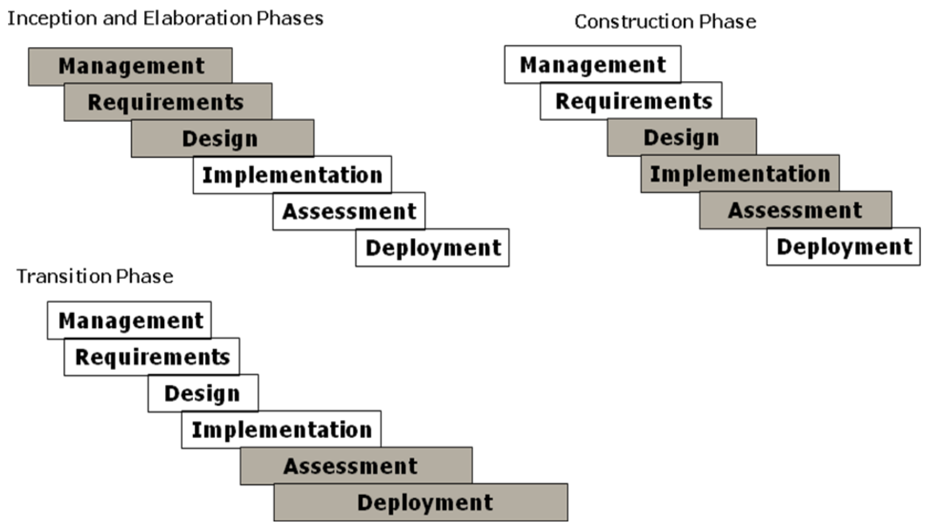
A : Iteration consists of sequential set of activities in various proportions, depending on where the iteration is located in the development cycle. Each iteration is defined in terms of a se t of allocated usage scenarios. The components neededto implement all selected scenarios are developed and integrated with the results of previous iterations. An individual iteration's workflow illustrated in 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, ortasks, 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 environmentcomponents
- Requirements: analyzing the baseline plan, the baseline architecture, and the baseline requirements set artifacts to fully elaborate the use cases to the demonstrated at the end of the iteration and their evaluation criteria.
- Design: Evolving the baseline architecture ad the baseline design set artifacts to elaborate fully the design model and test model components necessary to demonstrate against the evolution criteria allocated to this iteration.


Allocated usage scenarios
Results from the Previous iteration
Management
Requirements
Design
Implementation
Assessment
Deployment
Results for the next iteration

- Implementation: developing any new components, and enhancing or modifying any existing components, to demonstrate the evolution criteria allocated to this iteration
- Assessment: evaluating the results of the iteration, including compliance with the allocated evaluation criteria and the quality of the current baselines; indentifying any rework required and determining whether it should be performed before deployment of this release or allocated to the next release.
- Deployment: transitioning the released either to an external organization or to internal closure by conducting a post mortem so that lessons learned can be captured and reflected in the next iteration.

The following is an example of a simple development life cycle, illustrates the differencebetween iterations and increments. This example also illustrates a typical build sequence from the perspective of an abstract layered architecture.



Iteration emphasis across the life cycle

3 MAJOR , MINOR MILESTONES

A : **MAJOR MILESTONES:**

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

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

The milestones may be conducted as one continuous meeting of all concerned parties or incrementally through mostly on-line review of the various artifacts. There are considerable differences in the levels of ceremony for these events depending on several factors.

The essence of each major milestone is to ensure that the requirements understanding, the life-cycle plans, and the product's form, function, and quality are evolving in balanced levels of detail and to ensure consistency among the various artifacts. The following table summarizes the balance of information across the major milestones.

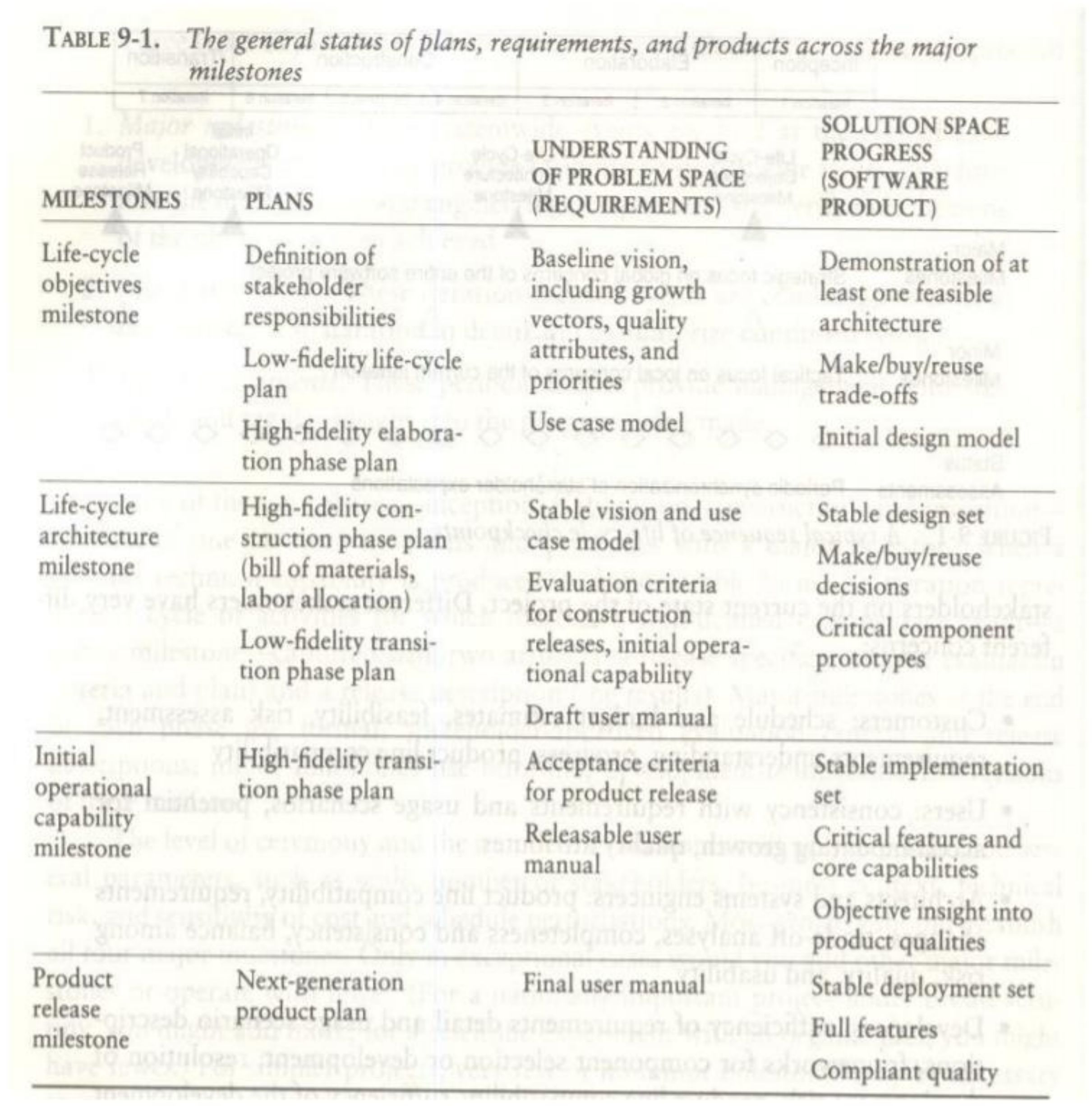


TABLE 9-1. *The general status of plans, requirements, and products across the major milestones*

MILESTONES	PLANS	UNDERSTANDING OF PROBLEM SPACE (REQUIREMENTS)	SOLUTION SPACE PROGRESS (SOFTWARE PRODUCT)
Life-cycle objectives milestone	Definition of stakeholder responsibilities Low-fidelity life-cycle plan High-fidelity elaboration phase plan	Baseline vision, including growth vectors, quality attributes, and priorities Use case model	Demonstration of at least one feasible architecture Make/buy/reuse trade-offs Initial design model
Life-cycle architecture milestone	High-fidelity construction phase plan (bill of materials, labor allocation) Low-fidelity transition phase plan	Stable vision and use case model Evaluation criteria for construction releases, initial operational capability Draft user manual	Stable design set Make/buy/reuse decisions Critical component prototypes
Initial operational capability milestone	High-fidelity transition phase plan	Acceptance criteria for product release Releasable user manual	Stable implementation set Critical features and core capabilities Objective insight into product qualities
Product release milestone	Next-generation product plan	Final user manual	Stable deployment set Full features Compliant quality

MINOR MILESTONES:

All iterations are not created equal. An iteration can take on very different forms and priorities, depending on where the project is in the life cycle. Early iterations focus on analysis and design with substantial elements of discovery, experimentation, and risk assessment. Later iterations focus much more on completeness, consistency, usability, and change management.

