UNIT I

Software Process Maturity

Software maturity Framework**,** Principles of Software Process Change, Software Process Assessment, The Initial Process, The Repeatable Process, The Defined Process, The Managed Process, The Optimizing Process.

Process Reference Models

Capability Maturity Model (CMM)**,** CMMI, PCMM, PSP, TSP.

Part-1 software Process Maturity

Software Maturity Framework

The CMM focuses on the capability of software organizations to produce high- quality products consistently and predictably. Software process capability is the inherent ability of a software process to produce planned results.

- **DEFINITION (Process)** A sequence of steps performed for a given purpose. The process integrates people, tools, and procedures.
- **DEFINTION (Software Process)** A set of activities, methods, practices, and transformations that people employ to develop and maintain software and the associated products (documents, etc.)
- **DEFINTION (Software Process Capability)** describes the range of expected results that can be achieved by following a software process.
- **DEFINITION (Software Process Performance)** the actual results achieved by following a software process.
- **DEFINTION (Software Process Maturity)** the extent to which a specific process is explicitly defined, managed, measured, controlled, and effective. As a software organization matures, it needs an infrastructure and culture to support its methods, practices, and procedures so that they endure after those who originally defined them have gone.
- **DEFINTION (Institutionalization)** is the building of infrastructure and culture to support methods, practices, and procedures so that they are the ongoing way of doing business.

Software Process Maturity Framework Five Maturity Levels:

- **Initial:** The software process is characterized by ad hoc, and occasionally even chaotic. Few processes are defined, and success depends on individual effort and heroics.
- **Repeatable:** Basic project management processes are established to track cost, schedule, and functionality. The necessary process discipline is in place to repeat earlier successes on projects with similar applications.
- **Defined:** The software process for both management and engineering activities is documented, standardized, and integrated into a standard software process for the organization. All projects use an approved, tailored version of the organization's standard software process for developing and maintaining software.

Managed: Detailed measures of the software process and product quality are collected. Both the

software process and products are quantitative understood and controlled.

- **Optimizing:** Continuous process improvement is enabled by quantitative feedback from the process and from piloting innovative ideas and technologies.

Basic Principles

- Automation of a poorly defined process will produce automation of poorly defined results
- Improvement should be made in some steps
- Educate/Train, Educate/Train, , Educate/Train

Principles of Software Process Change Software process management has 2 key areas

- People
- Design methods

People: A good mix of talent is required. The best people are always in short supply. You probably have about the best team you can get right now. With proper leadership, education, training and support, most people can do better work than they are currently doing.

Design: When domain knowledge is combined with the ability to produce a good design, a quality product will result.

Six Basic Principles of Software Process Change

1. **Major changes to the software process must start at the top**
   - Major changes requires leadership. Managers must provide good leadership, even though they may not do the work; they must set priorities: furnish resources and provide continuing support.
2. **Ultimately, everyone must be involved.**
   - With an immature software process, software professionals are forced to improvise solutions. in a mature process, these individual actions are more structured, efficient and reinforcing. People are the most important aspect. It's necessary to focus on repairing the process and not the people.
3. **Effective changes require the team to have common goals and knowledge of the current process.**
   - An effective change program requires a reasonable understanding of the current status. An assessment is an effective way to gain this understanding. Software professionals generally need most help in controlling requirements., coordinating changes, making plans. managing interdependencies and coping with system design issues.

4. **Change is continuous**

   - One of the most difficult things for a management team to recognize is that human interactive processes are never static. Both problems and people are in constant flux, and this fluidity all for periodic adjustment of tasks and relationships. In dealing with these dynamics, 3 points are important.

   - Relative changes generally make things worse

   -Every defect is an improvement opportunity

   -Crisis prevention is more important than crisis recovery

5. **Software process changes will not be retained without conscious effort and periodic reinforcements**

   - Precise and accurate work is hard. Its rarely sustained for long without reinforcement.

   Human adoption of new process methods involves 4 stages

   - Installation, Practice , Proficiency , Naturalness

6. **Software process improvement requires investment**

   - While the need for dedicating resources to improvement seems self- evident it's surprising how often managers rely on exhorting their people to try harder

   - To improve the aft rare process someone must work on it

   - Unplanned process improvement is wishful thinking.

A software process assessment is a disciplined examination of the software processes used by an organization, based on a process model. The assessment includes the identification and characterization of current practices, identifying areas of strengths and weaknesses, and the ability of current practices to control or avoid significant causes of poor (software) quality, cost, and schedule.

A software assessment (or audit) can be of three types.

- A **self-assessment (first-party assessment)** is performed internally by an organization's own personnel.
- A **second-party assessment** is performed by an external assessment team or the organization is assessed by a customer.
- A **third-party assessment** is performed by an external party or (e.g., a supplier being assessed by a third party to verify its ability to enter contracts with a customer).

Software process assessments are performed in an open and collaborative environment. They are for the use of the organization to improve its software

processes, and the results are confidential to the organization. The organization being assessed must have members on the assessment team.

Software Process Maturity Assessment

The scope of a software process assessment can cover all the processes in the organization, a selected subset of the software processes, or a specific project. Most of the standard-based process assessment approaches are invariably based on the concept of process maturity.

When the assessment target is the organization, the results of a process assessment may differ, even on successive applications of the same method. There are two reasons for the different results. They are,

- The organization being investigated must be determined. For a large company, several definitions of organization are possible and therefore the actual scope of appraisal may differ in successive assessments.
- Even in what appears to be the same organization, the sample of projects selected to represent the organization may affect the scope and outcome.

When the target unit of assessment is at the project level, the assessment should include all meaningful factors that contribute to the success or failure of the project. It should not be limited by established dimensions of a given process maturity model. Here the degree of implementation and their effectiveness as substantiated by project data are assessed.

Process maturity becomes relevant when an organization intends to embark on an overall long-term improvement strategy. Software project assessments should be independent assessments in order to be objective.

Software Process Assessment Cycle

According to Paulk and colleagues (1995), the CMM-based assessment approach uses a six-step cycle. They are −

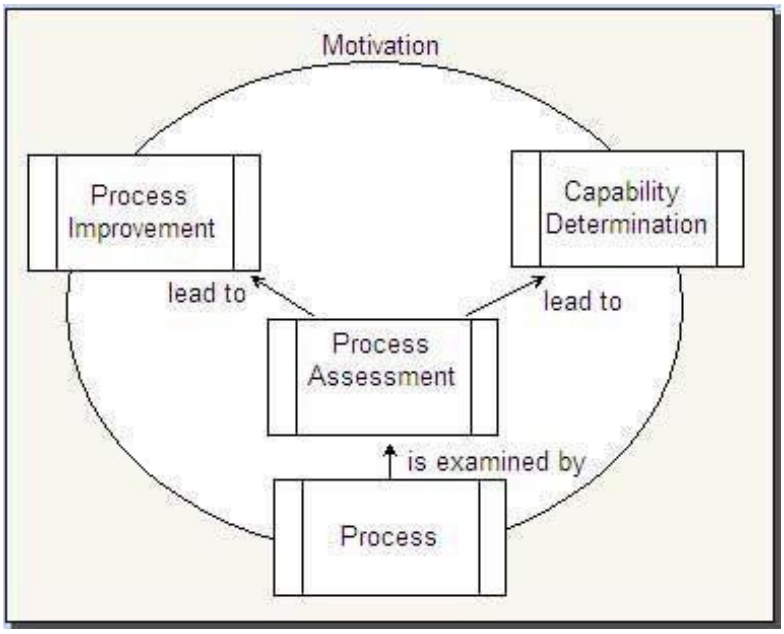
- Select a team - The members of the team should be professionals knowledgeable in software engineering and management.
- The representatives of the site to be appraised complete the standard process maturity questionnaire.

- The assessment team performs an analysis of the questionnaire responses and identifies the areas that warrant further exploration according to the CMM key process areas.
- The assessment team conducts a site visit to gain an understanding of the software process followed by the site.
- The assessment team produces a list of findings that identifies the strengths and weakness of the organization's software process.
- The assessment team prepares a Key Process Area (KPA) profile analysis and presents the results to the appropriate audience.

For example, the assessment team must be led by an authorized SEI Lead Assessor. The team must consist of between four to ten team members. At least, one team member must be from the organization being assessed, and all team members must complete the SEI's Introduction to the CMM course (or its equivalent) and the SEI's CBA IPI team training course. Team members must also meet some selection guidelines.

With regard to data collection, the CBA IPI relies on four methods −

- The standard maturity questionnaire
- Individual and group interviews
- Document reviews
- Feedback from the review of the draft findings with the assessment participants



SCAMPI

The Standard CMMI Assessment Method for Process Improvement (SCAMPI) was developed to satisfy the CMMI model requirements (Software Engineering Institute, 2000). It is also based on the CBA IPI. Both the CBA IPI and the SCAMPI consist of three phases −

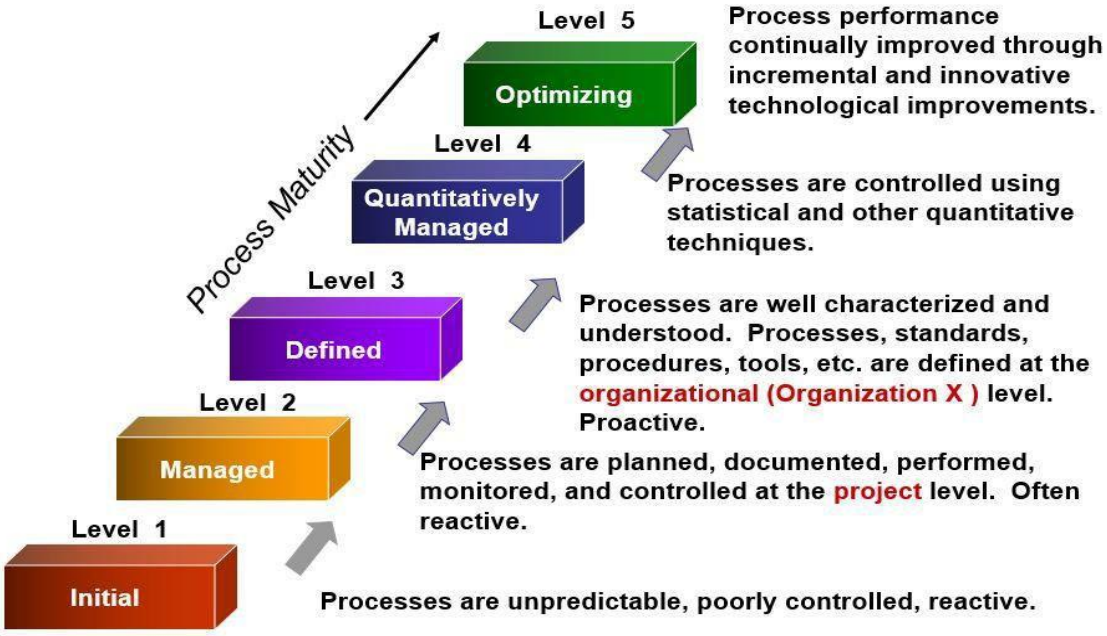
- Plan and preparation
- Conduct the assessment onsite
- Report results

The activities for the plan and preparation phase include −

- Identify the assessment scope
- Develop the assessment plan
- Prepare and train the assessment team
- Make a brief assessment of participants
- Administer the CMMI Appraisal Questionnaire
- Examine the questionnaire responses
- Conduct an initial document review

The activities for the onsite assessment phase include −

- Conduct an opening meeting
- Conduct interviews
- Consolidate information
- Prepare the presentation of draft findings
- Present the draft findings
- Consolidate, rate, and prepare the final findings



Part -2 Process Reference Models

Capability Maturity Model(CMM)

What is CMM?

Capability Maturity Model is used as a benchmark to measure the maturity of an organization's software process.

CMM was developed at the Software engineering institute in the late 80's. It was developed as a result of a study financed by the U.S Air Force as a way to evaluate the work of subcontractors.

CMM was first introduced in late 80's in U.S Air Force to evaluate the work of subcontractors. Later on, with improved version, it was implemented to track the quality of the software development system.

The entire CMM level is divided into five levels.

- **Level 1** (Initial): Where requirements for the system are usually uncertain, misunderstood and uncontrolled. The process is usually chaotic and ad-hoc.
- **Level 2** (Managed): Estimate project cost, schedule, and functionality. Software standards are defined
- **Level 3** (Defined): Makes sure that product meets the requirements and intended use
- **Level 4** (Quantitatively Managed): Manages the project's processes and sub- processes statistically
- **Level 5** (Maturity): Identify and deploy new tools and process improvements to meet needs and business objectives

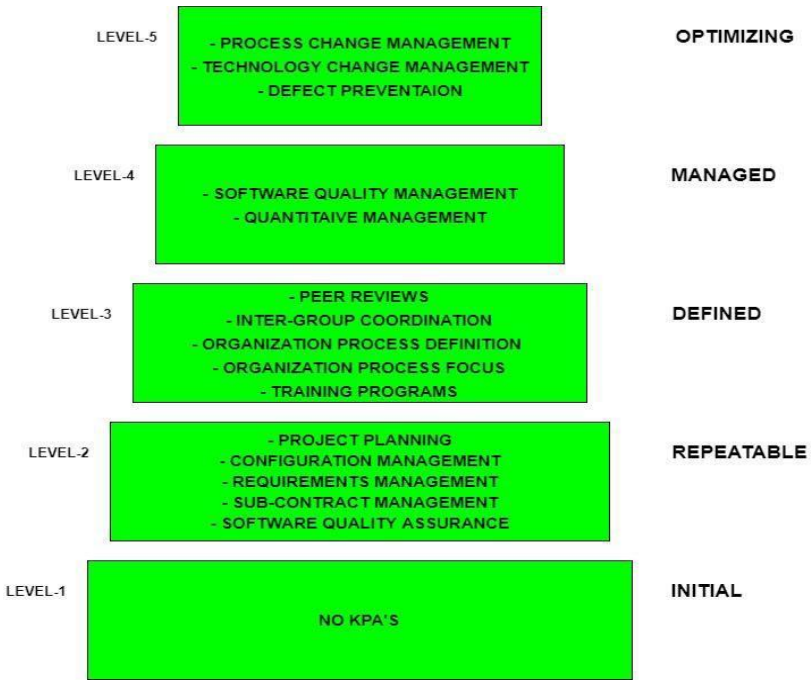
Limitations of CMM Models

- CMM determines what a process should address instead of how it should be implemented
- It does not explain every possibility of software process improvement
- It concentrates on software issues but does not consider strategic business planning, adopting technologies, establishing product line and managing human resources
- It does not tell on what kind of business an organization should be in
- CMM will not be useful in the project having a crisis right now

Why Use CMM?

Today CMM act as a "seal of approval" in the software industry. It helps in various ways to improve the software quality.

- It guides towards repeatable standard process and hence reduce the learning time on how to get things done
- Practicing CMM means practicing standard protocol for development, which means it not only helps the team to save time but also gives a clear view of what to do and what to expect
- The quality activities gel well with the project rather than thought of as a separate event
- It acts as a commuter between the project and the team
- CMM efforts are always towards the improvement of the process



Capability maturity model integration (CMMI) is an approach or methodology for improving and refining the software development process within an organization. It is based on a process model or a structured collection of practices.

CMMI is used to guide the improvement process across a project, division or even an entire organizational structure. It also allows companies to integrate organizational functions that are traditionally separate, set goals for process improvements and priorities, provide guidance for quality processes, and act as a point of reference for appraising processes.

Difference between CMM and CMMI

1. CMM came first but was later improved and was succeeded by CMMI.
2. Different sets of CMMS have problems with overlaps, contradictions, and lack of standardization. CMMI later addressed these problems.
3. Initially, CMM describes specifically about software engineering whereas CMMI describes integrated processes and disciplines as it applies both to software and systems engineering.
4. CMMI is much more useful and universal than the older CMM.

People Capability Maturity Model (PCMM)

The **People Capability Maturity Model** consists of five maturity levels. Each maturity level is an evolutionary plateau at which one or more domains of the organization's processes are transformed to achieve a new level of organizational capability. The five levels of **People CMM** are defined as follows:

1. At **PCMM Level 1**, an organization has no consistent way of performing workforce practices. Most workforce practices are applied without analysis of impact.
2. At **PCMM Level 2**, organizations establish a foundation on which they deploy common workforce practices across the organization. The goal of Level 2 is to have managers take responsibility for managing and developing their people. For example, the first benefit an organization experiences as it achieves Level 2 is a reduction in voluntary turnover. The turnover costs that are avoided by improved workforce retention more than pay for the improvement costs associated with achieving Level 2.
3. At **PCMM Level 3**, the organization identifies and develops workforce competencies and aligns workforce and work group competencies with business strategies and objectives. For example, the workforce practices that were implemented at Level 2 are now standardized and adapted to encourage and reward growth in the organization's workforce competencies.
4. At **PCMM Level 4**, the organization empowers and integrates workforce competencies and manages performance quantitatively. For example, the organization is able to predict its capability for performing work because it can quantify the capability of its workforce and of the competency-based processes they use in performing their assignments.
5. At **PCMM Level 5**, the organization continuously improves and aligns personal, work-group, and organizational capability. For example, at Maturity Level 5, organizations treat continuous improvement as an orderly business process to be performed in an orderly way on a regular basis.

