Software Project Management

Course Contents:

Unit 01: **Software Management Practice and Software Economics**

1.1 Conventional Software Management Theory and Practice
1.2 Software Economics and Cost Estimation
1.3 Improving Software Economics
1.4 Software Process
1.5 Team Effectiveness and Software Environment & Quality Target
1.6 Principles of Conventional Software Engineering
1.7 Principles of Modern Software Management
1.8 Iterative Process

Project

- An individual or collaborative enterprise that is carefully planned to achieve a particular aim.
- A **project** is a series of tasks that need to be completed in order to reach a specific outcome.
- A **project** can also be defined as a set of inputs and outputs required to achieve a particular goal.
- **Projects** can range from simple to complex and can be managed by one person or a hundred.

Examples of **Project**:

- Planning a large reception or an event, that is a **project**.
- This is because, it was a specific reception for a specific reason and It was held on a specific date and time.
- That means reception was unique, temporary, and it had a defined beginning and end, and reception created a specific product or service to invited persons.

Classification of Projects.

- There are many ways to classify a project such as:
  - By size (cost, duration, team, business value, number of departments affected, and so on)
  - By type (new, maintenance, upgrade, **strategic**, tactical, operational)

Why Projects Fail?

- Not enough resources
- Not enough time
- Unclear specifications
- Changes in scope
- Disagreement among stakeholders
- Bad Plan
- Lack of project management

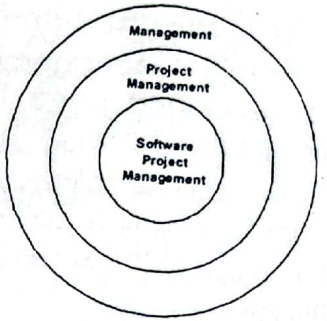
Software Project

- A Software Project is the complete procedure of software development from requirement gathering to testing and maintenance, carried out according to the execution methodologies, in a specified period of time to achieve intended software product.

Has four Ps:

- People – Organized and effective, highly motivated and coordinated team to do quality work & to achieve effective communication.
- Product – Product requirements from customer, portioned and position for work
- Process – Roadmap adopted to the people & problem
- Project – Organized to get succeed

Software Project Management



Factors affecting Software Projects

Table 1.2 Summary of Factor Rankings for Successful, Challenged, and Impaired Projects

Rank	Factors for Successful Projects	Factors for Challenged Projects	Factors for Impaired Projects
1	User involvement	Lack of user input	Incomplete requirements
2	Executive management support	Incomplete requirements	Lack of user involvement
3	Clear statement of requirements	Changing requirements & specifications	Lack of resources
4	Proper planning	Lack of executive support	Unrealistic expectations
5	Realistic expectations	Technology incompetence	Lack of executive support
6	Smaller project milestones	Lack of resources	Changing requirements & specifications
7	Competent staff	Unrealistic expectations	Lack of planning
8	Ownership	Unclear objectives	Didn't need it any longer
9	Clear vision & objectives	Unrealistic time frames	Lack of IT management
10	Hard working, focused team	New technology	Technology illiteracy

Source: Adapted from The Standish Group, *CHAOS* (West Yarmouth, MA, 1995), http://www.standishgroup.com/visitor/chaos.htm

Software Project Management

- It is a process of managing, allocating and timing resourrces to develope computer software that meets requirements.
- In Software Project Management, the end users and developers need to know the length, duration and cost of the project.
- SPM is a sub-discpline of project management in which software project are planned, implemented, monitored and controlled.

- It consists of eight tasks:
  - Problem Identification
  - Problem Defining
  - Project Planning
  - Project Organization
  - Resource Allocation
  - Project Scheduling
  - Tracking, Reporting and Controlling
  - Project Transmission
- Advantages of SPM: Easy manage a company's projects; Accessibility and Cost

•12

- So, Software Project Management is the art and science of planning and leading SOFTWARE PROJECT.
- In problem ieentifiction and definition, the decisions are made while approving, declining or prioritizing projects.
- In problem identification, project is identified, defined and justified.
- In problem definition, the purpose of the project is clarified. The main product is project proposal.
- In project planning, it describes a series of actions or steps that are needed to for the development of work product.

•13

- In project organization, the functions of the personnel are integrated. It is done in parallel with project planning.
- So that, Project Management Software is Software is used for Project Planning, Schedulling, Resource allocation, and change management.
- The important of SPM – is a kinds of tools allow companies to become competitive in their environments, optimising time and effort and keeping project on track.
- The main objective of SPM is to deliver an information system that is acceptable to users and is developed on time and within budget.