- **Iteration readiness review:** this informal milestone is conducted at the start of each iteration to review the detailed iteration plan the evolution criteria that have been allocated to this iteration.

- **Iteration Assessment review:** this informal milestone is conducted at the end of each iteration to assess the degree of which the iteration achieved its objectives and satisfied its evaluation criteria, to review iteration achieved its objectives and satisfied its evaluation criteria, to review iteration results, to review qualification test results, to determine the amount of rework to be done, and to review the impact of the iteration results on the plan for subsequent iterations.

The format and content of these minor milestones tend to be highly dependent on the project and the organizational culture. Figure 9-4 identifies the various minor milestones to be considered when a project is being planned.

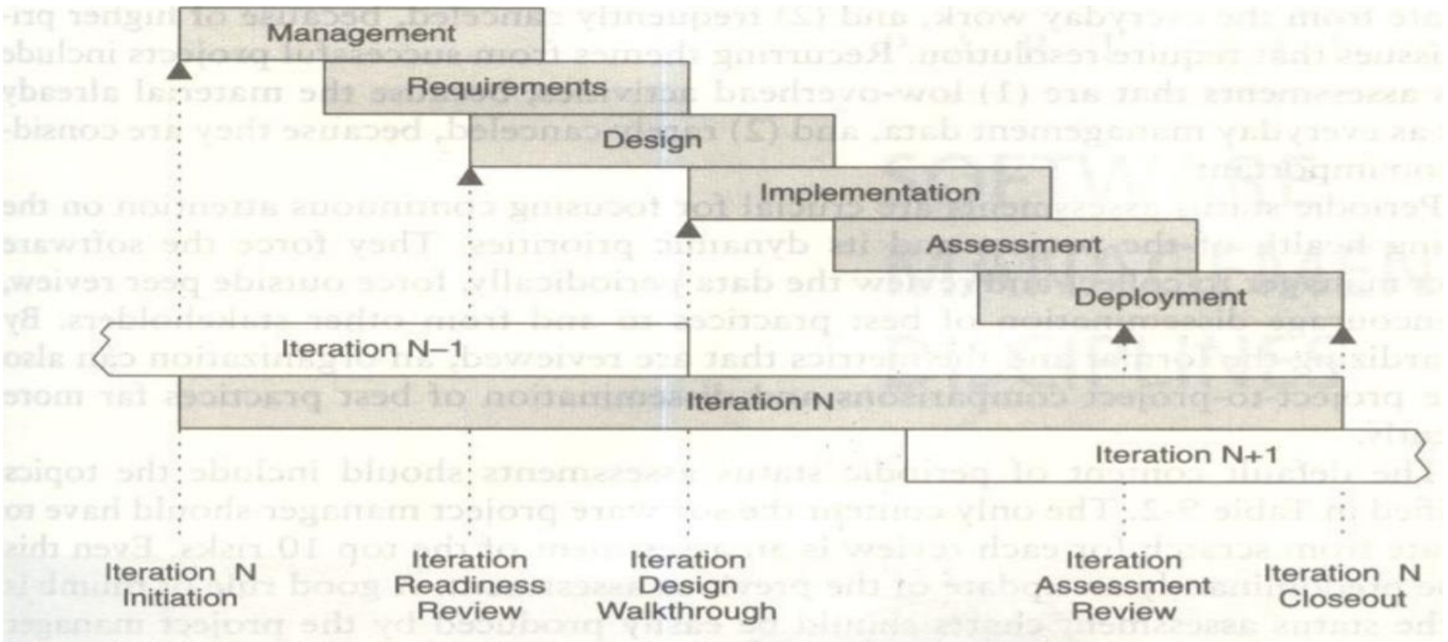


FIGURE 9-4. *Typical minor milestones in the life cycle of an iteration*

4 PERIODIC STATUS ASSESMENT

A: **PERIODIC STATUS ASSESSMENTS:**

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 configurations

• A mechanism for disseminating process, progress, quality trends, practices, and experience information to and from all stakeholders in an open forum

Recurring themes from unsuccessful projects include status assessments that are (1) high-overhead activities, because the work associated with generating the status is separate from the everyday work, and (2) frequently canceled, because of higher priority issues that require resolution. Recurring themes from successful projects include status assessments that are (1) low-overhead activities, because the material already exists as everyday management data, and (2) rarely canceled, because they are considered too important.

Periodic status assessments are crucial for focusing continuous attention on the evolving health of the project and its dynamic priorities. They force the software project manager to collect and review the data periodically, force outside peer review, and encourage dissemination of best practices to and from other stakeholders. By standardizing the format and the metrics that are reviewed, an organization can also enable project-to-project comparisons and dissemination of best practices far more efficiently.

The default content of periodic status assessments should include the topics identified in Table 9-2. The only content the software project manager should have to generate from scratch for each review is an assessment of the top 10 risks. Even this will be predominantly an update of the previous assessment. A good rule of thumb is that the status assessment charts should be easily produced by the project manager with one day's notice. This minimal effort is possible if the data exist within an automated environment.

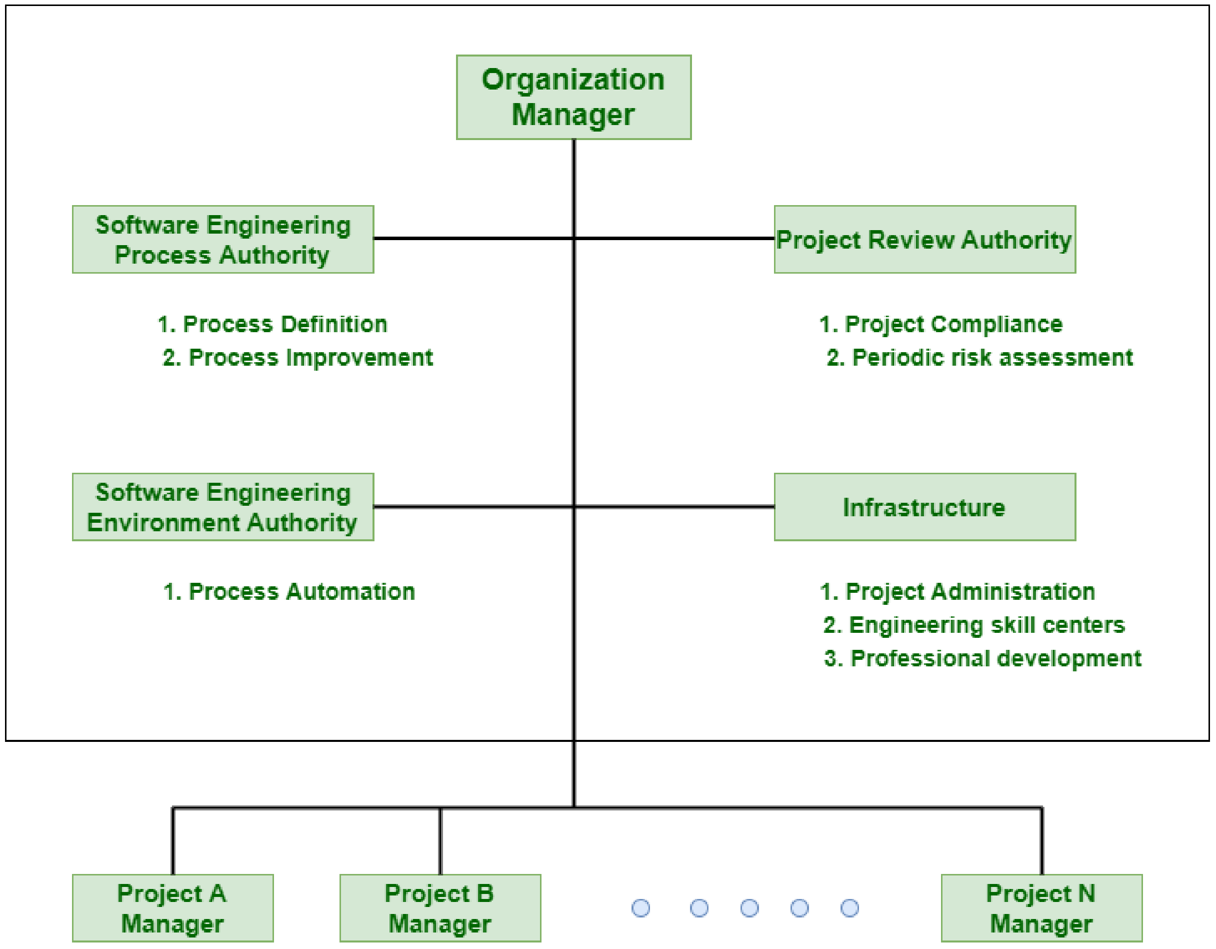
TOPIC	CONTENT
Personnel	Staffing plan Vs. actuals Attritions, additions
Financial trends	Expenditure plan Vs. actuals for the previous, current and next major milestones Revenue forecasts
Top 10 risks	Issues and criticality resolution plans Quantification (cost, time, quality) of exposure
Technical Progress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