The **People Capability Maturity Model** was designed initially for knowledge- intense organizations and workforce management processes. However, it can be applied in almost any organizational setting, either as a guide in implementing workforce improvement activities or as a vehicle for assessing workforce practices.

Personal Software Process (PSP)

- The Personal Software Process (PSP) shows engineers how to
- manage the quality of their projects
- make commitments they can meet
- improve estimating and planning
- reduce defects in their products

PSP emphasizes the need to record and analyze the types of errors you make, so you can develop strategies eliminate them.

PSP model Framework Activities

- Planning – isolates requirements and based on these develops both size & resource estimates. A defect estimate is made.
- High level Design – external specification of all components. All issues are recorded and tracked.
- High level Design Review- formal verification to uncover errors
- Development- metrics are maintained for all important tasks & work results.
- Postmortem- using measures & metrics collected effectiveness of process is determined an improved.

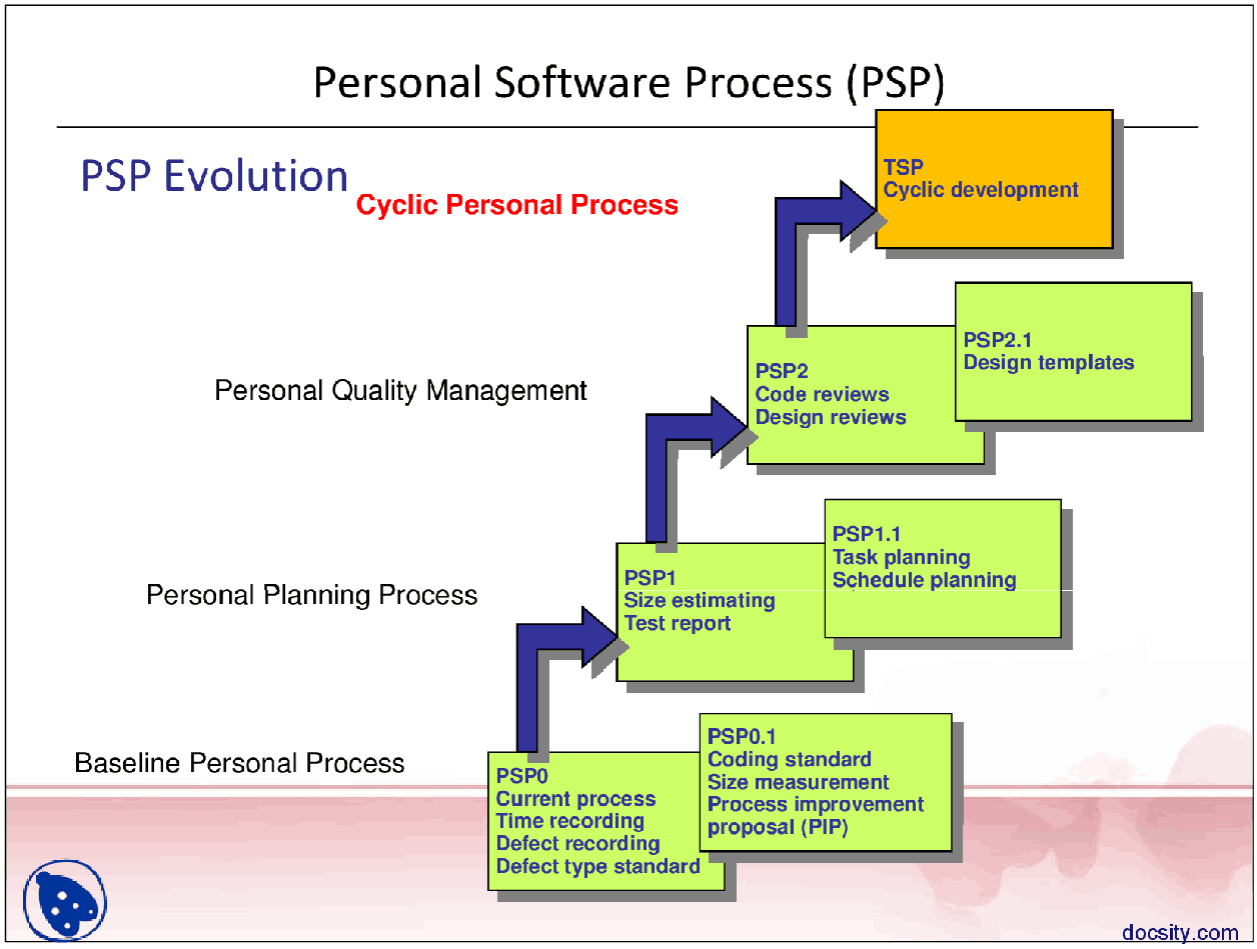
Because personnel costs constitute 70 percent of the cost of software development, the skills and work habits of engineers largely determine the results of the software development process.

Based on practices found in the CMMI, the PSP can be used by engineers as a guide to a disciplined and structured approach to developing software. The

PSP is a prerequisite for an organization planning to introduce the TSP.

- The PSP can be applied to many parts of the software development process, including

- small-program development
- requirement definition
- document writing
- systems tests
- systems maintenance
- enhancement of large software systems



Team Software Process (TSP)

- The Team Software Process (TSP), along with the Personal Software Process, helps the high-performance engineer to

- ensure quality software products

- create secure software products
- improve process management in an organization

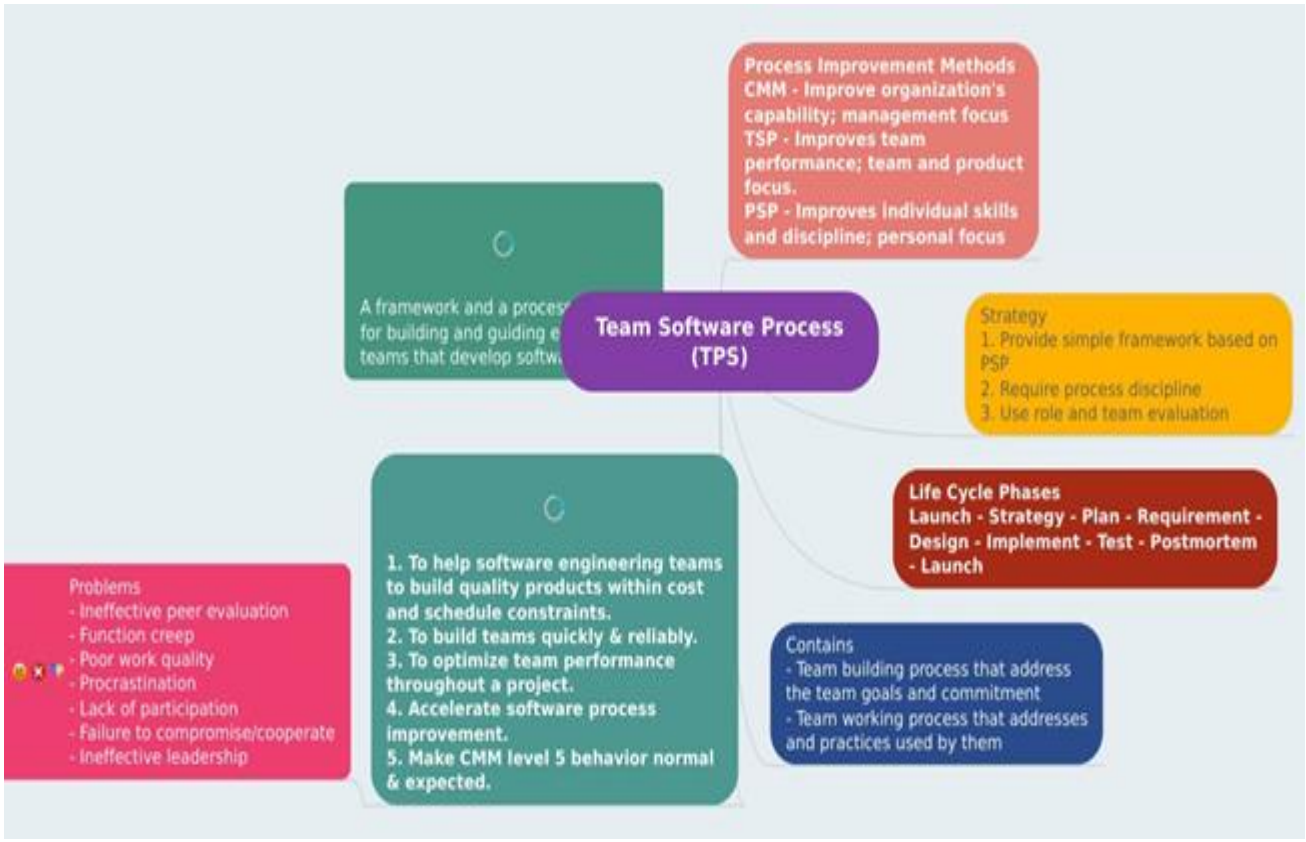
TSP Framework Activities

- Launch high level design
- Implementation
- Integration
- Test
- postmortem
- Engineering groups use the TSP to apply integrated team concepts to the development of software-intensive systems. A launch process walks teams and their managers through

- establishing goals
- defining team roles
- assessing risks
- producing a team plan

Benefits of TSP

- The TSP provides a defined process framework for managing, tracking and reporting the team's progress.
- Using TSP, an organization can build self-directed teams that plan and track their work, establish goals, and own their processes and plans. These can be pure software teams or integrated product teams of 3 to 20 engineers.
- TSP will help your organization establish a mature and disciplined engineering practice that produces secure, reliable software.


Process Improvement Methods
CMM - Improve organization's capability; management focus
TSP - Improves team performance; team and product focus.
PSP - Improves individual skills and discipline; personal focus
A framework and a proces
for building and guiding e
teams that develop softw
Team Software Process (TPS)
Strategy
1. Provide simple framework based on PSP
2. Require process discipline
3. Use role and team evaluation
Life Cycle Phases
Launch - Strategy - Plan - Requirement - Design - Implement - Test - Postmortem - Launch
1. To help software engineering teams to build quality products within cost and schedule constraints.
2. To build teams quickly & reliably.
3. To optimize team performance throughout a project.
4. Accelerate software process improvement.
5. Make CMM level 5 behavior normal & expected.
Contains
- Team building process that address the team goals and commitment
- Team working process that addresses and practices used by them
Problems
- Ineffective peer evaluation
- Function creep
- Poor work quality
- Procrastination
- Lack of participation
- Failure to compromise/cooperate
- Ineffective leadership

UNIT – II

Software Project Management Renaissance

Conventional Software Management, Evolution of Software Economics, Improving Software Economics, The old way and the new way.

Life-Cycle Phases and Process artifacts

Engineering and Production stages, inception phase, elaboration phase, construction phase, transition phase, artifact sets, management artifacts, engineering artifacts and pragmatic artifacts, model based software architectures

Part –I Software Project Management Renaissance

Conventional Software Management

1. The best thing about software is its flexibility:
   - It can be programmed to do almost anything.
2. The worst thing about software is its flexibility:
   - The "almost anything" characteristic has made it difficult to plan, monitor, and control software

development.

3. In the mid-1990s, three important analyses were performed on the software engineering industry.

All three analyses given the same general conclusion:-

"The success rate for software

projects is very low". They Summarized as follows:

1. Software development is still highly unpredictable. Only 10% of software projects are delivered successfully within initial budget and scheduled time.
2. Management discipline is more differentiator in success or failure than are technology advances.
3. The level of software scrap and rework is indicative of an immature process.

Software management process framework:

WATERFALLL MODEL

1. It is the baseline process for most conventional software projects have used.
2. We can examine this model in two ways:

IN THEORY

IN PRACTICE

IN THEORY:-

In 1970, Winston Royce presented a paper called "Managing the Development of Large Scale Software Systems" at IEEE WESCON.

Where he made three primary points:

1. There are two essential steps common to the development of computer programs:
   - analysis

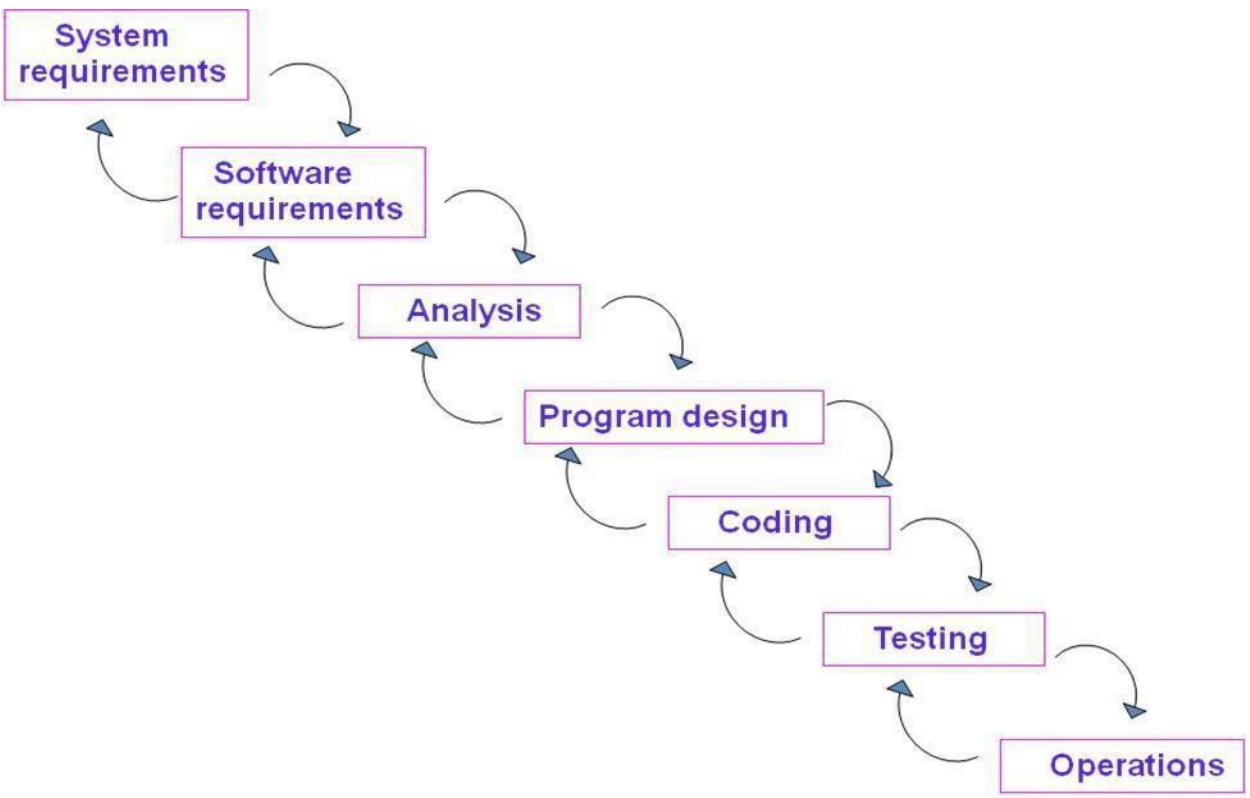
- coding

Waterfall Model Part 1: The two basic steps to build a program

2. In order to mange and control all of the intellectual freedom associated with software development one should follow the following steps:

- **System requirements definition**
- **Software requirements definition**
- **Program design and testing**

These steps addition to the analysis and coding steps



Waterfall Model Part 2: The large – scale system approach

3. Since the testing phase is at the end of the development cycle in the waterfall model, it may be risky and invites failure.

So we need to do either the requirements must be modified or a substantial design changes is warranted by breaking the software in to different pieces.

-**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 give 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 carryout proper testing the following things to be done:
  i) Employ a team of test specialists who were not responsible for the original design.
  ii) Employ visual inspections to spot the obvious errors like dropped minus signs, missing factors of two, jumps to wrong addresses.
  iii) Test every logic phase.
  iv) 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,
  i) Preliminary software review during preliminary program design step.
  ii) Critical software review during program design.
  iii) Final software acceptance review following testing.

IN PRACTICE:-

- Whatever the advices that are given by the software developers and the theory behind the waterfall model, some software projects still practice the conventional software management approach.

Projects intended for trouble frequently exhibit the following symptoms:

i) Protracted (delayed) integration

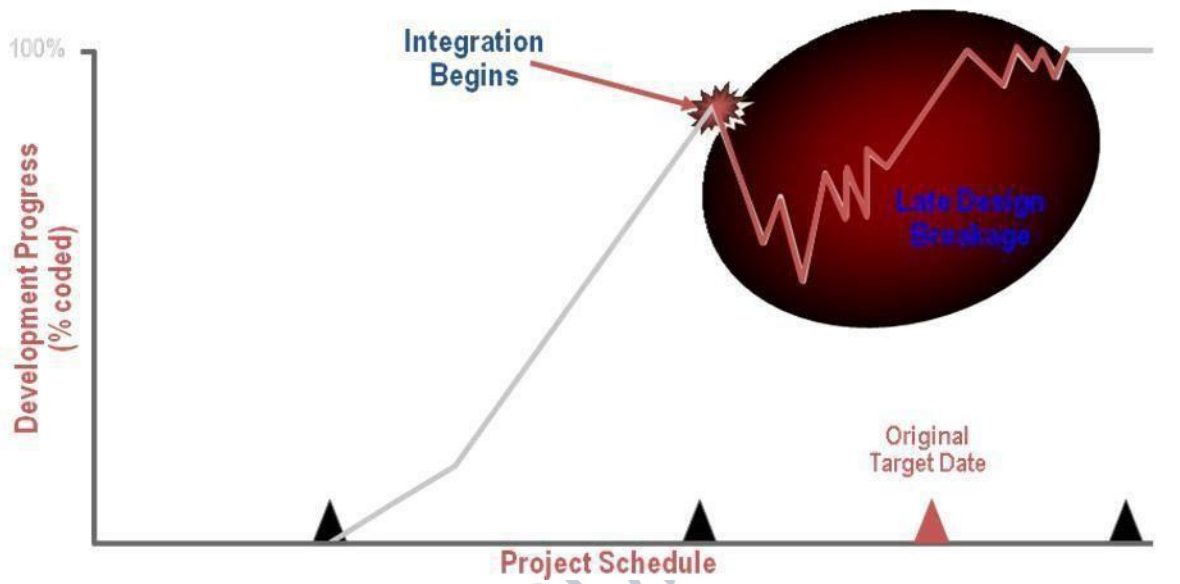
- In the conventional model, the entire system was designed on paper, then implemented all at once, then integrated. Only at the end of this process was it possible to perform system testing to verify that the

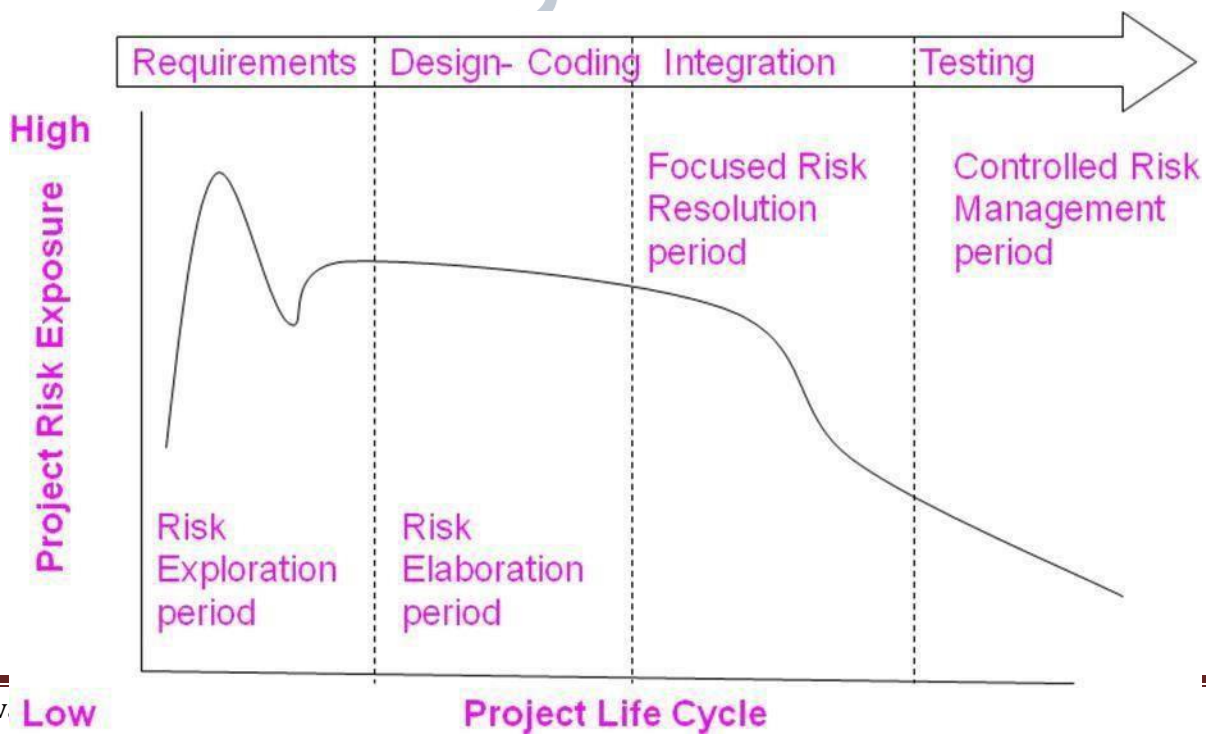
fundamental architecture was sound.

What Happens in Practice

Sequential activities:

Requirements → Design → Code → Integration → Test





ii) 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".

iii) Requirements-Driven Functional Decomposition

-Traditionally, the software development process has been requirement-driven: An attempt is made to provide a precise requirements definition and then to implement exactly those requirements.

-This approach depends on specifying requirements completely and clearly before other development activities begin.

- It frankly treats all requirements as equally important.

- Specification of requirements is a difficult and important part of the software development process. **iv) Adversarial Stakeholder Relationships**

The following sequence of events was typical for most contractual software efforts:

- The contractor prepared a draft contact-deliverable document that captured an intermediate artifact and delivered it to the customer for approval.

- The customer was expected to provide comments (within 15 to 30 days)

- The contractor integrated these comments and submitted a final version for approval (within 15 to 30 days)

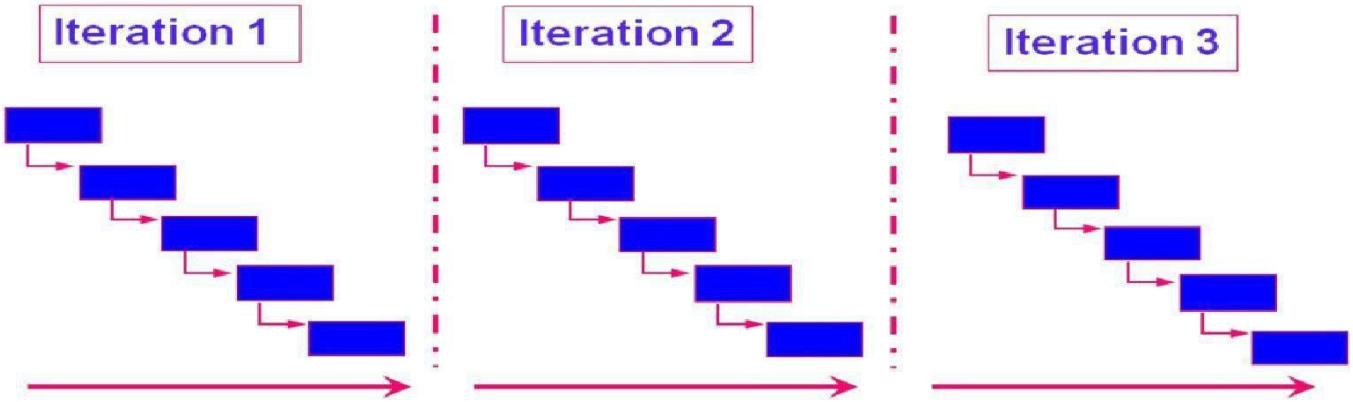
Project Stakeholders :

- Stakeholders are the people involved in or affected by project activities
- Stakeholders include
  - the project sponsor and project team
  - support staff
  - customers
  - users
  - suppliers
  - opponents to the project

v) Focus on Documents and Review Meetings

- The conventional process focused on various documents that attempted to describe the software product.
- Contractors 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.
- Most design reviews resulted in low engineering and high cost in terms of the effort and schedule involved in their preparation and conduct.

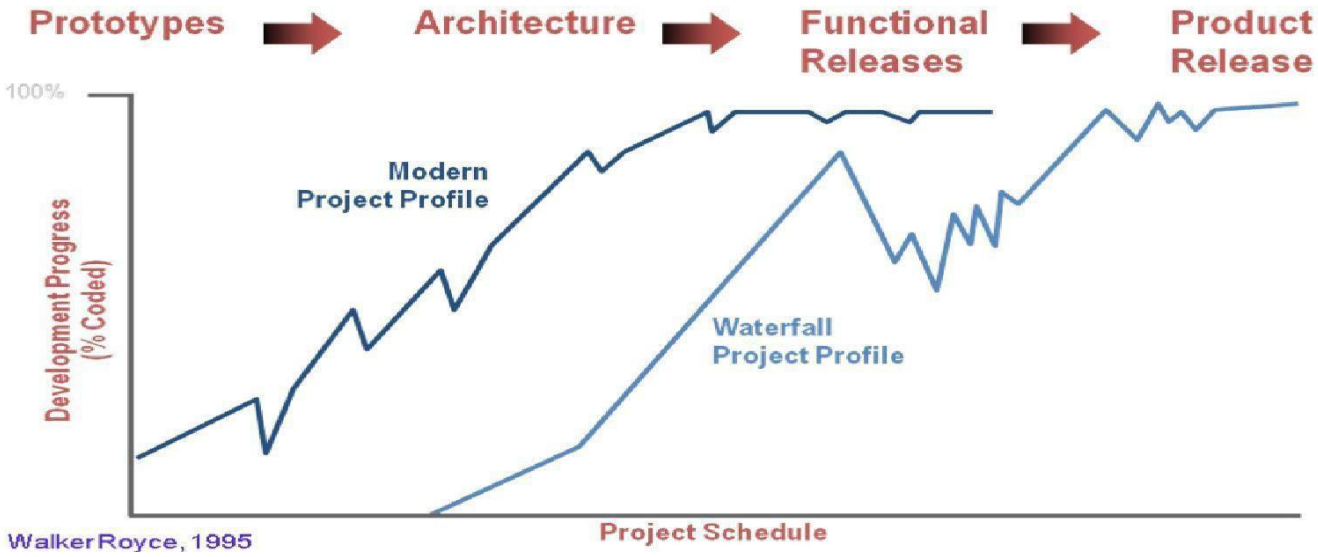
Iterative Development



- Earliest iterations address greatest risks
- Each iteration produces an executable release
- Each iteration includes integration and test

Better Progress Profile

Sequential phases, but iterative activities



Iterative Development Phases

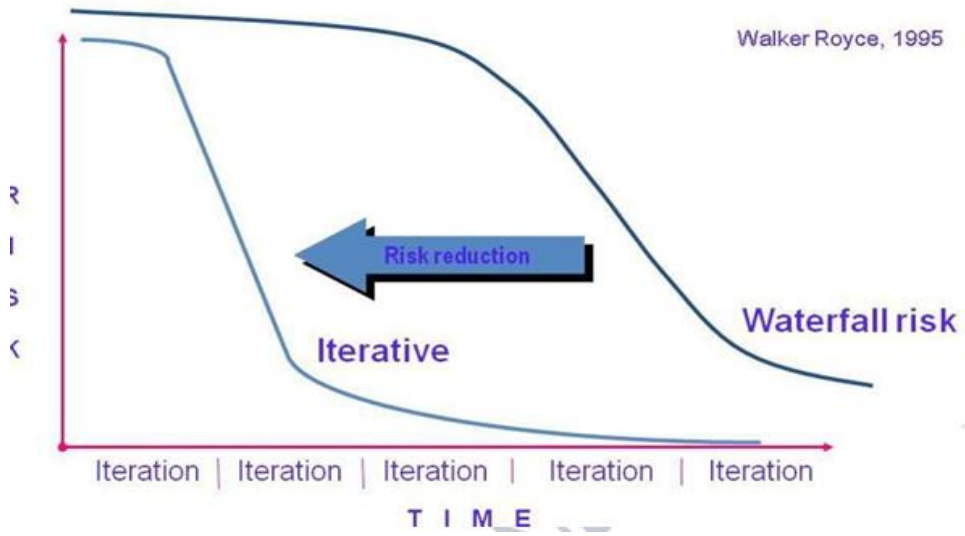
Inception: Agreement on overall scope
- Vision, high-level requirements, business case
- Not detailed requirements

Elaboration: Agreement on design approach
- Baseline architecture, most requirements detailed
- Not detailed design

Construction: Apply approach
- Working product, system test complete

Transition: Validate solution
- Stakeholder acceptance

Accelerate Risk Reduction



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.

Evolution of Software Economics

Project Sizes :

- Size as team strength could be :
  - Trivial (Minor) Size: 1 person
  - Small Size: 5 people
  - Moderate Size: 25 people
  - Large Size: 125 people
  - Huge Size: 625 people
- The more the size, the greater are the costs of management overhead, communication, synchronizations among various projects or modules, etc.

Reduce Software Size:

The less software we write, the better it is for project management and for product quality
- The cost of software is not just in the cost of „coding" alone; it is also in
  - Analysis of requirements
  - Design
  - Review of requirements, design and code
  - Test Planning and preparation
  - Testing
  - Bug fix
  - Regression testing
  - „Coding" takes around 15% of development cost
- Clearly, if we reduce 15 hrs of coding, we can directly reduce 100 hrs of development effort, and also reduce the project team size appropriately !

Size reduction is defined in terms of human-generated source code.
Most often, this might still mean that the computer-generated executable code is at least the same or even more

- Software Size could be reduced by
  - Software Re-use
  - Use of COTS (Commercial Off-The Shelf Software)
  - Programming Languages

PRAGMATIC SOFTWARE ESTIMATION:

- If there is no proper well-documented case studies then it is difficult to estimate the cost of the

software. It is one of the critical problem in software cost estimation.

- But the cost model vendors claim that their tools are well suitable for estimating

- iterative development projects.

- In order to estimate the cost of a project the following three topics should be considered,

1) Which cost estimation model to use.

2) Whether to measure software size in SLOC or function point.

3) What constitutes a good estimate.

- There are a lot of software cost estimation models are available such as, COCOMO,

CHECKPOINT, ESTIMACS, Knowledge Plan, Price-S,
ProQMS, SEER, SLIM, SOFTCOST, and SPQR/20.

- Of which COCOMO is one of the most open and well-documented cost estimation models

- The software size can be measured by using

1) SLOC 2) Function points

- Most software experts argued that the SLOC is a poor measure of size. But it has some value in the software Industry.

- SLOC worked well in applications that were custom built why because of easy to automate and instrument.

- Now a days there are so many automatic source code generators are available and there are so many advanced higher-level languages are available. So SLOC is a uncertain measure.

- The main advantage of function points is that this method is independent of the technology and is therefore a much better primitive unit for comparisons among projects and organizations.

- The main disadvantage of function points is that the primitive definitions are abstract and measurements are not easily derived directly from the evolving artifacts.

- Function points is more accurate estimator in the early phases of a project life cycle. In later phases, SLOC becomes a more useful and precise measurement basis of various metrics perspectives.

- The most real-world use of cost models is bottom-up rather than top-down.

- The software project manager defines the target cost of the software, then manipulates the parameters and sizing until the target cost can be justified.

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 3-1. *Important trends in improving software economics*

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

REDUCING SOFTWARE PRODUCT SIZE:

- By choosing the type of the language
- By using Object-Oriented methods and visual modeling
- By reusing the existing components and building reusable components &

By using commercial components,we can reduce the product size of a software.

OBJECT ORIENTED METHODS AND VISUAL MODELING:

- There has been a widespread movements in the 1990s toward Object- Oriented technology.
- Some studies concluded that Object-Oriented programming languages appear to benefit both software productivity and software quality. One of such Object-Oriented method is UML-Unified Modeling Language.

Booch described the following three reasons for the success of the projects that are using Object-Oriented concepts:

1) An OO-model of the problem and its solution encourages a common vocabulary between the end user of a system and its developers, thus creating a shared understanding of the problem being solved.

2) The use of continuous integration creates opportunities to recognize risk early and make incremental corrections without weaken the entire development effort.

3) An OO-architecture provides a clear separation among different elements of a system, crating firewalls that prevent a change in one part of the system from the entire architecture.

He also suggested five characteristics of a successful OO-Project,

1) A cruel focus on the development of a system that provides a well understood collection of essential minimal characteristics.

2) The existence of a culture that is centered on results, encourages communication, and yet is not afraid to fail.

3) The effective use of OO-modeling.

4) The existence of a strong architectural vision.

5) The application of a well-managed iterative and incremental development life cycle.

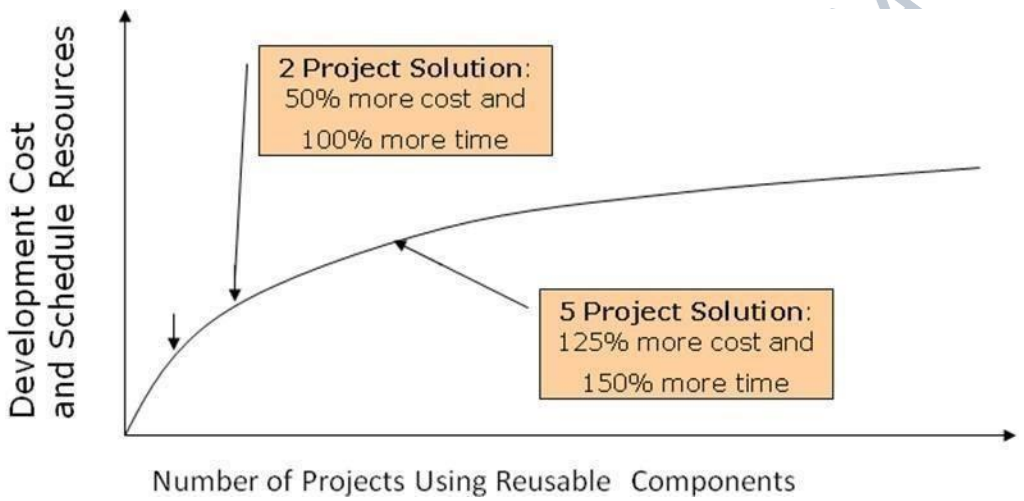
REUSE:

Organizations that translates reusable components into commercial products has the following characteristics:

- 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.