UNIT-4

1 LINE OF BUSINESS ORGANIZATION

A: **Line-Of-Business Organizations:**

Below is a diagram is given that shows roles and responsibilities of a default line-of-business organization. Line business organizations need to support projects with infrastructure that are necessary and essential to make use of a common process. Line of business simply a general term that describes and explains products and services simply offered by a business or manufacturer. Software lines of the business are generally motivated and supported by Return of Investment (ROI), new business discriminators, market diversification, and profitability.



Default roles in a Software Line-of-Business Organization

1. **Software Engineering Process Authority (SEPA) –**
   It is team that is responsible for exchanging information and guidance of project both to and from project practitioners. The project practitioners simply perform work and are usually responsible for one or more process activities. SEPA is a very important and essential role or responsibility in any of organizations.

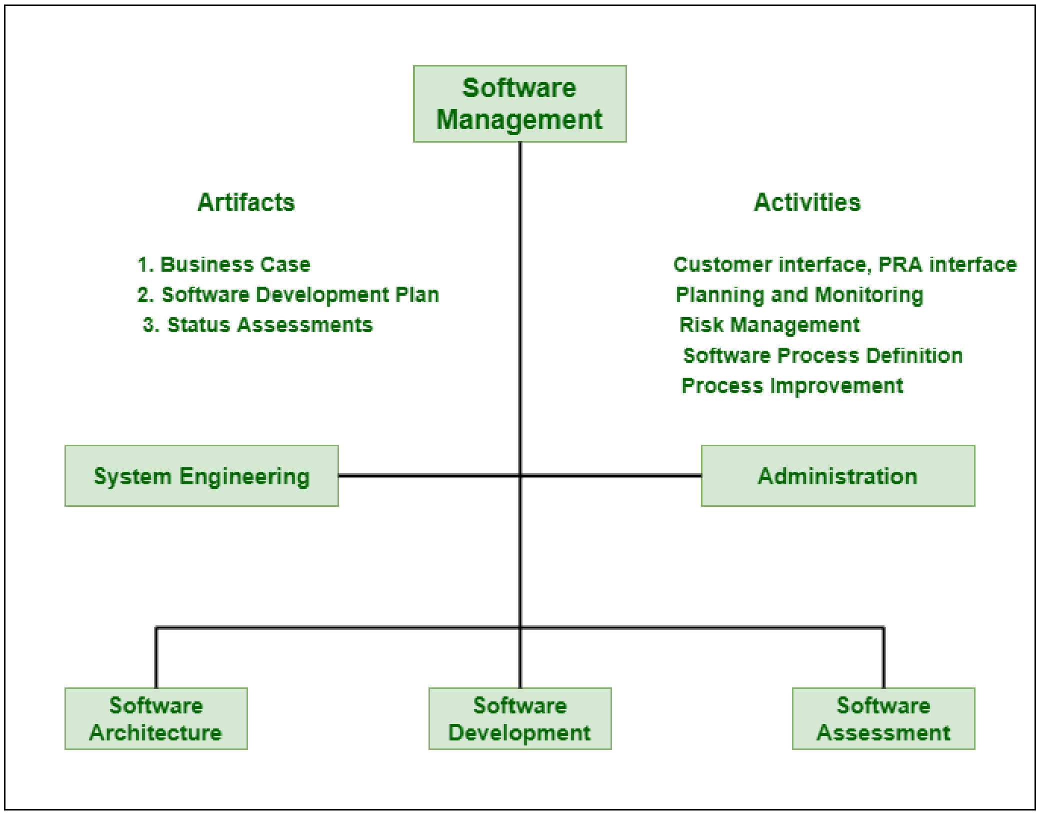
2. **Project Review Authority (PRA) –**
   Project review is simply a scheduled status meeting that is taken on a regular basis. It includes project progress, issues, and risks. It is responsible for project review. The PRA generally reviews both conformance to contractual obligations and organizational policy obligations of project.

3. **Software Engineering Environment Authority (SEEA) –**
   SEEA is a very important role and is very much needed to achieve an ROI for a common process. It is simply responsible for supporting and managing a standard environment. Due to this, different tools, techniques, and training can be effectively amortized across all types of projects.

4. **Infrastructure –**
   Organizational infrastructure generally consists of systems, protocols, and various processes that provide structure to an organization, support human resources, supports organization in carrying out its vision, mission, goals, and values. It can range from trivial to largely entrenched bureaucracies. Various components of organizational infrastructure are Project administration, Engineering skill centers, and professional development.

2 PROJECT ORGANIZTION RESPONSIBILITIES

A : Below diagram given that shows roles and responsibilities of a default project organization. Project organizations generally need to allocate artifacts and responsibilities across project team simply to ensure and confirm a balance of global (architecture) and local (component) concerns.



Default Project Organization and Responsibilities

Teams of Organization :