COMMERCIAL COMPONENTS

It is an Organization‟s policies, procedures, and practices for pursuing a software-intensive line of business.

The focus of this process is of organizational economics, long-term strategies, and a software ROI.

- **Macro process:**

A project‟s policies, and practices for producing a complete software product within certain cost, schedule, and quality constraints.

The focus of the macroprocess is on creating an sufficient instance of the metaprocess for a specific set of constraints.

- **Micro process:**

A projects team‟s policies, procedures, and practices for achieving an artifact of a software process.

The focus of the microprocess is on achieving an intermediate product baseline with sufficient functionality as economically and rapidly as practical.

The objective of process improvement is to maximize the allocation of resources to productive activities and minimize the impact of overhead activities on resources such as personnel, computers, and schedule.

Schedule improvement has at least three dimensions.

1. We could take an N-step process and improve the efficiency of each step.
2. We could take an N-step process and eliminate some steps so that it is now only an M-step process.
3. We could take an N-step process and use more concurrency in the activities being performed or the resources being applied.

IMPROVING TEAM EFFECTIVENESS:

- COCOMO model suggests that the combined effects of personnel skill and experience can have an impact on productivity as much as a factor of four over the unskilled personnel.

- Balance and coverage are two of the most important features of excellent teams. Whenever a team is in out of balance then it is vulnerable.

- It is the responsibility of the project manager to keep track of his teams. Since teamwork is much more important than the sum of the individuals.

Boehm – staffing principles:

- **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.

4) **The principle of team balance:** Select people who will complement and synchronize with one another.

5) **The principle of phase-out:** Keeping a misfit on the team doesn‟t benefit anyone.

In general, staffing is achieved by these common methods:

- If people are already available with required skill set, just take them
- If people are already available but do not have the required skills, re-train them
- If people are not available, recruit trained people
- If you are not able to recruit skilled people, recruit and train people

Staffing of key personnel is very important:

- Project Manager
- Software Architect

Important Project Manager Skills:

- **Hiring skills.** Few decisions are as important as hiring decisions. Placing the right person in the right job seems obvious but is surprisingly hard to achieve.
- **Customer-interface skill.** Avoiding adversarial relationships among stake- holders is a prerequisite for success.
- **Decision-making skill.** The jillion books written about management have failed to provide a clear definition of this attribute. We all know a good leader when we run into one, and decision-making skill seems obvious despite its intangible definition.
- **Team-building skill.** Teamwork requires that a manager establish trust, motivate progress, exploit eccentric prima donnas, transition average people into top performers, eliminate misfits, and consolidate diverse opinions into a team direction.
- **Selling skill.** Successful project managers must sell all stakeholders (including themselves) on decisions and priorities, sell candidates on job positions, sell changes to the status quo in the face of resistance, and sell achievements against objectives. In practice, selling requires continuous negotiation, compromise, and empathy.

Important Software Architect Skills:

- Technical Skills: the most important skills for an architect. These must include skills in both, the problem domain and the solution domain
- People Management Skills: must ensure that all people understand and implement the architecture in exactly the way he has conceptualized it. This calls for a lot of people management skills and patience.
- Role Model: must be a role model for the software engineers – they would emulate all good (and also all bad !) things that the architect does

IMPROVING AUTOMATION THROUGH SOFTWARE ENVIRONMENTS

The following are the some of the configuration management environments which provide the foundation for executing and implementing the process:

Planning tools, Quality assurance and analysis tools, Test tools, and User interfaces provide crucial automation support for evolving the software engineering artifacts.

TABLE 3-5. *General quality improvements with a modern process*

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

- Project inception. The proposed design was asserted to be low risk with adequate performance margin.
- Initial design review. Optimistic assessments of adequate design margin were based mostly on paper analysis or rough simulation of the critical threads. In most cases, the actual application algorithms and database sizes were fairly well understood. However, the infrastructure—including the operating system overhead, the database management overhead, and the interprocess and network communications overhead—and all the secondary threads were typically misunderstood.
- Mid-life-cycle design review. The assessments started whittling away at the margin, as early benchmarks and initial tests began exposing the optimism inherent in earlier estimates.
- Integration and test. Serious performance problems were uncovered, necessitating fundamental changes in the architecture. The underlying infrastructure was usually the scapegoat, but the real culprit was immature use of the infrastructure, immature architectural solutions, or poorly understood early design trade-offs.

PEER INSPECTIONS: A PRAGMATIC VIEW:

- 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

THE OLD WAY AND THE NEW

- Over the past two decades software development is a re-engineering process. Now it is replaced by advanced software engineering technologies.

- This transition is was motivated by the unsatisfactory demand for the software and reduced cost.

THE PRINCIPLES OF CONVENTIONAL SOFTWARE ENGINEERING

Based on many years of software development experience, the software industry proposed so many principles (nearly 201 by – Davis‟s). Of which Davis‟s top 30 principles are:

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 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 ifrequired.

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

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

9) **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.

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

11) **Good management** is more important than good technology

12) 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.

13) **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.

14) **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.

15) **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.

16) **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 firsttime.

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

18) **Design without documentation is not design**. I have often heard software engineers say, "I have finished the design. All that is left is thedocumentation."

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

vii) 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.

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

ix) Use coupling and cohesion. Coupling and cohesion are the best ways to measure software‟s inherent maintainability and adaptability.

x) 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.

xi) Don't test your own software. Software developers should never be the primary testers of their own software.

xii) 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.

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

THE 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.

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.

4) 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.

5) Capture design artifacts in rigorous, model-based notation:

- A model-based approach supports the evolution of semantically rich graphical and textual design notations.

- Visual modeling with rigorous notations and formal machine- process able language provides more objective measures than the traditional approach of human review and inspection of ad hoc design representations in paper doc.

6) Instrument the process for objective quality control and progress assessment:

- Life-cycle assessment of the progress and quality of all intermediate product must be integrated into the process.

- The best assessment mechanisms are well-defined measures derived directly from the evolving engineering artifacts and integrated into all activities and teams.

7) Use a demonstration-based approach to assess intermediate artifacts:

Transitioning from whether the artifact is an early prototype, a baseline architecture, or a beta capability into an executable demonstration of relevant provides more tangible understanding of the design tradeoffs, early integration and earlier elimination of architectural defects.

8) Plan intermediate releases in groups of usage scenarios with evolving levels of detail:

9) Establish a configurable process that economically scalable:

No single process is suitable for all software developments. The process must ensure that there is economy of scale and ROI.

Architecture-first approach → The central design element

Design and integration first, then production and test

Iterative life-cycle process → The risk management element

Risk control through ever-increasing function, performance, quality

Component-based development → The technology element

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

"Software Project Management"
Walker Royce

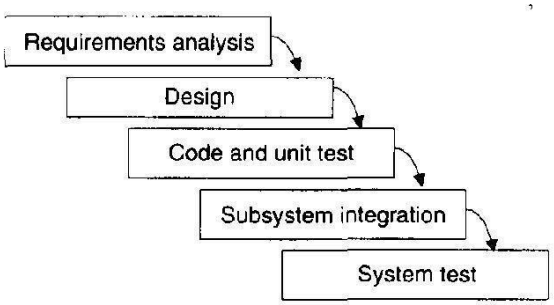
31/112

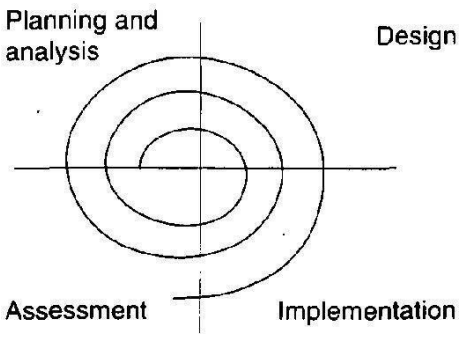
Waterfall Process

Requirements first
Custom development
Change avoidance
Ad hoc tools

Iterative Process

Architecture first
Component-based development
Change management
Round-trip engineering





PART-II Life cycle phases and process artifacts

LIFE-CYCLE PHASES

- If there is a well defined separation between "research and development" activities and"production" activities then the software is said to be in successful development process.

- Most of the software's fail due to the following characteristics ,

1) An overemphasis on research and development.

2) An overemphasis on production.

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 andcomponent 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 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.

1. INCEPTION PHASE:

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

PRIMARY OBJECTIVES

1) Establishing the project's scope and boundary conditions

2) Distinguishing the critical use cases of the system and the primary scenarios of operation

3) Demonstrating at least one candidate architecture against some of the primary scenarios

4) Estimating cost and schedule for the entire project

5) Estimating potential risks

ESSENTIAL ACTIVITIES:

1) Formulating the scope of the project
2) Synthesizing the architecture
3) Planning and preparing a business case

2. 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.
- At most of the time the process may accommodate changes, the elaboration phase activities must ensure that the architecture, requirements, and plans are stable. And also the cost and schedule for the completion of the development can be predicted within an acceptable range.

PRIMARY OBJECTIVES

1) Base lining the architecture as rapidly as practical
2) Base lining the vision
3) Base lining a high-reliability plan for the construction phase
4) Demonstrating that the baseline architecture will support the vision at a reasonable cost in a reasonable time.

ESSENTIAL ACTIVITIES

1) Elaborating the vision
2) Elaborating the process and infrastructure
3) Elaborating the architecture and selecting components

3. CONSTRUCTION PHASE

During this phase all the remaining components and application features are developed software is integrated where ever required.

- If it is a big project then parallel construction increments are generated.

PRIMARY OBJECTIVES

1) Minimizing development costs
2) Achieving adequate quality as rapidly as practical
3) Achieving useful version (alpha, beta, and other releases) as rapidly as practical

ESSENTIAL ACTIVITIES

1) Resource management, control, and process optimization
2) Complete component development and testing evaluation criteria
3) Assessment of product release criteria of the vision

4. TRANSITION PHASE

Whenever a project is grown-up completely and to be deployed in the end-user domain this phase is called transition phase. It includes 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

PRIMARY OBJECTIVES

1) Achieving user self-supportability

2) Achieving stakeholder concurrence

3) Achieving final product baseline as rapidly and cost-effectively as practical

ESSENTIAL ACTIVITIES

1) Synchronization and integration of concurrent construction increments into consistent deployment baselines

2) Deployment-specific engineering

3) Assessment of deployment baselines against the complete vision and acceptance criteria in the requirement set.

Artifacts of the Process

- Conventional s/w projects focused on the sequential development of s/w artifacts:
- Build the requirements
- Construct a design model traceable to the requirements &
- Compile and test the implementation for deployment.

-This process can work for small-scale, purely custom developments in which the design representation, implementation representation and deployment representation are closely aligned.

- This approach is doesn't work for most of today's s/w systems why because of having complexity and are composed of numerous components some are custom, some reused, some commercial products.

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*.

- *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:

1) Management 2) Requirements 3) Design

4) Implementation 5) 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**.**

THE MANAGEMENT SET

It captures the artifacts associated with process planning and execution. These artifacts use ad hoc notation including text, graphics, or whatever representation is required to capture the "contracts" among,

- **project personnel:**

project manager, architects, developers, testers, marketers, administrators

- **stakeholders:** funding authority, user, s/w project manager, organization manager, regulatory agency & between project personnel and stakeholders Management artifacts are evaluated, assessed, and measured through a combination of

1) Relevant stakeholder review.

2) Analysis of changes between the current version of the artifact and previous versions.

3) Major milestone demonstrations of the balance among all artifacts.

THE ENGINEERING SETS:

1) REQUIREMENT SET:

- 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**

1) Analysis of consistency with the release specifications of the mgmt set.

2) Analysis of consistency between the vision and the requirement models.

3) Mapping against the design, implementation, and deployment sets to evaluate the consistency and completeness and the semantic balance between information in the different sets.

4) Analysis of changes between the current version of the artifacts and previous versions.

5) Subjective review of other dimensions of quality.

2) DESIGN SET:

- UML notations are used to engineer the design models for the solution.

- It contains various levels of abstraction and enough structural and behavioral information to determine a bill of materials.

- Design model information can be clearly and, in many cases, automatically translated into a subset of the implementation and deployment set artifacts.

The design set is evaluated, assessed, and measured through a combination of

1) Analysis of the internal consistency and quality of the design model.

2) Analysis of consistency with the requirements models.

3) Translation into implementation and deployment sets and notations to evaluate the consistency and completeness and semantic balance between information in the sets.

4) Analysis of changes between the current version of the design model and previous versions.

5) Subjective review of other dimensions of quality.

3) IMPLEMENTATION SET:

- The implementation set include source code that represents the tangible implementations of components and any executables necessary for stand-alone testing of components.

- Executables are the primitive parts that are needed to construct the end product, including custom components, APIs of commercial components.

- Implementation set artifacts can also be translated into a subset of the deployment set. Implementation sets are human-readable formats that are evaluated, assessed, and measured through a combination of

1) Analysis of consistency with design models.

2) Translation into deployment set notations to evaluate consistency and completeness among artifact sets.

3) Execution of stand-alone component test cases that automatically c o m p a r e expected results with actual results.

4) Analysis of changes b/w the current version of the implementation set and previous versions.

5) Subjective review of other dimensions of quality.

4) DEPLOYMENT SET:

- 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

1) 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.

2) Testing the partitioning, replication, and allocation strategies in mapping components of the implementation set to physical resources of the deployment system.

3) Testing against the defined usage scenarios in the user manual such as installation, user-oriented dynamic reconfiguration, mainstream usage, and anomaly management.

4) Analysis of changes b/w the current version of the deployment set and previous versions.

5) Subjective review of other dimensions of quality.

Each artifact set uses different notations to capture the relevant artifact.

1) **Management set notations** (ad hoc text, graphics, use case notation) capture the plans, process, objectives, and acceptance criteria.

2) **Requirement notation** (structured text and UML models) capture the engineering context and the operational concept.

3) **Implementation notations** (software languages) capture the building blocks of the solution in humanreadable formats.

4) **Deployment notations** (executables and data files) capture the solution in machine-readable formats.

ARTIFACTS EVOLUTION OVER THE LIFE CYCLE

- Each state of development represents a certain amount of precision in the final system description.
- Early in the lifecycle, precision is low and the representation is generally high.

Eventually, the precision
of representation is high and everything is specified in full detail.

- At any point in the lifecycle, the five sets will be in different states of completeness. However, they should be at compatible levels of detail and reasonably traceable to one another.

- Performing detailed traceability and consistency analyses early in the life cycle

i.e. when precision is low and changes are frequent usually has a low ROI. **Inception phase:** It mainly focuses on critical requirements, usually with a secondary focus on an initial deployment view, little implementation and high- level focus on the design architecture but not on design detail.

Elaboration phase: It include generation of an executable prototype, involves subsets of development in all four sets. A portion of all four sets must be evolved to some level of completion before an architecture baseline can be established.

Fig: Life-Cycle evolution of the artifact sets

Construction: Its main focus on design and implementation. In the early stages the main focus is on the depth of the design artifacts. Later, in construction, realizing the design in source code and individually tested

components. This stage should drive the requirements, design, and implementation sets almost to completion. Substantial work is also done on the deployment set, at least to test one or a few instances of the programmed system through alpha or beta releases.

Transition: The main focus is on achieving consistency and completeness of the deployment set in the context of another set. Residual defects are resolved, and feedback from alpha, beta, and system testing is incorporated.

MANAGEMENT ARTIFACTS:

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

• The WBS is the architecture of project plan. It encapsulate change and evolve with appropriatelevel of details.

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

• A WBS provides the following information structure

- A delineation of all significant tasks.

- A clear task decomposition for assignment of responsibilities.

- A framework for scheduling ,debugging and expenditure tracking.

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

- Therefore WBS is a driving vehicle for budgeting and collecting cost.
- The structure of cost accountability is a serious project planning constraints.

Business case:

- Managing change is one of the fundamental primitives of an iterative development process.
- This flexibility increases the content, quality, and number of iterations that a project can achieve within a given schedule.
- Once software is placed in a controlled baseline, all changes must be formally tracked and managed.
- Most of the change management activities can be automated by automating data entry and maintaining change records online.

Engineering Artifacts

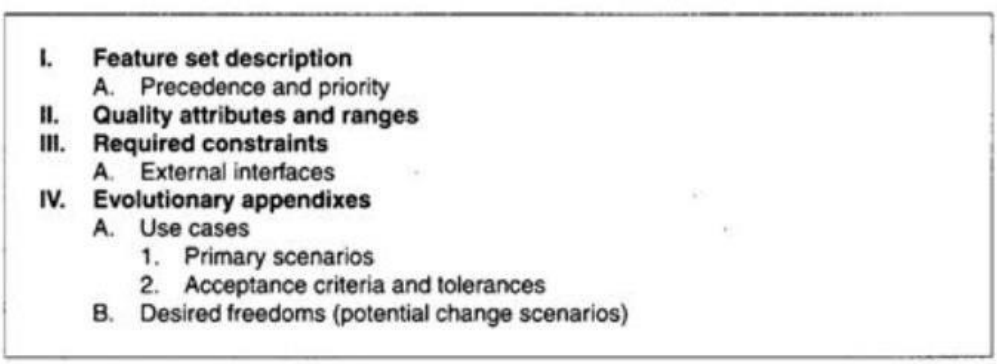
Engineering Artifacts are captured in rigorous engineering notations such as UML, programming languages, or executable machine codes. Three Engineering Artifacts are:

1. Vision Document.
2. Architecture Description.
3. S/W User Manual.

Vision Document

The source for capturing the expectations among stakeholders.

- Written from the users' perspective.
- Focus is on essential features of the system, and the acceptable levels of quality.
- Includes the operational concept



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

FIGURE 6-9. *Typical vision document outline*

Pragmatic Artifacts

- Pragmatic Meaning is dealing with things sensibly and realistically in a way that is based on practical rather than theoretical considerations.
- People want to review information but don't understand the language of the artifact.
- People want to review the 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: It is documentation that gets used.
- Paper is tangible(perceptible by touch): electronic artifacts are too easy to change.

Model-Based Software Architectures

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 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 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.

ARCHITECTURE: A TECHNICAL PERSPECTIVE

- Software architecture include the structure of software systems, their behavior, and the patterns that guide these elements, their collaborations, and their composition.
- An architecture framework is defined in terms of views is the abstraction of the UML models in the design set. Where as architecture view is an abstraction of the design model, include fullbreadth and depth of information.

Most real-world systems require four types of views:

1) Design: describes architecturally significant structures and functions of the design model.

2) Process: describes concurrency and control thread relationships among the design, component, and deployment views.

3) Component: describes the structure of the implementation set.

4) Deployment: describes the structure of the deployment set.

The design set include all UML design models describing the solution space.

- The design, process, and use case models provide for visualization of the logical and behavioral aspect of the design.
- The component model provides for visualization of the implementation set.

The deployment model provides for visualization of the deployment set.

1. The *use case view* describes how the system's critical use cases are realized by elements of the design model. It is modeled statistically by using use case diagrams, and dynamically by using any of the UML behavioral diagrams.

2. The *design view* describes the architecturally significant elements of the design model. It is modeled statistically by using class and object diagrams, and dynamically using any of the UML behavioral diagrams.

3. The *process view* addresses the run-time collaboration issues involved in executing the architecture on a distributed deployment model, including logical software topology, inter process communication, and state mgmt. it is modeled statistically using deployment diagrams, and dynamically using any of the UML behavioral diagrams.

4. The *component view* describes the architecturally significant elements of the implementation set. It is modeled statistically using component diagrams, and dynamically using any of the UML behavioral diagrams.

The *deployment view* addresses the executable realization of the system, including the allocation of logical processes in the distributed view to physical resources of the deployment network. It is modeled statistically using deployment diagrams, and dynamically using any of UML behavioral diagrams.

Architecture descriptions take on different forms and styles in different organizations and domains. At any given time, an architecture requires a subset of artifacts in engineering set.

- An architecture baseline is defined as a balanced subset of information across all sets, where as an architecture description is completely encapsulated within the design set.

Generally architecture base line include:

1) Requirements 2) Design

3) Implementation 4) Deployment

UNIT-III

Workflows and Checkpoints of process

Software process workflows, Iteration workflows, Major milestones, Minor milestones,

Periodic status assessments.

Process Planning

Work breakdown structures, Planning guidelines, cost and schedule estimating process,

iteration planning process, Pragmatic planning.

Part –I Work flows and check points of process

Workflows of the Process:

- In most of the cases a process is a sequence of activities why because of easy to understand, represent, plans and conduct.
- But the simplistic activity sequences are not realistic why because it includes many teams, making progress on many artifacts that must be synchronized, cross-checked, homogenized, merged and integrated.
- In order to manage complex software's the workflow of the software process is to be changed that is distributed.
- Modern software process avoids the life-cycle phases like inception, elaboration, construction and transition. It tells only the state of the project rather than a sequence of activities in each phase.
- The activities of the process are organized in to seven major workflows:

1) Management 2) Environment 3) Requirements
4) Design 5) Implementation 6) Assessment
7) Deployment

- These activities are performed concurrently, with varying levels of effort and emphasis as a project progresses through the life cycle.
- The management workflow is concerned with three disciplines:

1) Planning 2) Project control 3) Organization

Software Process Workflows

Previous chapters introduced a life-cycle macroprocess and the fundamental sets of artifacts. The macroprocess comprises discrete phases and iterations, but not discrete activities. A continuum of activities occurs in each phase and iteration. The next-level process description is the microprocesses, or workflows, that produce the artifacts. The term workflow is used to mean a thread of cohesive and mostly sequential activities. Workflows are mapped to product artifacts as described in Chapter 6 and to project teams as described in Chapter
11. There are seven top-level workflows:

1. **Management workflow**: controlling the process and ensuring win conditions for all stakeholders

2. **Environment workflow:** automating the process and evolving the maintenance environment

3. **Requirements workflow:** analyzing the problem space and evolving the requirements artifacts

4. **Design workflow:** modeling the solution and evolving the architecture and design artifacts

5. **Implementation workflow:** programming the components and evolving the implementation and deployment artifacts

6. **Assessment workflow**: assessing the trends in process and product quality

7. **Deployment workflow:** transitioning the end products to the user

Figure 8-1 illustrates the relative levels of effort expected across the phases in each of the top-level workflows. It represents one of the key signatures of a modern process framework and provides a viewpoint from which to discuss several of the key principles introduced in Chapter 4.

1. **Architecture-first approach.**
   Extensive requirements analysis, design, implementation, and assessment activities are performed before the construction phase, when full-scale implementation is ,the focus. This early life-cycle focus on implementing and testing the architecture must proceed full-scale

2. **Iterative life-cycle process.**

"Each phase portrays at least two iterations of each workflow. This default is intended to be descriptive, **not prescriptive**. Some projects may require only one iteration in a phase; others may require several iterations. The point is that the activities and artifacts of any given … core discipline may require more than one pass to achieve adequate results."

3. Round-trip Engineering

"Raising the environment activities to a first-class … Core Supporting Discipline is critical. The environment is the tangible embodiment of the project's process, methods, and notations for producing the artifacts."

4. Demonstration-based Approach

Implementation and assessment activities are initiated **early** in the life cycle, reflecting the emphasis on constructing executable subsets of the evolving architecture.

Iteration Workflows

- 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.

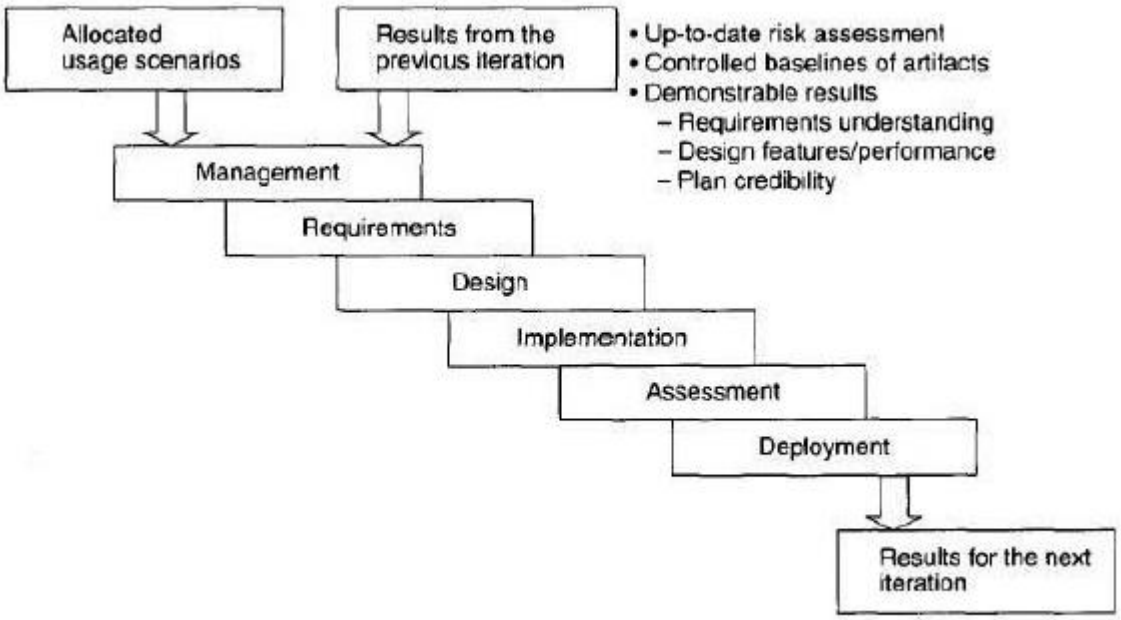


FIGURE 8-2. *The workflow of an iteration*

Workflow of an iteration

Example of usage scenario(ATM banking for the week)

- sally places her bank card into the ATM.
- sally successfully logs into the ATM using her personal identification number.

- sally deposits her weekly paycheck of $350 into her savings account.
- sally pays her phone bill of $75, her electric bill of $145, her cable bill of $55, and her water bill of $85 from her savings account
- sally attempts to withdraw $100 from her savings account for the weekend but discovers that she has insufficient funds
- sally withdraws $40 and gets her card back

Iteration emphasis across life cycle

Iteration Contents

- Management is concerned with the content of the iterations, assigning work, and the contents of anticipated releases.
- Environment is concerned primarily with maintaining the software change database – tracking, prioritizing, addressing any changes that arise.

Requirements: is concerned with looking carefully at the **baseline plan, architecture, and requirement set artifacts** needed to fully expand the use cases that need to be demonstrated at the end of this iteration – as well as their evaluation criteria.

Design: is concerned with **developing the design model and test model** by evolving the baseline architecture and design set artifacts against the **evaluation criteria** for a specific iteration; Also need to update the design set artifacts in response to the activities within this iteration

Implementation: addresses developing (from scratch), **customizing, or acquiring** (purchase, reuse) new components and to test and **integrate** these components into to **current architectural baseline.**

Assessment: concerned with evaluating the results of each iteration to insure compliance with **evaluation criteria** and to identify **rework** needed before deployment of this release or allocated to the next iteration; also, assess the results for this iteration so as to improve the next iteration's **procedure**.

Deployment: concerned with transmitting the release either internally or to an external organization for exercise.

CHECKPOINTS OF THE PROCESS

Check pointing is a technique to add fault tolerance into computing systems. It basically consists of saving a snapshot of the application's state, so that it can restart from that point in case of failure. This is particularly important for long running application that is executed in vulnerable computing system.

- It is important to place visible milestones in the life cycle in order to discuss the progress of the project by the stakeholders and also to achieve,

1) Synchronize stakeholder expectations and achieve agreement among the requirements, the design, and the plan perspectives.

2) Synchronize related artifacts into a consistent and balanced state.

3) Identify the important risks, issues, and out-of-tolerance conditions.

4) Perform a global review for the whole life cycle, not just the current situation of an individual perspective or intermediate product.

Three sequence of project checkpoints are used to synchronize stakeholder expectations throughout the lifecycle:

1) Major milestones 2) Minor milestones 3) Status assessments

- The most important major milestone is usually the event that transitions the project from the elaboration phase into the construction phase.

- The format and content of minor milestones are highly dependent on the project and the organizational culture.

- Periodic status assessments are important for focusing continuous attention on the evolving health of the project and its dynamic priorities.

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

1) Major milestones: These are the system wide events are held at the end of each development phase.

They provide visibility to system wide issues.

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

3) Status assessments: These are periodic events provide management with frequent and regular insight into the progress being made.

- An iteration represents a cycle of activities. Each of the lifecycle phases undergoes one or more iterations.

Minor milestones capture two artifacts: a release specification and a release description. Major milestones at the end of each phase use formal, stakeholderapprovedevaluation criteria and release descriptions; minor milestones use informal, development-team-controlled versions of these artifacts.

- Most projects should establish all four major milestones. Only in exceptional case you add other major milestones or operate with fewer. For simpler projects, very few or no minor milestones may be necessary to manage intermediate results, and the number of status assessments may be infrequent.

MAJOR MILESTONES:

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 different concerns:

Customers: schedule and budget estimates, feasibility, risk assessment, requirements understanding, progress, product line compatibility.

Users: consistency with requirements and usage scenarios, potential for accommodating growth,

quality attributes. Architects and systems engineers:product line compatibility, requirements changes, tradeoff analyses, completeness

Developers: Sufficiency of requirements detail and usage scenario descriptions, frameworks for component selection or development, resolution of development risk, product line compatibility, sufficiency of the development environment.

Maintainers: sufficiency of product and documentation artifacts, understandability, interoperability. with existing systems, sufficiency of maintenance environment.

Others: regulatory agencies, independent verification and validation contractors, venture capital investors, subcontractors, associate contractors, and sales andmarketing teams.

Life-Cycle Objective Milestone: These milestones occur at the end of the inception phase. The goal is to present to all stakeholders a recommendation on how to proceed with development, including a plan, estimated cost and schedule, and expected benefits and cost savings.

Life- Cycle Architecture Milestone: These milestones occur at the end of the elaboration phase. Primary goal is to demonstrate an executable architecture to all stakeholders. A more detailed plan for the construction phase is presented for approval. Critical issues relative to requirements and the operational concept are addressed.

Initial Operational Capability Milestone: These milestones occur late in theconstruction 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.

Product Release Milestone: Occur 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 and software quality metrics are reviewed to determine whether quality is sufficient for transition to the support organization.

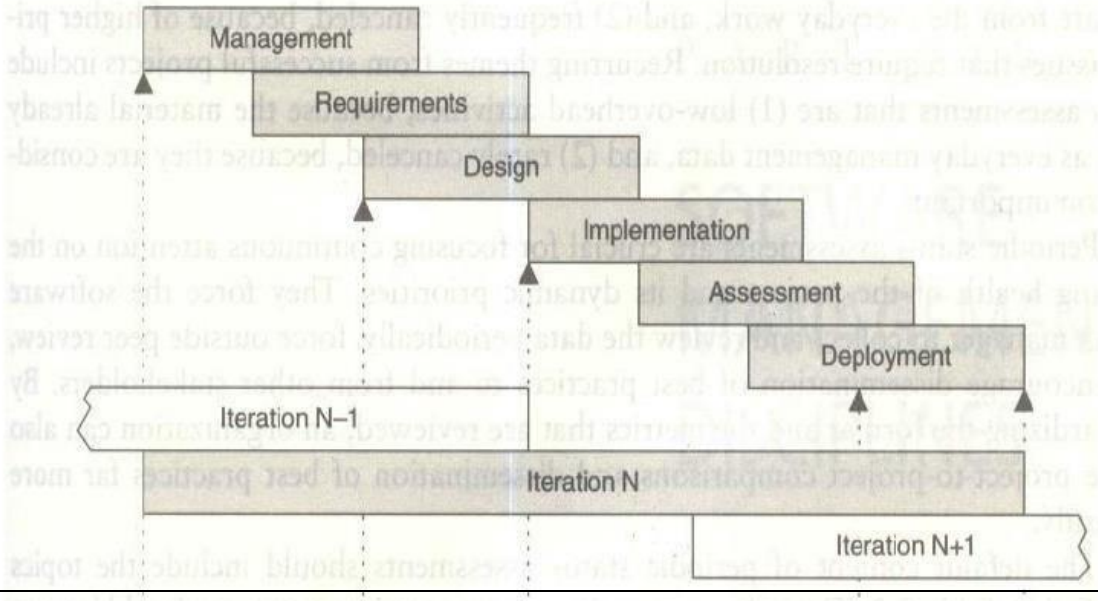
MINOR MILESTONES:

The number of iteration-specific, informal milestones needed depends on the content and length of the iteration. Iterations which have one- month to six-month duration have only two milestones are needed: the iteration readiness review and iteration assessment review. For longer iterations some other intermediate review points are added. All iterations are not created equal. Iteration takes different forms

and priorities, depending on where the project is in the life cycle. Early iterations focus on analysis and design. Later iterations focus 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 and the evaluation criteria that have been allocated to this iteration.

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



PERIODIC STATUS ASSESSMENTS:

- 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. Also provide,

1) A mechanism for openly addressing, communicating, and resolving management issues, technical issues, and project risks.

2) A mechanism for broadcast process, progress, quality trends, practices, and experience information to and from all stakeholders in an open forum.

3) Objective data derived directly from on-going activities and evolving product configurations.

Iterative Process Planning:

- Like software development, project planning is also an iterative process.

- Like software, plan is also an intangible one. Plans have an engineering stage,during which the plan is developed, and a production stage, where the plan is executed.

Part-II Process Planning

Work breakdown structures

- 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:

1) A pictorial description of all significant work.

2) A clear task decomposition for assignment of responsibilities.