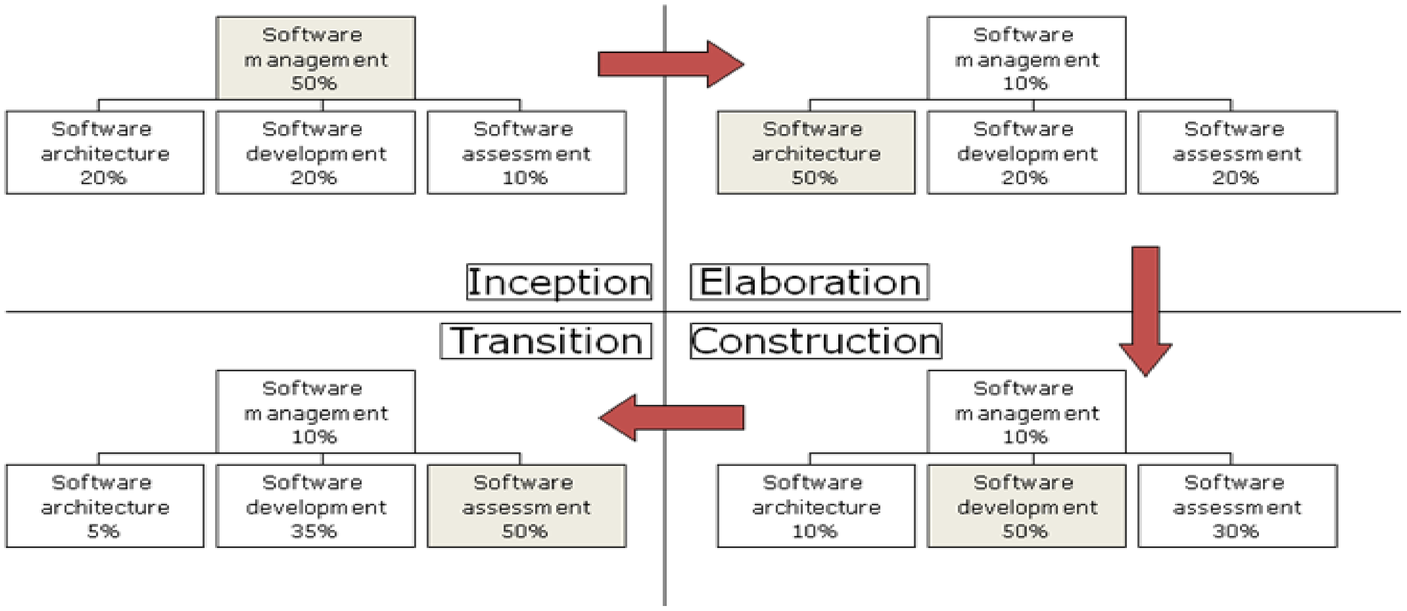
- **Project Management Team –**
  It is an active and highly enthusiastic participant. They are responsible for producing, developing, and managing project.
- **Architecture Team –**
  They are generally responsible for real artifacts and even for integration of components. They also find out risks of product misalignment with requirements of stakeholders and simply ensure that solution fits defined purpose.
- **Development Team –**
  They are responsible for all work that is necessary to produce working and validated assets.
- **Assessment Team –**
  They are responsible for assessing quality of deliverables.

3 EVOLUTION OF ORGANIZATIONS

A : The project organization represents the architecture of the team and needs to evolve consistent with the project plan captured in the work breakdown structure. The following figure illustrates how the team's centre 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.



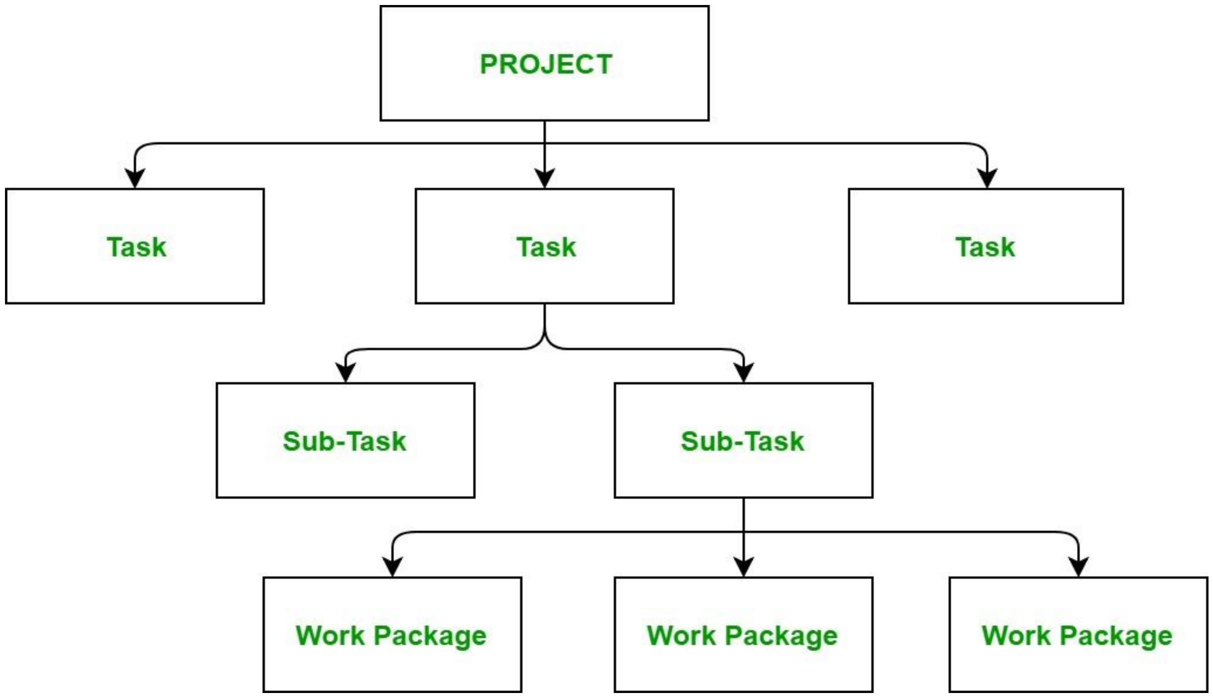
- Elaboration team: an architecture focused organization in which the driving forces of the project reside in the software architecture team and are supported by the software development software assessment teams has necessary to achieve a stable architecture baseline
- Construction team: a fairly balanced organization in which most of the activity resides in the software development and software assessment teams
- Transition team: a customer focus organization in which usage feedback drives the deployment activities.

4 WBS

A : A **Work Breakdown Structure** includes dividing a large and complex project into simpler, manageable and independent tasks. The root of this tree (structure) is labelled by the Project name itself. For constructing a work breakdown structure, each node is recursively decomposed into smaller sub-activities, until at the leaf level, the activities becomes undividable and independent. It follows a Top-Down approach.

Steps:

- **Step-1:** Identify the major activities of the project.
- **Step-2:** Identify the sub-activities of the major activities.
- **Step-3:** Repeat till undividable, simple and independent activities are created.



Construction of Work Breakdown Structure:
Firstly, the project managers and top level management identifies the main deliverables of the project. After this important step, these main deliverables are broke down into smaller higher-level tasks and this complete process is done recursively to produce much smaller independent tasks. It depends on the project manager and team that upto which level of detail they want to break down their project.
Generally the lowest level tasks are the most simplest and independent tasks and takes less than two weeks worth of work. Hence, there is no rule for upto which level we may build the work breakdown structure of the project as it totally depends upon the type of project we are working on and the management of the company. The efficiency and success of the whole project majorly depends on the quality of the Work Breakdown Structure of the project and hence, it implies its importance.

Uses:

- It allows to do a precise cost estimation of each activity.
- It allows to estimate the time that each activity will take more precisely.
- It allows easy management of the project.
- It helps in proper organisation of the project by the top management.

- 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*

- A Management
  - AA Inception phase management
    - AAA Business case development
    - AAB Elaboration phase release specifications
    - AAC Elaboration phase WBS baselining
    - AAD Software development plan
    - AAE Inception phase project control and status assessments
  - AB Elaboration phase management
    - ABA Construction phase release specifications
    - ABB Construction phase WBS baselining
    - ABC Elaboration phase project control and status assessments
  - AC Construction phase management
    - ACA Deployment phase planning
    - ACB Deployment phase WBS baselining
    - ACC Construction phase project control and status assessments
  - AD Transition phase management
    - ADA Next generation planning
    - ADB Transition phase project control and status assessments
- B Environment
  - BA Inception phase environment specification
  - BB Elaboration phase environment baselining
    - BBA Development environment installation and administration
    - BBB Development environment integration and custom toolsmithing
    - BBC SCO database formulation
  - BC Construction phase environment maintenance
    - BCA Development environment installation and administration
    - BCB SCO database maintenance
  - BD Transition phase environment maintenance
    - BDA Development environment maintenance and administration
    - BDB SCO database maintenance
    - BDC Maintenance environment packaging and transition
- C Requirements
  - CA Inception phase requirements development
    - CAA Vision specification
    - CAB Use case modeling
  - CB Elaboration phase requirements baselining
    - CBA Vision baselining
    - CBB Use case model baselining
  - CC Construction phase requirements maintenance
  - CD Transition phase requirements maintenance