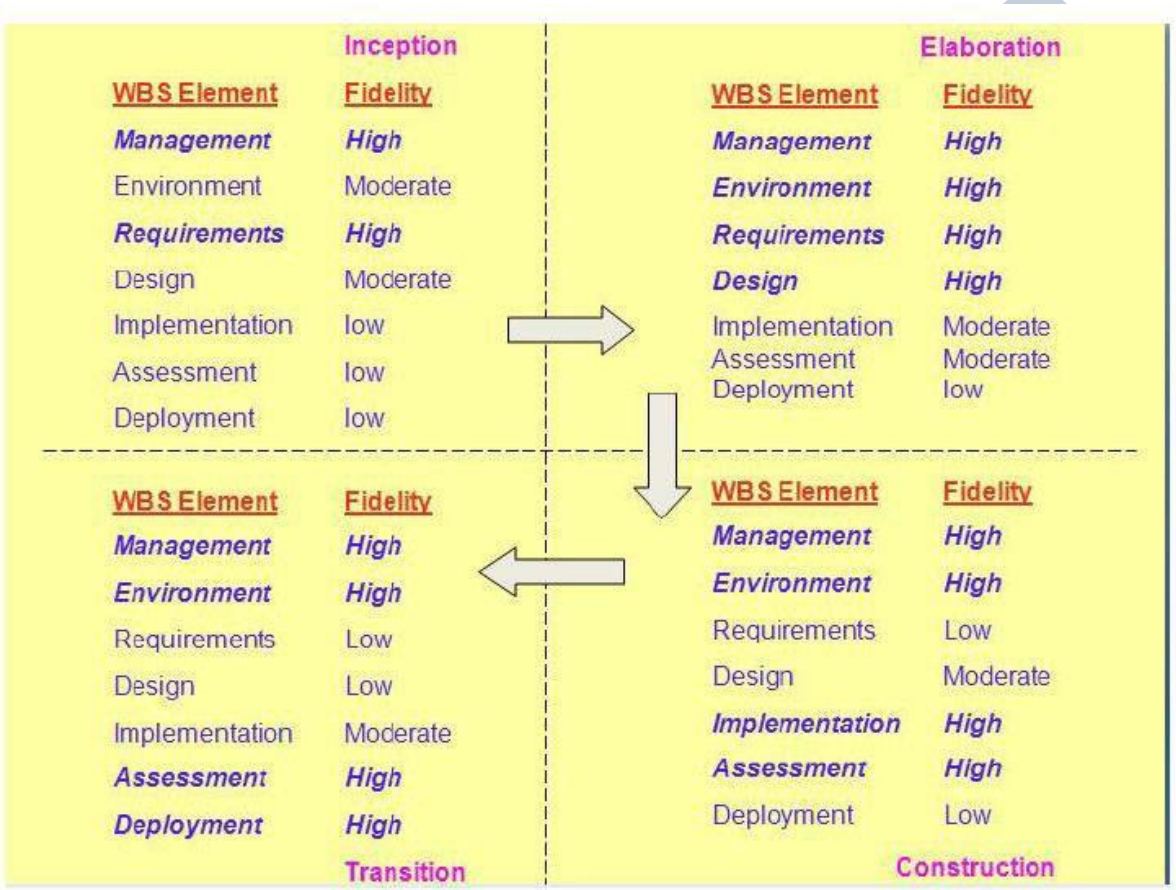
3) A framework for scheduling, budgeting, and expenditure tracking.

1) First-level elements: WBS elements are the workflows and are allocated to single team; provide the structure for the purpose of planning and comparison with the other projects.

2) Second-level elements: elements are defined for each phase of the life cycle.These elements allow the faithfulness of the plan to evolve more naturally with the level of understanding of the requirements and architecture, and the risks therein.

3) Third-level elements:

- These elements are defined for the focus of activities that produce the artifacts of each phase.
- These elements may be the lowest level in the hierarchy that collects the cost of discrete artifacts for a given phase, or they may be decomposed further into several lower level activities that, taken together, produce a single artifact.



PLANNING GUIDELINES:

- Software projects span a broad range of application domains.
- It is valuable but risky to make specific planning suggestions independent of project context.

- Planning provides a skeleton of the project from which the management people

can decide the starting point of the project.

- In order to proper plan it is necessary to capture the planning guidelines from most expertise and experience people.

- Project-independent planning advice is also risky. Adopting the planning guidelines blindly without being adapted to specific project circumstances is risk.

The above table provides default allocation for budgeted costs of each first-level WBS element.

- Sometimes these values may vary across projects but this allocation provides a good benchmark for assessing the plan by understanding the foundation for deviations from these guidelines.

- It is cost allocation table not the effort allocation.

The cost and schedule estimating process

Project plans need to be derived from two perspectives:

1) Forward-looking, top-down approach: It starts with an understanding 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 sequences would occur:

a) The software project manager develops a characterization of the overall size, process, environment, people, and quality required for the project.

b) A macro-level estimate of the total effort and schedule is developed using a software cost estimation model.

c) The software project manager partitions the estimate for the effort into a top level WBS using guidelines (table 10-1) and also partitions the schedule into major milestone dates and partition the effort into a staffing profile using guidelines

(table 10-2).

d) Subproject managers are given the responsibility for decomposing each of the WBS elements into lower levels using their tip-level allocation, staffing profile, and major milestone dates as constraints.

2) Backward-looking, bottom-up approach: We start with the end in mind, analyze the micro-level budgets and schedules, then sum all these elements into higher level budgets and intermediate milestones. This approach tends to define the WBS from the lowest levels upward. From this perspective, the following planning sequences would occur:

a) The lowest level WBS elements are elaborated into detailed tasks. These estimates tend to incorporate the project-specific parameters in an exaggerated way.

b) Estimates are combined and integrated into higher level budgets and milestones.

c) 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 topdown and bottom-up estimates.

- 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 bottomup 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.

The iteration planning process

Planning is concerned with defining the actual sequence of intermediate results. An Evolutionary build plan is important because there are always adjustments in build content and schedule as early conjecture evolves into well-understood project circumstances.

Iteration is used to mean a complete synchronization across the project, with a well-orchestrated global assessment of the entire project baseline. Inception iterations: The early prototyping activities integrate the foundation components of candidate architecture
- and provide an executable framework for elaborating the critical use cases of the system. This framework includes existing components, commercial components, and custom prototypes

sufficient to demonstrate candidate architecture and sufficient requirements understanding to establish a credible business case, vision, and software development plan.

Elaboration iterations: These iterations result in architecture, including a complete framework and infrastructure for execution. Upon completion of the architecture iteration, a few critical use cases should be demonstrable:

(1) initializing the architecture, (2) injecting a scenario to drive the worst-case data processing flow through the system (for example, the peak transaction throughput or peak load scenario), and (3) injecting a scenario to drive the worst-case control flow through the system (for example, orchestrating the fault-tolerance use cases).

Construction iterations: Most projects require at least two major construction iterations: an alpha release and a beta release.

Transition iterations: Most projects use a single iteration to transition a beta release into the final product. The general guideline is that most projects will use between four and nine iterations. The typical project would have the following six-iteration profile:

one iteration in inception: an architecture prototype
Two iterations in elaboration: architecture prototype and architecture baseline
Two iterations in construction: alpha and beta releases one iteration in transition: product release

A very large or unprecedented project with many stakeholders may require additional inception iteration and two additional iterations in construction, for a total of nine iterations.

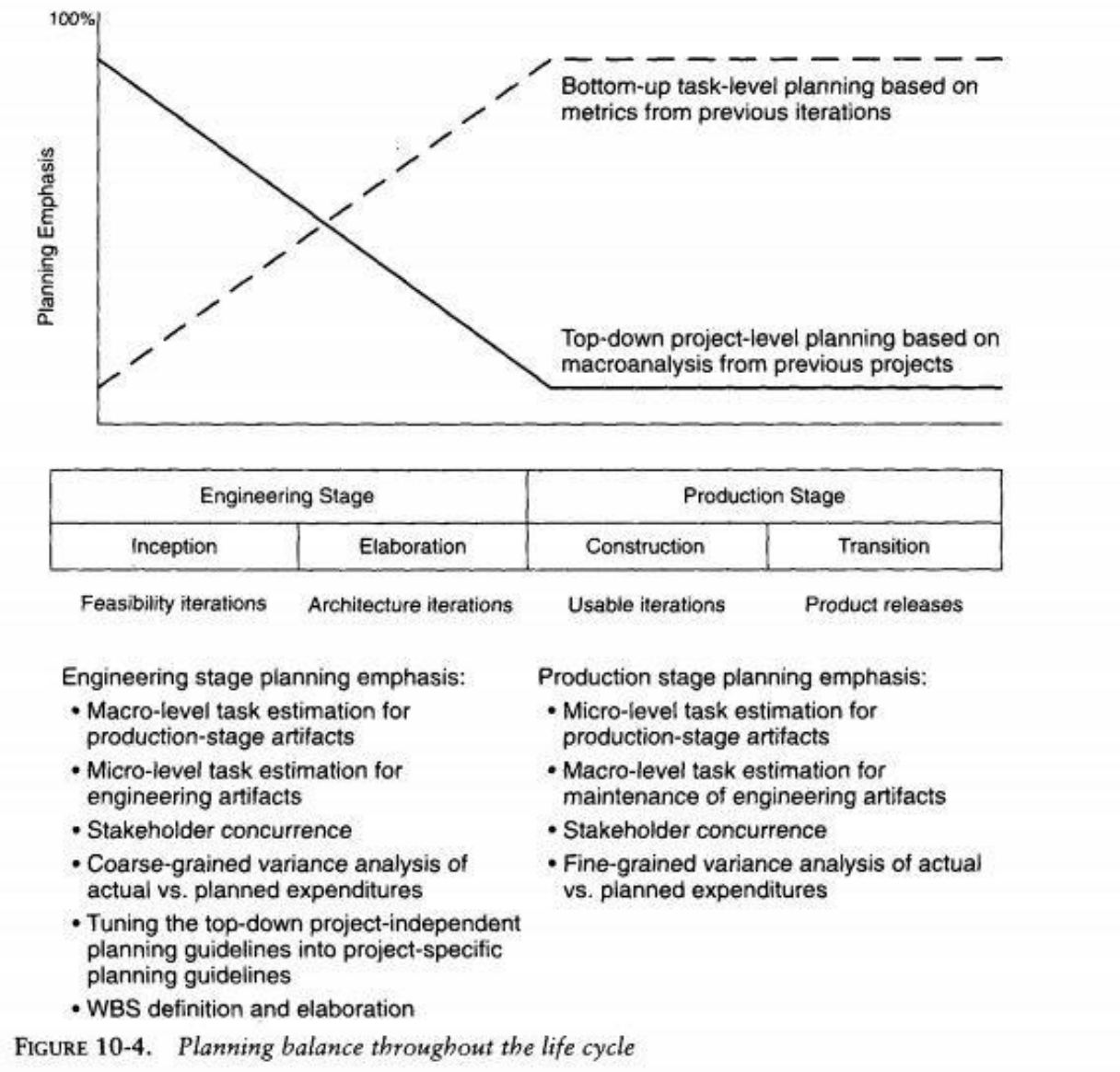


FIGURE 10-4. *Planning balance throughout the life cycle*

Pragmatic planning

Even though good planning is more dynamic in an iterative process,
doing it accurately is far easier. While executing iteration N of any phase, the software project manager must be monitoring and controlling against a plan that was initiated in iteration N - 1 and must be planning iteration N + 1. The art of good project· management is to make trade- offs in the current iteration plan and the next iteration plan based on
objective results in the current Iteration and previous iterations A side from bad architectures and misunderstood requirements, inadequate planning (and subsequent bad management) is one of the most common reasons for project failures. Conversely, the success of every successful project can be attributed in part to good planning.

UNIT-IV

Project Organizations

Line-of- business organizations, project organizations, evolution of organizations, process automation.

Project Control and process instrumentation

The seven core metrics, management indicators, quality indicators, life-cycle expectations, Pragmatic software metrics, and metrics automation.

Part-1 Project Organizations and Responsibilities

Project Organizations and Responsibilities

- **Organizations** engaged in software Line-of-Business need to support projects with the infrastructure necessary to use a common process.
- **Project** organizations need to allocate artifacts & responsibilities across project team to ensure a balance of global (architecture) & local (component) concerns.
- **The organization** must evolve with the WBS & Life cycle concerns. **Software lines of business & product teams have different motivation. Software lines of business** are motivated by return of investment (ROI),

new business discriminators, market diversification &profitability.

- **Project teams** are motivated by the cost, Schedule &quality of specific deliverables

1) Line-Of-Business Organizations:

The main features of default organization are as follows:

- Responsibility for process definition & maintenance is specific to a cohesive line of business.
- Responsibility for process automation is an organizational role & isequal in importance to the process definition role.
- Organizational role may be fulfilled by a single individual or several different teams. Fig: Default roles in a software Line-of-Business Organization. **Software Engineering Process Authority (SEPA)**

The SEPA facilities the exchange of information & process guidance both to & from project practitioners This role is accountable to General Manager for maintaining a current assessment

of the organization's process maturity & its plan for future improvement

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 **Software Engineering Environment Authority(SEEA)**

The SEEA is responsible for automating the organization's process, maintaining the organization's standard environment, Training projects to use the Environment &maintaining organization-wide reusable assets The SEEA role is necessary to achieve a significant ROI for common process.

Infrastructure

An organization's infrastructure provides human resources support, project- independent research & development, &other capital software engineering assets.

2) Project organizations:

Artifacts Activities

- **Business case Customer interface, PRA interface**
- **Software development plan Planning, monitoring**
- **Status assessments Risk management**
- **Software process definition**
- **Process improvement**

Figure 11-2. Default project organization and responsibilities

Software Management

Software Development Software Architecture Software Assessment System engineering Administration

- The above figure shows a default project organization and maps project-level roles and responsibilities.
- The main features of the default organization are as follows:
- **The project management team** is an active participant, responsible for producing as well as managing.
- **The architecture team** is responsible for real artifacts and for the integration of components, not just for staff functions.
- **The development team** owns the component construction and maintenance activities.
- The assessment team is separate from development.

- **Quality** is everyone's into all activities and checkpoints.
- Each team takes responsibility for a different quality perspective.

3) EVOLUTION OF ORGANIZATIONS:

Transition Construction

Inception:

Software management: **50%**
Software Architecture: 20%
Software development: 20% Software
Assessment
(measurement/evaluation):10%

Elaboration:

Software management: 10%
Software Architecture: **50%**
Software development: 20% Software
Assessment
(measurement/evaluation):20%

Construction:

Software management: 10%
Software Architecture: 10%
Software development: **50%** Software
Assessment
(measurement/evaluation):30%

Transition:

Software management: 10%
Software Architecture: 5%
Software development: 35% Software
Assessment
(measurement/evaluation):**50% The**

Process Automation

Introductory Remarks:

The environment must be the first-class artifact of the process.

Process automation& change management is critical to an iterative process. If the change is expensive then the development organization will resist it.

Round-trip engineering& integrated environments promote change freedom & effective evolution of technical artifacts.

Metric automation is crucial to effective project control.

External stakeholders need access to environment resources to improve interaction with the development team & add value to the process.

The three levels of process which requires a certain degree of process automation for the corresponding process to be carried out efficiently.

MANAGEMENT INDICATORS:

1. There are three fundamental sets of management metrics: technical progress, financial status, and staffing progress
2. By examining these perspectives, management can generally assess whether a project is on budget and on schedule
3. Most managers know their resource expenditures in terms of costs and schedule. The problem is to assess how much technical progress has been made.
4. Following three are management indicators:

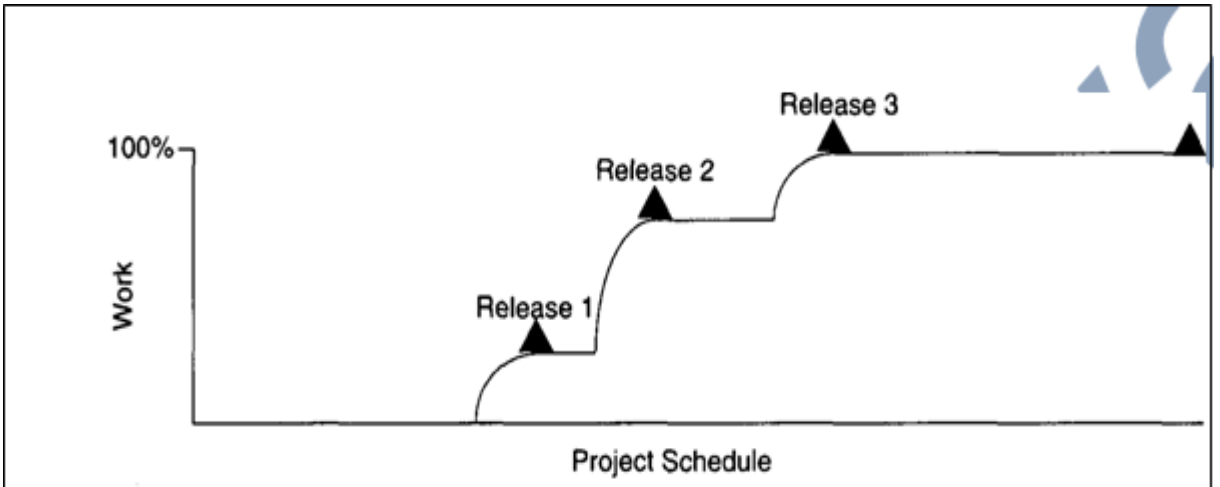


FIGURE 13-1. *Expected progress for a typical project with three major releases*

STAFFING A N D TEAM DYNAMICS:

1. Tracking actual versus planned staffing is a necessary is well-understood management metric.
2. There is one other important management indicator, the relationship between attrition and additions.
3. Increases in new staff can slow overall project progress as new people consume the more productive time.
4. Low attrition of good people is a sign of success.
5. If progress motivation is not there, good engineers will migrate elsewhere.
6. An increase in unplanned attrition-namely, people leaving a project prematurely-is one of the most obvious indicators that a project is destined for trouble.
7. The causes of such attrition can vary, but they are usually personnel dissatisfaction with management methods, lack of teamwork, or probability of failure in meeting the planned objectives.