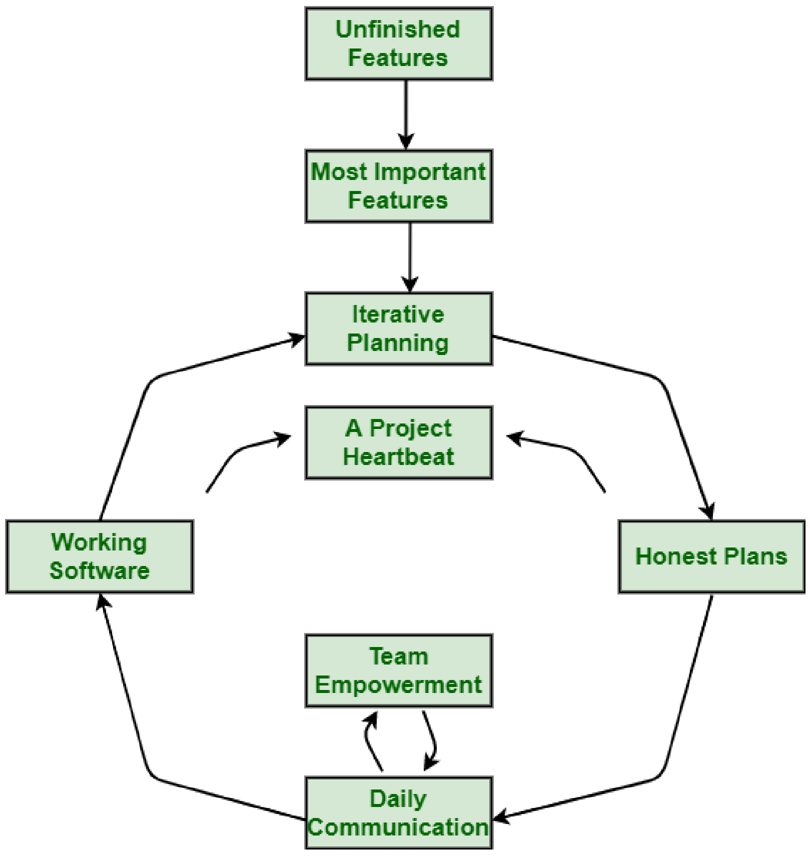
- D Design
  - DA Inception phase architecture prototyping
  - DB Elaboration phase architecture baselining
    - DBA Architecture design modeling
    - DBB Design demonstration planning and conduct
    - DBC Software architecture description
  - DC Construction phase design modeling
    - DCA Architecture design model maintenance
    - DCB Component design modeling
  - DD Transition phase design maintenance
- E Implementation
  - EA Inception phase component prototyping
  - EB Elaboration phase component implementation
    - EBA Critical component coding demonstration integration
  - EC Construction phase component implementation
    - ECA Initial release(s) component coding and stand-alone testing
    - ECB Alpha release component coding and stand-alone testing
    - ECC Beta release component coding and stand-alone testing
    - ECD Component maintenance
  - ED Transition phase component maintenance
- F Assessment
  - FA Inception phase assessment planning
  - FB Elaboration phase assessment
    - FBA Test modeling
    - FBB Architecture test scenario implementation
    - FBC Demonstration assessment and release descriptions
  - FC Construction phase assessment
    - FCA Initial release assessment and release description
    - FCB Alpha release assessment and release description
    - FCC Beta release assessment and release description
  - FD Transition phase assessment
    - FDA Product release assessment and release descriptions
- G Deployment
  - GA Inception phase deployment planning
  - GB Elaboration phase deployment planning
  - GC Construction phase deployment
    - GCA User manual baselining
  - GD Transition phase deployment
    - GDA Product transition to user

FIGURE 10-2. *Default work breakdown structure*

5 ITERATIVE PLANNING PROCESS

A : **Iteration planning** is generally process to adapt as project unfolds by making alterations in plans. Plans are changed simply due to based upon feedback from monitoring process, some changes on project assumptions, risks, and changes in scope, budget, or schedule. It is very essential to include the team in planning process. Basically, planning is generally concerned with explaining and defining and the actual sequence of intermediate results. It is an event where each of team members identifies how much of team backlog, they can commit to delivering during an upcoming iteration.

Iteration planning is generally process of just discussing and planning next cycle, phase, or iteration of software application that is in process of development. An evolutionary developed plan is very essential because there are always adjustments in developed content and schedule as an early conjecture that simply evolves into highly and well-understood project circumstances.



- **Inception Iterations :**
  In some of cases where project includes new product roll-out or simply creation of new technology, iterations might be essential to further explain scope of project, risks, and all benefits. It might also involve a further increase in quality of use-case model, business case, risk list, architectural proof-of-concept, or even project and iteration plans. Extension of this inception phase can also be advised in some of cases where both risk and required investment are high. It can also be advised where problem domain is new or team is not experienced.
  The early prototyping technique also integrates foundation components of architecture of candidate and also provide an executable framework for explaining and simply elaborating use cases of system. To achieve prototype that is acceptable, Large-scale, and custom developments

are two iterations. But its better for various projects to be able to get by use of only one. Inception iteration is generally responsible to establish scope and vision and to explain and define business case.

- **Elaboration Iterations :**
  In each of iteration, supporting environment is refined further. If initial elaboration is only focused on preparation of environment for analysis and design and implementation after then second iteration might focus on preparation of test environment. The preparation of test environment basically includes configuring test processes, writing development case part, and also preparing or generating templates and guidelines that are needed to be followed for test and setting up test tools.
  Elaboration iteration also results in architecture with complete framework and infrastructure for execution. To achieve an architecture baseline that is acceptable, some of projects require two iterations. Some additional and extra iterations may be required by unprecedented architectures. On other hand, projects that are developed on highly and well-established architecture framework can be get by use of single iteration. During elaboration iteration, requirements are defined fully and architectures are well-established.

- **Construction Iterations :**
  The essential result of late iteration in construction phase is that more functionality is added, which yields an increasing complete system. During construction iteration, use cases are generally realized and architectures are fleshed-out. Some of projects need and require two construction iteration. First is an alpha release that includes executable capability for each of critical use cases. Second is a beta release that provides 95 % of total product capability breadth and also achieves some of essential quality attributes that are needed and important.

- **Transition Iterations :**
  Transition iteration is generally responsible to migrate product into user community. Several projects use only one iteration to transition beta release into end product or final product. Even these projects learn to live with only one iteration among beta release and end product or final product release.

UNIT-5

1: 7 CORE METRICS

A :

METRIC	PURPOSE	PERSPECTIVES
Work and progress	Iteration planning, plan vs actuals, management indicator	SLOC, function points, object points, scenarios, test cases, SCOs
Budget 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	Software changes
Breakage and adoptability	Convergence, software scrap, quality indicator	Reworked SLOC per change, by type, by release/component/subsystem
Rework and adoptability	Convergence, software rework, quality indicator	Average hours per change, by type, by release/component/subsystem
MTBF and Maturity	Test coverage/adequacy, Robustness for use, Quality indicator	Failure counters, test hours until failure, by release/component/subsystem

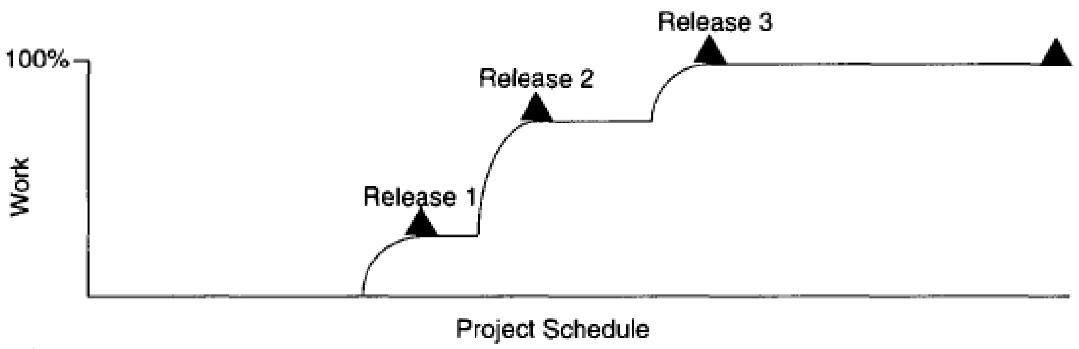
2 QUALITY AND MANAGEMENT INDICATORS

A:

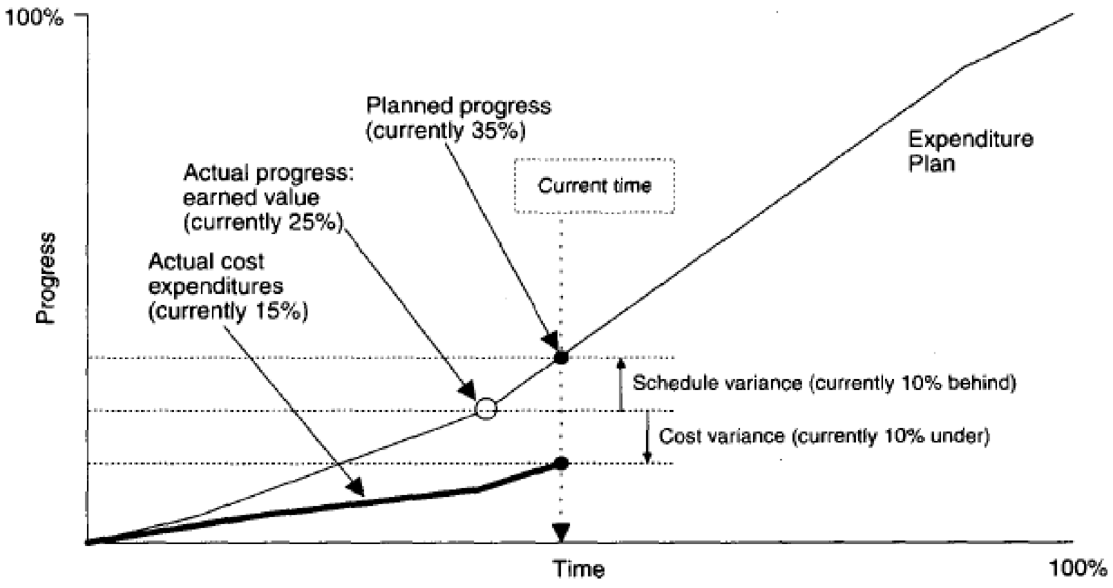
Management Indicators:

Metrics that are given below must be interpreted to serve as indicators. They might indicate problem or issue, minor perturbation, or simply change that is generally being managed. Metrics are given below :

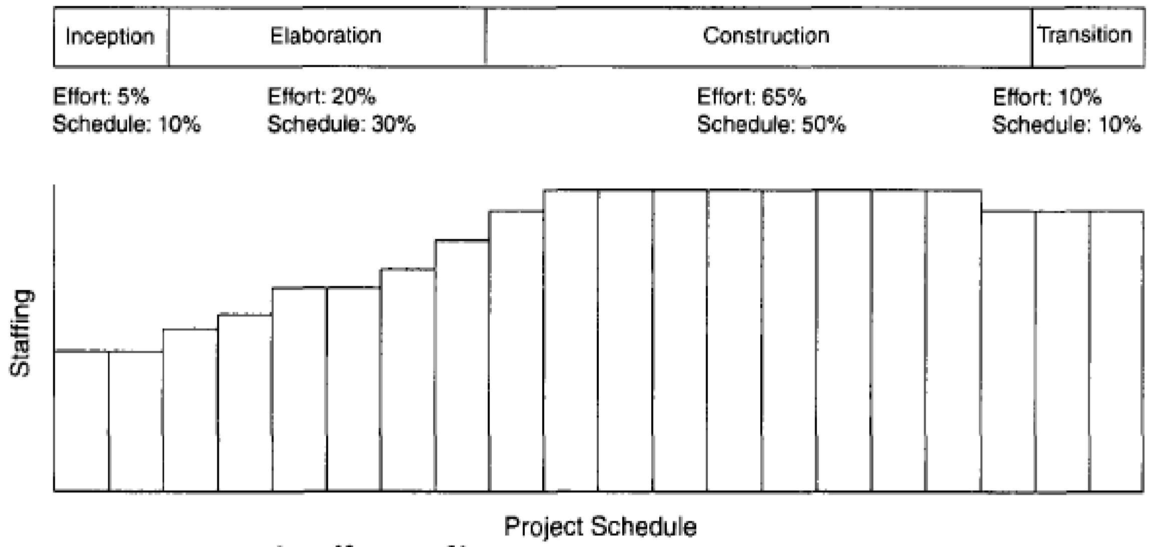
- **Work and progress –**
  Work that is performed over given period of time. Its planning about iteration that means to determine and discuss planning next cycle, phase, or iteration.



-
- **Budgeted cost and expenditure –**
  Cost that is incurred over given period of time. Its Financial insight that means to understand implications regarding financial decisions that are made today.



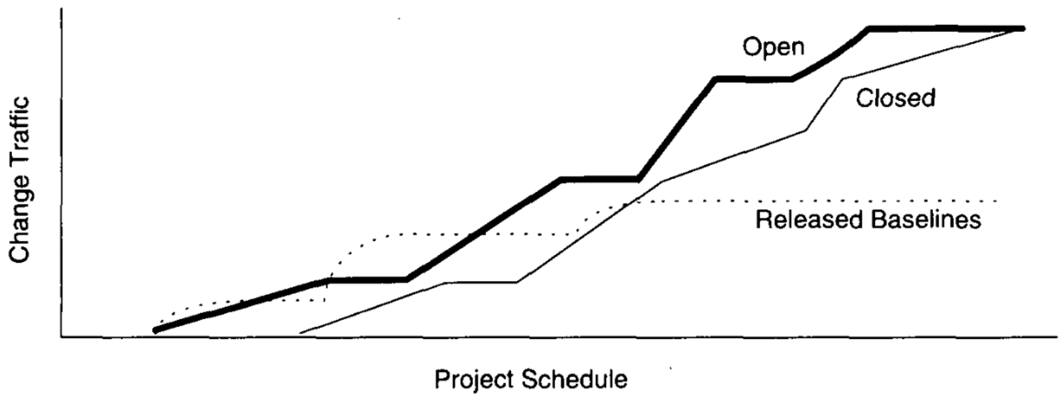
-
- **Staffing team dynamics –**
  Personnel changes occurred over given period of time. Its resource plan that means to allocate and utilize resources simply to achieve highest efficiency of these resources.



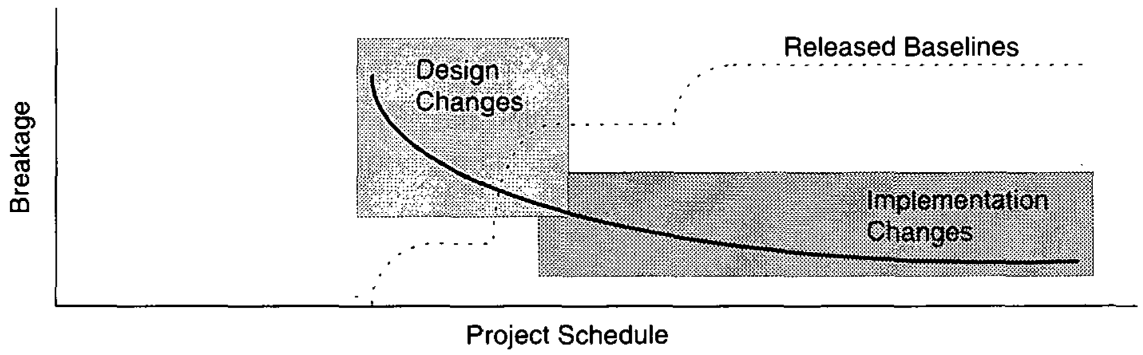
-

 Quality Indicators :
Quality indicators are Key Performance Indicators (KPI) that are very critical during realization of project. These indicators are needed to be monitored carefully in order to confirm and ensure that team is working or performing on proper tasks. Metrics are given below :