BREAKAGE AND MODULARITY:

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

1. Modularity is the average breakage trend over time.
2. For a healthy project, the trend expectation is decreasing or stable. (Figure 13
3. This indicator provides insight into the benign (not harmful) or malignant (harmful) character of software change.
4. In a mature iterative development process, earlier changes are expected to result in more scrap than later changes.
5. Breakage trends that are increasing with time clearly indicate that product maintainability is

REWORK AND

1. Rework is defined as the average cost of change.
2. Which is the effort to analyze, resolve, and retest all changes to software baselines.
3. Adaptability is defined as the rework trend over time.
4. For a healthy project, the trend expectation is decreasing or stable.
5. Not all changes are treated equal. Some changes can be made in a staff-hour, while others take staff-weeks.
6. This metric provides insight into rework measurement.
7. In a mature iterative development process, earlier changes are expected to require more rework than later changes.
8. Rework trends that are increasing with time clearly indicate that product maintainability is difficult.

1. **Software errors can be categorized into two types**: deterministic and nondeterministic. Physicists would characterize these as Bohr-bugs and Heisen-bugs, respectively.
2. Bohr-bugs represent a class of errors that always result when the software is used in a same way. These errors are predominantly caused by coding errors, and changes are typically isolated to a single component.
1. Heisen-bugs are software faults that are coincidental situation. These errors are almost always design errors and typically are not repeatable even when the software is used in the same apparent way.
2. Conventional software programs executing a single program on a single processor typically contained only Bohr-bugs.

3. Modern, distributed systems with numerous interoperating components executing across a network of processors are vulnerable to Heisen-bugs, which are far more complicated to detect, analyze, and resolve.
4. The best way to mature a software product is to establish an initial test infrastructure that allows execution of test early in the life cycle.

Write note on lifecycle expectations.(13.3 table)

TABLE 13-3. *The default pattern of life-cycle metrics evolution*

METRIC	INCEPTION	ELABORATION	CONSTRUCTION	TRANSITION
Progress	5%	25%	90%	100%
Architecture	30%	90%	100%	100%
Applications	<5%	20%	85%	100%
Expenditures	Low	Moderate	High	High
Effort	5%	25%	90%	100%
Schedule	10%	40%	90%	100%
Staffing	Small team	Ramp up	Steady	Varying
Stability	Volatile	Moderate	Moderate	Stable
Architecture	Volatile	Moderate	Stable	Stable
Applications	Volatile	Volatile	Moderate	Stable
Modularity	50%–100%	25%–50%	<25%	5%–10%
Architecture	>50%	>50%	<15%	<5%
Applications	>80%	>80%	<25%	<10%
Adaptability	Varying	Varying	Benign	Benign
Architecture	Varying	Moderate	Benign	Benign
Applications	Varying	Varying	Moderate	Benign
Maturity	Prototype	Fragile	Usable	Robust
Architecture	Prototype	Usable	Robust	Robust
Applications	Prototype	Fragile	Usable	Robust

There is no mathematical evidence for using the seven core metrics. However, there were specific reasons for selecting them:

- The quality indicators are derived from the activity of ongoing process.
- They provide insight into the waste generated by the process. Scrap and rework metrics of manufacturing processes.
- Rather than focus on the value, they explicitly concentrate on the trends or changes with respect to time.
- The combination of insight from the current value and the current trend provides sufficient information for management.

Write note on pragmatic software metrics

Basic characteristics of good metric are as follows:

1. It is considered meaningful by the customer, manager, and performer. If any one of these stakeholders does not see the metric as meaningful, it will not be used.
2. It demonstrates quantifiable correlation between process perturbations (problems) and business performance.
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, and requirement),
4. It displays trends. This is an important characteristic. Understanding the change in a metric's value with respect to time, subsequent (later) projects, subsequent releases, and so forth is an extremely important perspective.
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 b y automated tools.

Following examples shows why the values of metrics should be interpreted 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 large 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 down if they are untrained new hires who demand extensive support from productive people to get up to speed.

1. There are many opportunities to automate the project control activities of a software project.
2. To analyze the data, software project control panel (SPCP) that maintains an on-line version of the status of s/w provides the key advantage.
3. This concept was first recommended by the Airlie Software Council [Brown,1996], using the "dashboard".(displaying metrics /score)
4. The idea is to provide a display panel that integrates d a t a from multiple sources to show the current s tatu s of the project.
5. For example, the software project manager would want to see a display with overall project values, a test manager may want to see a display focused on metrics specific to an upcoming beta release, and development managers may be interested only in data concerning the subsystems and components for which they are responsible.
6. The panel can support standard features such as warning lights, thresholds, variable scales, digital formats, and analog formats to present an overview of the current situation.
7. This automation support can improve m a n a g e m e n t insight i n t o p r o g r e s s a n d quality trends and improve the acceptance of metrics by the engineering team.

To implement a complete SPCP, it is necessary to define and develop the following

- Metrics primitives: indicators, trends, comparisons, and progressions
- A graphical u s e r interface: GUI support for a software p r o j e c t m a n a g e r role and flexibility to support other roles.
- Metrics collection agents: data extraction from the environment tools.
- Metrics data management server: stores the data.
- metrics definitions: actual metrics presentations for requirements progress (extracted from requirements set artifacts), design progress (extracted from design set artifacts), implementation progress (extracted from implementation set artifacts), assessment progress (extracted from deployment set artifacts), and other progress dimensions (extracted from manual sources, financial management systems, management artifacts, etc.)
- Actors: typically, the monitor and the administrator.

What are four graphical objects of top level display

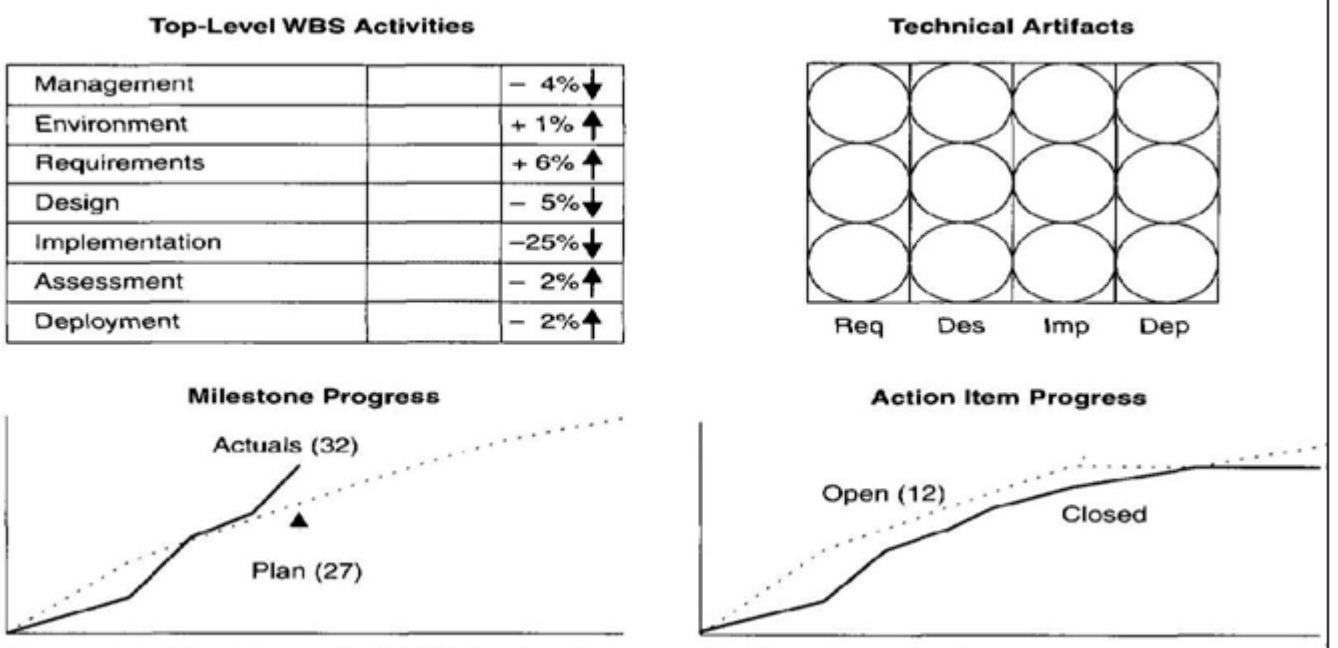


FIGURE 13-10. *Example SPCP display for a top-level project situation*

1. Project activity status. The graphical o b j e c t in the upper left provides an overview of the status of the top-level WBS elements. The seven elements could be coded red, yellow, and green to reflect the current e a r n e d v a l u e status. (In Figure 13-10, they are coded with white and shades of gray.) For example, green would represent ahead of plan, yellow would indicate within 10% of plan, and red would identify elements that have a greater than 10% cost or schedule variance.
2. Technical artifact status . The graphical ob j e ct in the upper right provides an overview of the status of technical artifacts. The Req light would display current state of requirements specifications. The Des light would do the same for the design models, the Imp light for the source code baseline, and the Dep light for the test program.
3. Milestone progress. The graphical object in the lower left provides a progress assessment of the achievement of milestones against plan.
4. Action item progress. The graphical object in the lower right provides a different perspective of progress, showing t h e current n u m b e r o f open and closed issues.

UNIT V

CCPDS-R Case Study and Future Software Project Management Practices

Modern Project Profiles, Next-Generation software Economics, Modern Process Transitions.

CCPDS-R Case Study and Future Software Project Management Practices

MODERN PROJECT PROFILES

Continuous Integration

In the iterative development process, firstly, the overall architecture of the project is created and then all the integration steps are evaluated to identify and eliminate the design errors. This approach eliminates problems such as downstream integration, late patches and shoe-horned software fixes by implementing sequential or continuous integration rather than implementing large-scale integration during the project completion.

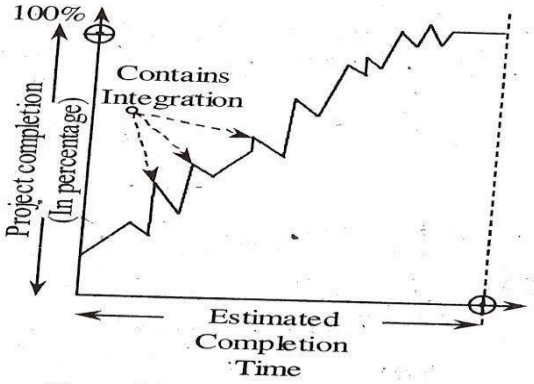


Figure (a): Modern Project Profile

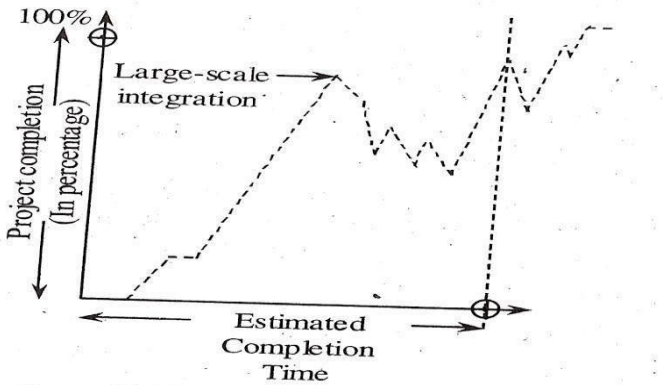


Figure (b): Traditional Project Profile

- Moreover, it produces feasible and a manageable design by delaying the 'design breakage' to the engineering phase, where they can be efficiently resolved. This can be one by making use of project demonstrations which forces integration into the design phase.
- With the help of this continuous integration incorporated in the iterative development process, the quality tradeoffs are better understood and the system features such as system performance, fault tolerance and maintainability are clearly visible even before the completion of the project.
- In the modern project profile, the distribution of cost among various workflows or project is completely different from that of traditional project profile as shown below:

Software Engineering Workflows	Conventional Process Expenditures	Modern process Ex
Management	**5%**	**10%**
Environment	**5%**	**10%**
Requirements	**5%**	**10%**
Design	**10%**	**15%**
Implementation	**30%**	**25%**
Assessment	**40%**	**25%**
Deployment	**5%**	**5%**

As shown in the table, the modern projects spend only 25% of their budget for integration and Assessment activities whereas; traditional projects spend almost 40% of their total budget for these activities. This is because, the traditional project involve inefficient large-scale integration and late identification of design issues.

Early Risk Resolution

- In the project development lifecycle, the engineering phase concentrates on identification and elimination of the risks associated with the resource commitments just before the production stage. The traditional projects involve, the solving of the simpler steps first and then goes to the complicated steps, as a result the progress will be visibly good, whereas, the modern projects focuses on 20% of the significant requirements, use cases, components and risk and hence they occasionally have simpler steps.

- To obtain a useful perspective of risk management, the project life cycle has to be applied on the principles of software management. The following are the 80:20 principles.
- The 80% of Engineering is utilized by 20% of the requirements.
- Before selecting any of the resources, try to completely understand all the requirement because irrelevant resource selection (i.e., resources selected based on prediction) may yield severe problems.
- 80% of the software cost is utilized by 20% of the components
- Firstly, the cost-critical components must be elaborated which forces the project to focus more on controlling the cost.
- 80% of the bugs occur because of 20% of the components
- Firstly, the reliability-critical components must be elaborated which give sufficient time for assessment activities like integration and testing, in order to achieve the desired level of maturity.
- 80% of the software scrap and rework is due to 20% if the changes.
- The change-critical components r elaborated first so that the changes that have more impact occur when the project is matured.
- 80% of the resource consumption is due to 20% of the components.
- Performance critical components are elaborated first so that, the trade-offs with reliability; changeability and cost-consumption can be solved as early as possible.
- 80% of the project progress is carried-out by 20% of the people
- It is important that planning and designing team should consist of best processionals because the entire success of the project depends upon a good plan and architecture.
- In the project life cycle the requirements and design are given the first and the second preference respectively. The third preference is given to the traceability between the requirement and the design components these preferences are given in order to make the design structure evolve into an organization so it parallels the structure of the requirements organization.

- Modern architecture finds it difficult to trace the requirements because of the following reasons.
    - Usage of Commercial components
    - Usage of legacy components
    - Usage of distributed resources
    - Usage of object oriented methods.
- Moreover, the complex relationships such as one-one, many-one, one-many, conditional, time-based and state based exists the requirements statement and the design elements.

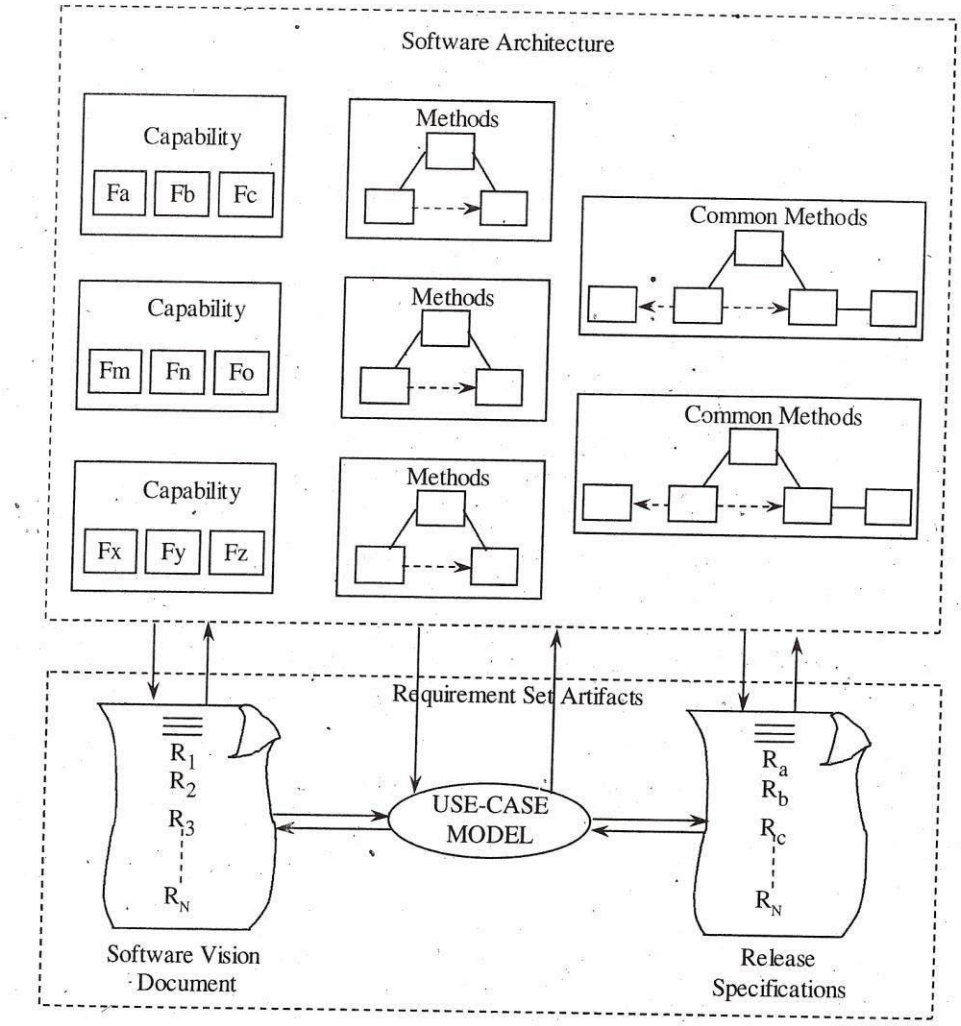


Figure: Software Component Organization of a Modern Process

As shown in the above figure, the top category system requirements are kept as the vision whereas, those with the lower category are evaluated. The motive behind theses artifacts is to gain fidelity with respect to the progress in the project lifecycle. This serves as a significant different from the traditional approach because, in traditional approach the fidelity is predicted early in the project life cycle.