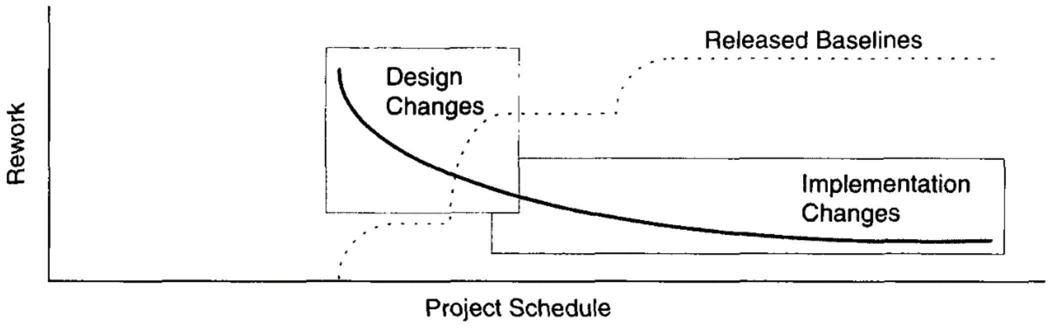
- **Change traffic and stability –**
  Change of traffic and stability over given period of time. Its planning about iteration that means to determine and discuss planning next cycle, phase, or iteration. They also indicate convergence schedule that means to indicate convergence points in schedule where two to more activities come together and explain dependencies of successor activity.



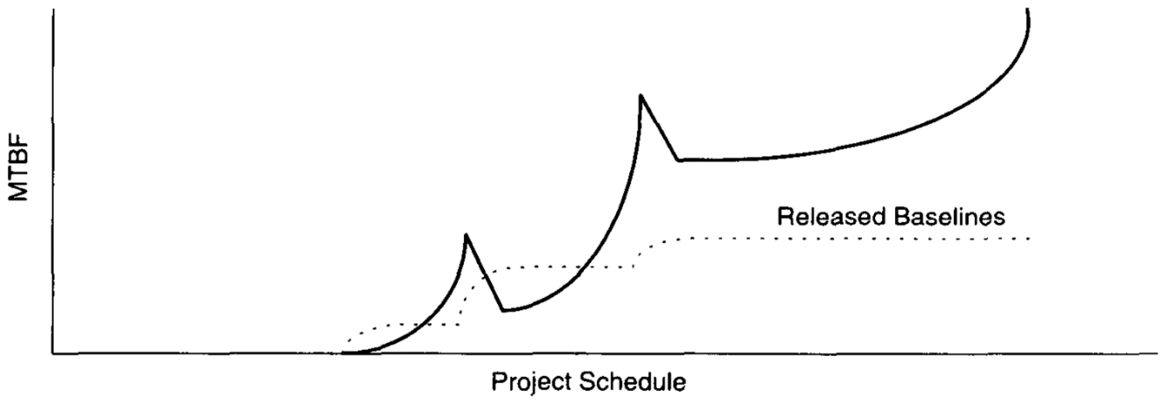
-
- **Breakage and modularity –**
  Average breakage and modularity per change over given period of time. Their is convergence, software scrap and to indicate quality of software.



-
- **Rework and adaptability –**
  Average rework per change over given period of time. Their is convergence, software scrap and to indicate quality of software.



-
- **Meantime between failure (MTBF) and maturity –**
  Rate of defect oven given period of time. Their is to indicate quality of software, test coverage simply to measure and calculate amount of testing performed by set of tests.



-

3 PRAGMATIC SOFTWARE METRICS

A : 1. It is considered meaningful by the customer, manager, and performer. If any one of these stakeholders does not see the metric as meaningful, it will not be used. "The customer is always right" is a sales motto, not an engineering tenet. Customers come to software engineering providers because the providers are more expert than they are at developing and managing software. Customers will accept metrics that are demonstrated to be meaningful to the developer.

2. It demonstrates quantifiable correlation between process perturbations and business performance. The only real organizational goals and objectives are financial: cost reduction, revenue increase, and margin increase.

3. It is objective and unambiguously defined. Objectivity should translate into some form of numeric representation (such as numbers, percentages, ratios) as opposed to textual representations (such as excellent, good, fair, poor). Ambiguity is minimized through well-understood units of measurement (such as staff-month, SLOC, change, function point, class, scenario, requirement), which are surprisingly hard to define precisely in the software engineering world.

4. It displays trends. This is an important characteristic. Understanding the change in a metric's value with respect to time, subsequent projects, subsequent releases, and so forth is an extremely important perspective, especially for today's iterative development models. It is very rare that a given metric drives the appropriate action directly. More typically, a metric presents a perspective. It is up to the decision authority (manager, team, or other information processing entity) to interpret the metric and decide what action is necessary.

5. It is a natural by-product of the process. The metric does not introduce new artifacts or overhead activities; it is derived directly from the mainstream engineering and management workflows.

6. It is supported by automation. Experience has demonstrated that the most successful metrics are those that are collected and reported by automated tools, in part because software tools require rigorous definitions of the data they process.

When metrics expose a problem, it is important to get underneath all the symptoms and diagnose it. Metrics usually display effects; the causes require synthesis of multiple perspectives and reasoning.

For example, reasoning is still required to interpret the following situations correctly:

• A low number of change requests to a software baseline may mean that the software is mature and error-free, or it may mean that the test team is on vacation.

• A software change order that has been open for a long time may mean that the problem was simple to diagnose and the solution required substantial rework, or it may mean that a problem was very time-consuming to diagnose and the solution required a simple change to a single line of code.

• A large increase in personnel in a given month may cause progress to increase proportionally if they are trained people who are productive from the outset. It may cause progress to decelerate if they are untrained new hires who demand extensive support from productive people to get up to speed.

4 PROCESS DISCRIMINATORS

A : **Introduction:-**In tailoring the management process to a specific domain or project, there are two dimensions of discriminating factors: technical complexity and management complexity. A process framework is not a project-specific process implementation with a well-defined recipe for success. The process framework must be configured to the specific characteristics of the project. The process discriminants are organized around six process parameters - scale, stakeholder cohesion, process flexibility, process maturity, architectural risk and domain experience.

1. Scale:the scale of the project is the team size, which drives the process configuration more than any other factor. There are many ways to measure scale, including number of sources lines of code, number of function points, number of use cases and number of dollars. The primary measure of scale is the size of the team. Five people are an optimal size for an engineering team.

A team consisting of 1 member is said to be trivial, a team of 5 is said to be small, a team of 25 is said to be moderate, a team of 125 is said to be large, a team of 625 is said to be huge and so on. As team size grows, a new level of personnel management is introduced at each factor of 5.

Trivial - sized projects require almost no management overhead. Only little documentation is required. Workflow is single-threaded. Performance is dependent on personnel skills.

Small projects consisting of 5 people require very little management overhead. Project milestones are easily planned, informally conducted and changed. There are a small number of individual workflows. Performance depends primarily on personnel skills. Process maturity is unimportant.

Moderate-sized projects consisting of 25 people require very moderate management overhead. Project milestones are formally planned and conducted. There are a small number of concurrent team workflows, each team consisting of multiple individual workflows. Performance is highly dependent on the skills of key personnel. Process maturity is valuable.

Large projects consisting of 125 people require substantial management overhead. Project milestones are formally planned and conducted. A large number of concurrent team workflows are necessary, each with multiple individual workflows. Performance is highly dependent on the skills of the key personnel. Process maturity is necessary.

Huge projects consisting of 625 people require substantial management overhead. Project milestones are very formally planned and conducted. There are a very large number of concurrent team workflows, each with multiple individual workflows. Performance is highly dependent on the skills of the key personnel. Project performance is still dependent on average people.

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

3. Process Flexibility or Rigor: The implementation of the project's process depends on the degree of rigor, formality and change freedom evolved from projects contract (vision document, business case and development plan). For very loose contracts such as building a commercial product within a business unit of a software company, management complexity is minimal. For a very rigorous contract, it could take many months to authorize a change in a release schedule.

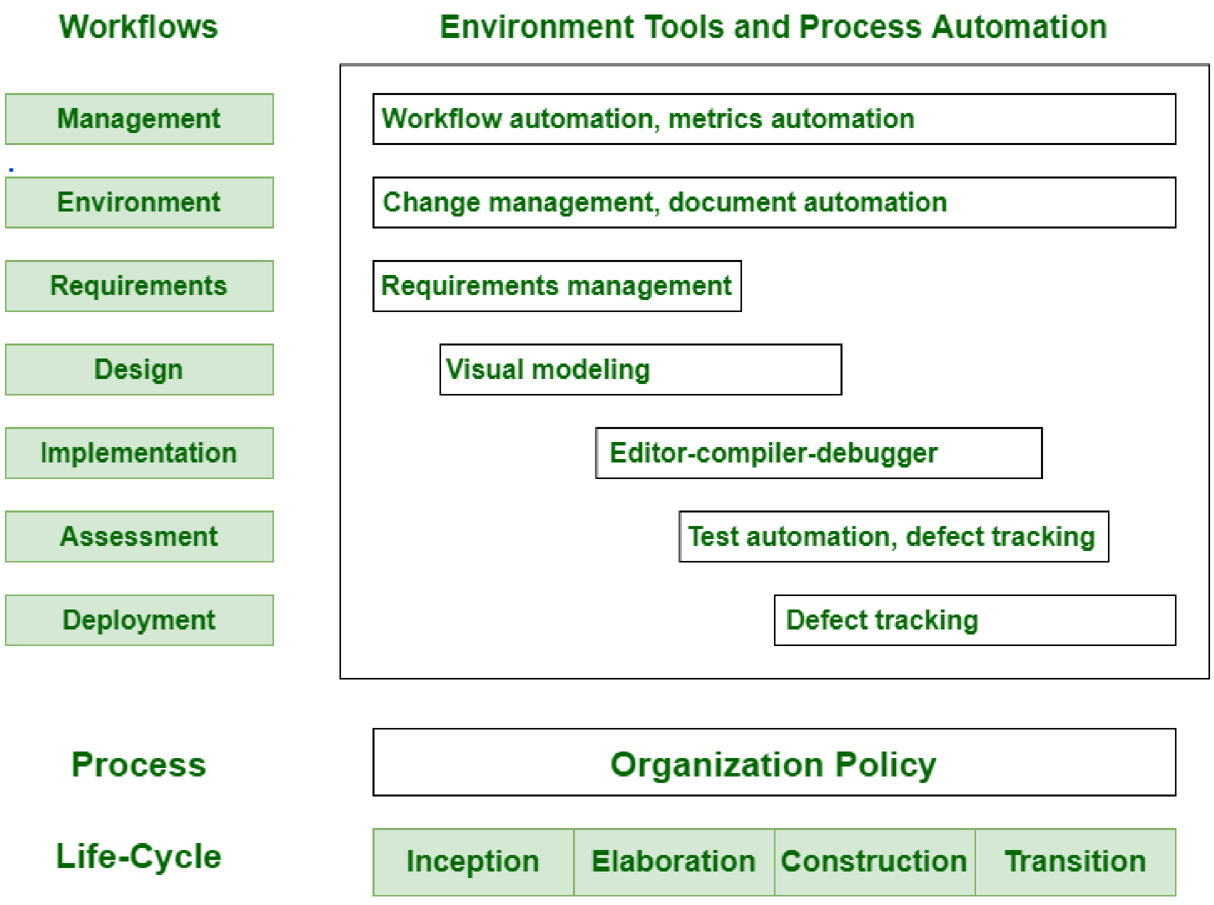
4. Process Maturity: The process maturity level of the development organization is the key driver of management complexity. Managing a mature process is very simpler than managing an immature process. Organization with a mature process have a high level of precedent experience in developing software 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, and trained personnel, planning metrics, artifact templates and workflow templates.

5. Architectural Risk: The degree of technical feasibility is an important dimension of defining a specific projects process. There are many sources of architecture risk. They are (1) system performance which includes resource utilization, response time, throughout and accuracy, (2) robustness to change which includes addition of new features & incorporation of new technology and (3) system reliability which includes predictable behavior and fault tolerance.

6. Domain Experience: The development organization's domain experience governs its ability to converge on an acceptable architecture in a minimum no of iterations.

5 AUTOMATION TOOLS

A : Process automation generally refers to use of digital technology simply to work and perform a process or processes. This is done to accomplish or complete workflow or function. To an iterative process, process automation and change management is a very critical one. Even if change will be too expensive, then development will resist and won't allow it. To automate process of software development, various tools are available nowadays.



Automation and Tool components that support the process workflows

In diagram given above, some of important tools are included and introduced that are very much needed across overall software process and correlates very well to process framework. Each of tools of software development map closely to one of process workflows and each of these process workflows have a distinct for automation support. Workflow automation generally makes complicated software process in an easy way to manage. Here you will see environment that is necessary to support process framework.

Some of concerns are associated with each workflow are given below :

1. **Management :**
   Nowadays, there are several opportunities and chances available for automation of project planning and control activities of management workflow. For creating planning artifacts, several tools are useful such as software cost estimation tools and Work Breakdown Structure (WBS)

tools. Workflow management software is an advanced platform that provides flexible tools to improve way you work in an efficient manner. Thus, automation support can also improve insight into metrics.

2. **Environment :**
   Automating development process and also developing an infrastructure for supporting different project workflows are very essential activities of engineering stage of life-cycle. environment that generally gives and provides process automation is a tangible artifact that is generally very critical to life-cycle of system being developed. Even, top-level WBS recognizes environment like a first-class workflow. Integrating their own environment and infrastructure for software development is one of main tasks for most of software organizations.
3. **Requirements :**
   Requirements management is a very systematic approach for identifying, documenting, organizing, and tracking changing requirements of a system. It is also responsible for establishing and maintaining agreement between user or customer and project team on changing requirements of system. If process wants strong traceability among requirements and design, then architecture is very much likely to evolve in a way that it optimizes requirements traceability other than design integrity. This effect is even more and highly effective and pronounced if tools are used for process automation. For effective requirement management, points that must include are maintaining a clear statement of requirements with attributes for every type of requirement and traceability to other requirements and other project artifacts.
4. **Design :**
   Workflow design is actually a visual depiction of each step that is involved in a workflow from start to end. It generally lays out each and every task sequentially and provides complete clarity into how data moves from one task to another one. Workflow design tools simply allow us to depict different tasks involved graphically as well as also depict performers, timelines, data, and other aspects that are crucial to execution. Visual modeling is primary support that is required and essential for design workflow. Visual model is generally used for capturing design models, representing them in a human-readable format, and also translating them into source code.
5. **Implementation :**
   The main focus and purpose of implementation workflow are to write and initially test software, relies primarily on programming environment (editor, compiler, debugger, etc.). But on other hand, it should also include substantial integration along with change management tools, visual modeling tools, and test automation tools. This is simply required to just support iteration to be productive. It is main focus of Construction phase. The implementation simply means to transform a design model into executable one.
6. **Assessment and Deployment :**
   Workflow assessment is initial step to identifying outdated software processes and just replace them with most effective process. This generally combines domain expertise, qualitative and quantitative information gathering and collection, proprietary tools, and much more. It requires and needs every tool discussed along with some additional capabilities simply to support test automation and test management. Defect tracking is also a tool that supports assessment.

6 The project environment

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

1. The prototyping environment includes an architecture testbed for prototyping project architectures to evaluate trade-offs during the inception and elaboration phases of the life cycle. This informal configuration of tools should be capable of supporting the following activities:

• Performance trade-offs and technical risk analyses

• Make/buy trade-offs and feasibility studies for commercial products

• Fault tolerance/dynamic reconfiguration trade-offs

• Analysis of the risks associated with transitioning to full-scale implementation

• Development of test scenarios, tools, and instrumentation suitable for analyzing the requirements

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

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