8.1.4 Teamwork among stakeholders

- Most of the characteristics of the classic development process worsen the stakeholder

relationship s which in turn makes the balancing of requirement product attributes and plans difficult. An iterative process which has a good relationship between the stakeholders mainly focuses on objective understanding by each and every individual stakeholder. This process needs highly skilled customers, users and monitors which have experience in both the application as well as software. Moreover, this process requires an organization whose focus is on producing a quality product and achieves customer satisfaction.

- The table below shows the tangible results of major milestones in a modern process.
- From the above table, it can be observed that the progress of the project is not possible unless all the demonstration objectives are satisfied. This statement does not present the renegotiation of objectives, even when the demonstration results allow the further processing of tradeoffs present in the requirement, design, plans and technology.
- Modern iterative process that rely on the results of the demonstration need al its stakeholders to be well-educated and with a g good analytical ability so as to distinguish between the obviously negative results and the real progress visible. For example, an early determined design error can be treated as a positive progress instead to a major issue.

Principles of Software Management

- Software management basically relies on the following principles, they are,

1. Process must be based on architecture-first approach

If the architecture is focused at the initial stage, then there will be a good foundation for almost 20% of the significant stuff that are responsible for the overall success of the project. This stuff include the requirements, components use cases, risks and errors. In other words, if the components that are being involved in the architecture are well known then the expenditure causes by scrap and rework will be comparatively less.

2. Develop an iterative life-cycle process that identifies the risks at an early stage

An iterative process supports a dynamic planning framework that facilitates the risk management predictable performance moreover, if the risks are resolved earlier, the predictability will be more and the scrap and rework expenses will be reduced.

3. After the design methods in-order to highlight components-based development.

The quantity of the human generated source code and the customized development can be reduced by concentrating on individual components rather than individual lines-of-code. The complexity of software is directly proportional to the number of artifacts it contains that is, if the solution is smaller then the complexity associated with its management is less.

4. Create a change management Environment

Highly-controlled baselines are needed to compensate the changes caused by various teams that concurrently work on the shared artifacts.

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

The roundtrip-engineering is an environment that enables the automation and synchronization of engineering information into various formats. The engineering information usually consists requirement specification, source code, design models test cases and executable code. The automation of this information allows the teams to focus more on engineering rather than dealing with over head involved.

Design artifacts must be captured in model based notation.

The design artifacts that are modeled using a model based notation like UML, are rich in graphics and texture. These modeled artifacts facilitate the following tasks.

- **Complexity control**
- **Objective fulfillment**
- **Performing automated analysis**

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

The progress in the lifecycle as well as the quality of intermediately products must be estimated and incorporated into the process. This can be done with the help of well defined estimation mechanism that are directly derived from the emerging artifacts. These mechanisms provide detailed information about trends and correlation with requirements.

8. Implement a Demonstration-based Approach for Estimation of intermediately Artifacts

This approach involves giving demonstration on different scenarios. It facilitates earl integration and better understanding of design trade-offs. Moreover, it eliminates architectural defects earlier in the lifecycle. The intermediately results of this approach are definitive.

The Points Increments and generations must be made based on the evolving levels of detail

Here, the 'levels of detail' refers to the level of understanding requirements and architecture. The requirements, iteration content, implementations and acceptance testing can be organized using cohesive usage scenarios.

10. Develop a configuration process that should be economically scalable

The process framework applied must be suitable for variety of applications. The process must make use of processing spirit, automation, architectural patterns and components such that it is economical and yield investment benefits.

NEXT GENERATION SOFTWARE ECONOMICS

Next generation software cost models

- In comparison to the current generation software cost modes, the next generation software cost models should perform the architecture engineering and application production separately. The cost associated with designing, building, testing and maintaining the architecture is defined in terms of scale, quality, process, technology and the team employed.
- After obtaining the stable architecture, the cost of the production is an exponential function of size, quality and complexity involved.
- The architecture stage cost model should reflect certain diseconomy of scale (exponent less than 1.0) because it is based on research and development-oriented concerns. Whereas the production stage cost model should reflect economy of scale (exponent less than 1.0) for production of commodities.
- The next generation software cost models should be designed in a way that, they can assess larger architectures with economy of scale. Thus, the process exponent will be less than 1.0 at the time of production because large systems have more automated proves components and architectures which are easily reusable.
- The next generation cost model developed on the basis of architecture-first approach is shown below.

- A Plan with less fidelity and risk resolution
- It is technology or schedule-based
- It has contracts with risk sharing
- Team size is small but with experienced professionals.
- The architecture team, consists of small number of software engineers
- The application team consists of small number of domain engineers.
- The output will be an executable architecture, production and requirements
- The focus of the architectural engineering will be on design and integration of entities as well as host development environment.
- It contains two phases they are inspection and elaboration.

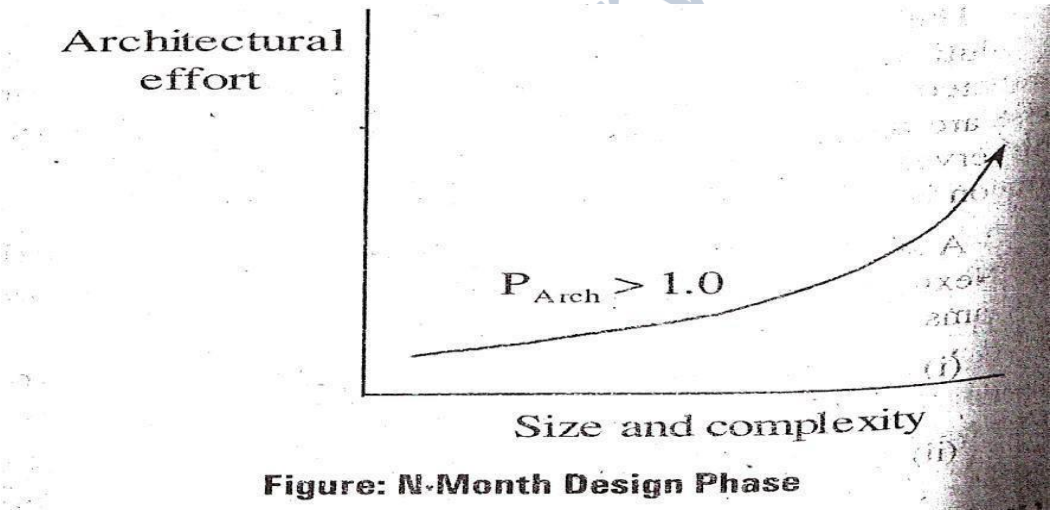


Figure: N-Month Design Phase

- At Application production stage
    - A plan with high fidelity and lower risk
    - It is cost-based
    - It has fixed-priced contracts
    - Team size is large and diverse as needed.
    - Architecture team consists of a small number of software engineers.

- The Application team may have nay number of domain engineers.
- The output will be a function which is deliverable and useful, tested

baseline and warranted quality.

- The focus of the application production will be on implementing testing

and maintaining target technology.

3. MODERN PROCESS TRANSITIONS

Indications of a successful project transition to a modern culture

Several indicators are available that can be observed in order to distinguish projects that have made a genuine cultural transition from projects that only pretends. The following are some rough indicators available.

The lower-level managers and the middle level managers should participate in the project development

Any organization which has an employee count less than or equal to 25 does not need to have pure managers. The responsibility of the managers in this type of organization will be similar to that of a project manager. Pure managers are needed when personal resources exceed 25. Firstly, these managers understand the status of the project them, develop the plans and estimate the results. The manager should participate in developing the plans. This transition affects the software project managers.

Tangible design and requirements

The traditional processes utilize tons of paper in order to generate the documents relevant to the desired project. Even the significant milestones of a project are expressed via documents. Thus, the traditional process spends most of their crucial time on document preparation instead of performing software development activities.

An iterative process involves the construction of systems that describe the architecture, negotiates the significant requirements, identifies and resolves the

risks etc. These milestones will be focused by all the stakeholders because they show progressive deliveries of important functionalities instead of documental descriptions about the project. Engineering teams will accept this transition of environment from to less document-driven while conventional monitors will refuse this transition.

Assertive Demonstrations are prioritized

The design errors are exposed by carrying-out demonstrations in the early stages of the life cycle. The stake holders should not over-react to these design errors because overemphasis of design errors will discourage the development organizations in producing the ambitious future iterating. This does not mean that stakeholders should bare all these errors. Infact, the stakeholders must follow all the significant steps needed for resolving these issues because these errors will sometimes lead to serious down-fall in the project.

This transition will unmark all the engineering or process issues so, it is mostly refused by management team, and widely accepted by users, customers and the engineering team.

The performance of the project can be determined earlier in the life cycle.

The success and failure of any project depends on the planning and architectural phases of life cycle so, these phases must employ high-skilled professionals. However, the remaining phases may work well an average team.

Earlier increments will be adolescent

The development organizations must ensure that customers and users should not expect to have good or reliable deliveries at the initial stages. This can be done by demonstration of flexible benefits in successive increments. The demonstration is similar to that of documentation but involves measuring of changes, fixes and upgrades based on the objectives so as to highlight the process quality and future environments.

Artifacts tend to be insignificant at the early stages but proves to be the most significant in the later stages

The details of the artifacts should not be considered unless a stable and a useful baseline is obtained. This transition is accepted by the development team while the conventional contract monitors refuse this transition.

Identifying and Resolving of real issues is done in a systematic order

The requirements and designs of any successful project arguments along with the continuous negotiations and trade-offs. The difference between real and apparent issued of a successful project can easily be determined. This transition may affect any team of stakeholders.

Everyone should focus on quality assurance

The software project manager should ensure that quality assurance is integrated in every aspect of project that is it should be integrated into every individuals role, every artifact, and every activity performed etc. There are some organizations which maintains a separate group of individuals know as quality assurance team, this team would perform inspections, meeting and checklist in order to measure quality assurance. However, this transition involves replacing of separate quality assurance team into an organizational teamwork with mature process, common objectives and common incentives. So, this transition is supported by engineering teams and avoided by quality assurance team and conventional managers.

CASE STUDIES:

COCOMO MODEL

The best known and most transparent cost model COCOMO (Constructive costmodel) was developed by Boehm, which was derived from the analysis of 63 software projects. Boehm proposed three levels of the model: basic, intermediate and detailed. COCOMO focuses mainly upon the intermediate mode.
The COCOMO model is based on the relationships between:
Equation 1:- Development effort is related to system size

$$MM = a.KDSI.b$$

Equation 2:- Effort and development time

$$TDEV = c.MM.d$$

where MM is the effort in man-months.

KDSI is the number of thousand delivered source instructions.
TDEV is the development time.
The coefficients a, b, c and d are dependent upon the 'mode of development which Boehm classified into 3 distinct modes:

1. Organic - Projects involve small teams working in familiar and stable environments.
Eg: - Payroll systems.
2. Semi - Detached - Mixture of experience within project teams. This lies in between organic and embedded modes.
Eg: Interactive banking system.
3. Embedded: - Projects that are developed under tight constraints, innovative, complex and have volatility of requirements.
Eg: - nuclear reactor control systems.
Development mode A B C D
Organic 3.2 1.05 2.5 0.38

Semi-detached 3.0 1.12 2.5 0.35
Embedded 2.8 1.20 2.5 0.32

In the intermediate mode it is possible to adjust the nominal effort obtained from the model by the influence of 15 cost drivers. These drivers deviate from the nominal figures, where particular project differ from the average project. For example, if the reliability of the software is very high, a factor rating of 1.4 can be assigned to that driver. Once all the factors for each driver have been chosen they are multiplied to arrive at an Effort Adjustment Factor (EAF).
The actual steps in producing an estimate using the intermediate COCOMO model are:
1. Identify the 'mode' of development for the new project.
2. Estimate the size of the project in KDSI to derive a nominal effort prediction.
3. Adjust the 15 cost drivers to reflect your project, producing an error adjustment factor.
4. Calculate the predicted project effort using equation 1 and the effort adjustment factor.
5. Calculate the project duration using equation 2.

Drawbacks:
1. It is hard to accurately estimate KDSI early on in the project, when most effort estimates are required.
2. Extremely vulnerable to mis-classification of the development mode.
3. Success depends largely on tuning the model to the needs of the organization, using historical data which is not always available.

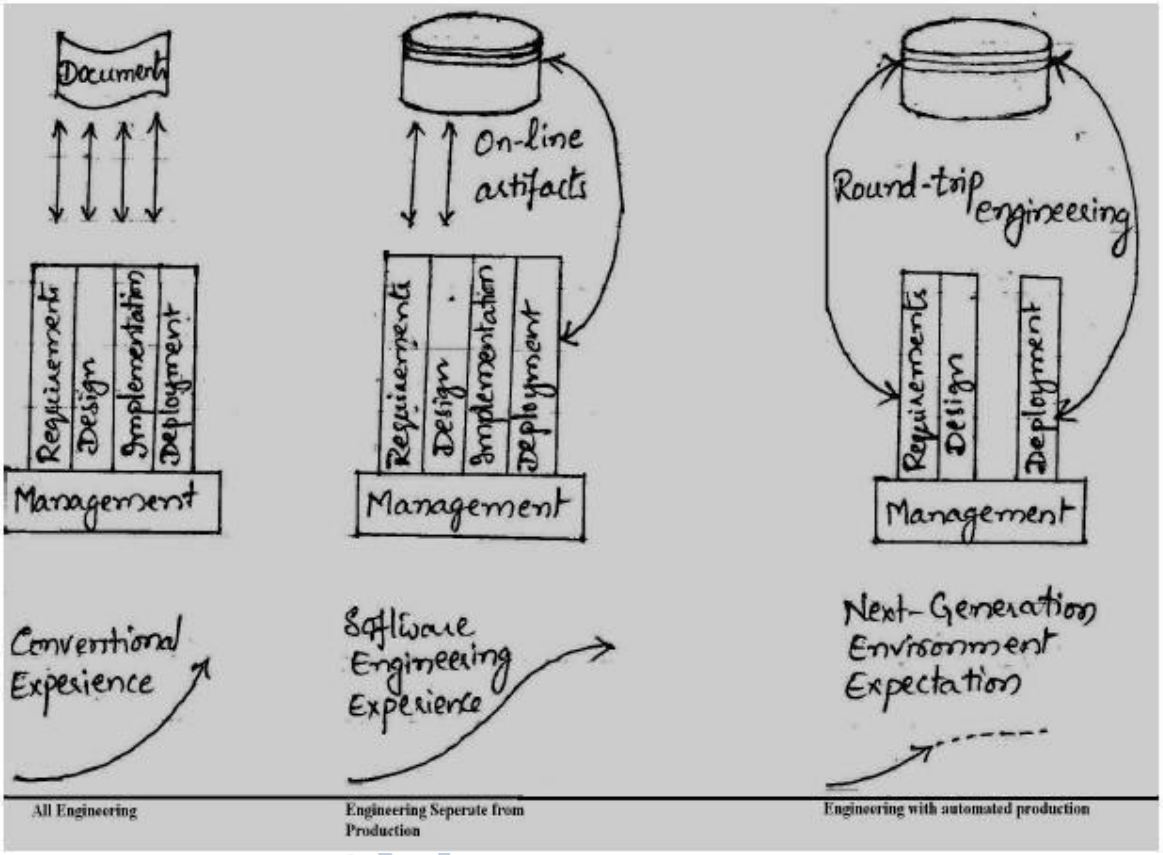
Advantages:
1. COCOMO is transparent. It can be seen how it works.
2. Drivers are particularly helpful to the operator to understand the impact of different factors that affect project costs.

Engineering Stage	Production Stage
Risk resolution, low-fidelity plan	Low-risk, high-fidelity plan
Schedule/ technology-driven	cost-driven
Risk sharing contracts/funding	Fixed-price contracts/ funding.
N-month design phase	M-month production increments
Team Size Architecture: small team of s/w engineers Application: small team of domain engineers Small and expert as possible	**Team Size** Architecture: small team of s/w engineers Application: as many as needed Large and diverse as needed
Product Executable architecture Production plans Requirements	**Product** Deliverable, useful function Tested baselines Warranted quality
Focus Design and integration Host development environment	**Focus** Implement, test and maintain Target technology
Phases	**Phases**
Inception and elaboration	Construction ad transition

Phases Phases

Inception and elaboration Construction ad transitionArchitecture and applications have different units of mass-scale and size. Scale ismeasured in terms of architecturally significant elements such as classes, components, processes and nodes. Size is measured in SLOC or megabyte of executable code. Next generation environments and infrastructures are moving to automate and standardize many of the management activities, thereby requiring a lower percentage of effort for overhead activities as scale increases. The two major improvements in next-generation cost estimation models are.

1. Separation of the engineering stage from the production stage to differentiate between architectural scale and implementation size.
2. Rigorous design notations such as UML to be more standardized. The automation of the construction process in next-generation environments is shown below.


Document
Requirement
Design
Implementation
Deployment
Management
Conventional Experience
All Engineering
On-line artifacts
Requirements
Design
Implementation
Deployment
Management
Software Engineering Experience
Engineering Seperate from Production
Round-trip engineering
Requirements
Design
Deployment
Management
Next-Generation Environment Expectation
Engineering with automated production