- So, Software Project Managements is an umbrella activity within Software Engineering

- Project Management involves the Planning, Monitoring and Control of People, Process and Events that occur as Software evolves from preliminary concept to an operational implementation.

- Project Management begins before any technical activity and continues throughout the Definition, Development and Support of Computer Software.

Conventional Software Management Theory and Practice

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:

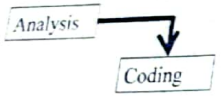
WATERFALLL MODEL

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

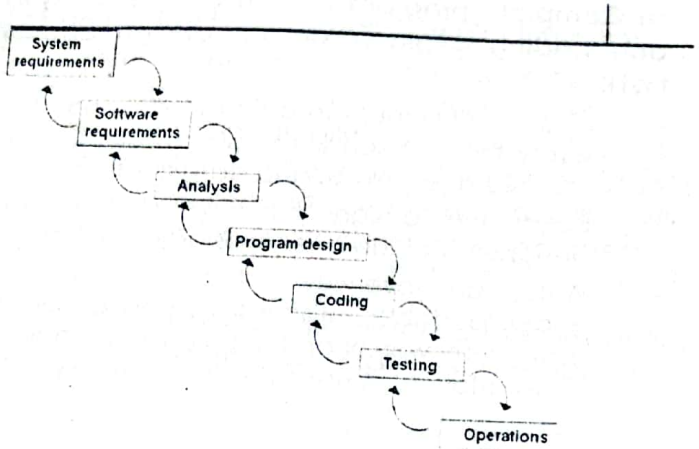
1. There are two essential steps common to the development of computer programs:
   - analysis
   - coding



- Analysis and Coding both involve creative work that directly contributes to the usefulness of the end product.

Waterfall Model Part I: 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

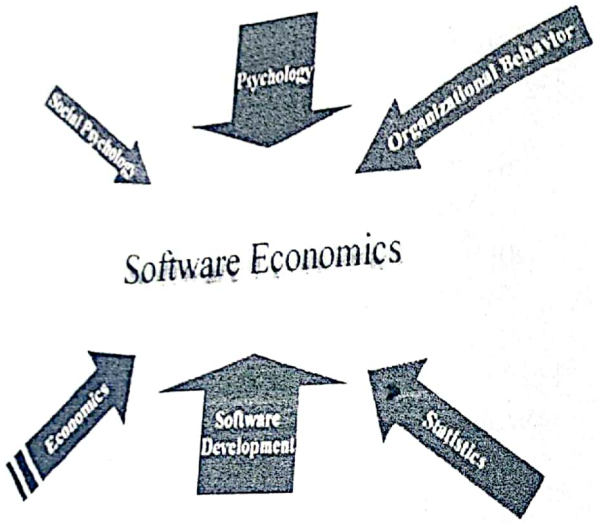
- Whatever the advices that are given by the software developers and the theory behind the waterfall model, some software projects still practice the conventional software management approach.
- Projects intended for trouble frequently exhibit the following symptoms:
  - i) Protracted (delayed) integration
- 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.

- Here the testing activities consume 40% or more life-cycle resources.

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

Software Economics and Cost Estimation

- Software economics is the study of how scarce project resources are allocated for software projects.
- Software economics helps software managers allocate those resources in the most efficient manner.
- Software Economics includes many different disciplines which are shown below:



- Software Economics examines the entire idea of software development and it includes many different disciplines such as:

- **Psychology** - focuses on the study of behavior and the reward/punishment model. "What get's rewarded gets done."

- **Social Psychology** - focuses on how people behave in an organization, quality of work life, and peer pressures.

- **Organizational Behavior** - is the process of analyzing the structure of an organization to understand those structural issues impacting organizational productivity and quality.

- **Economics** - the study of prices (components), costs, and scarcity.

- **Statistics** - deals with quantitative and qualitative techniques for the collection of data, how data is analyzed, and how results are presented.

- **Increasing Marginal Cost**
  - As the size of a software project the unit cost (or average cost) rises.
  - In other words software has increasing marginal costs and there are few economies of scale when developing a software application.
  - Any large engineering or construction project follows this same economic model.
  - Hours per function point (average costs) go up as project and organizational size increase.

- **Diseconomies of Scale**
  - In all software projects there are some basic principles which cause diseconomies of scale. That is:
  - There are low fixed costs relative to variable costs
  - Communication becomes difficult as project becomes larger
  - Multiple logical paths grow in a nonlinear manner as size increases
  - Interrelationships of functions grow geometrically as project becomes large.

Software Economics Basic Factors:

- All kinds of software cost model can be abstracted into a function of five basic factors:
  1. Size
  2. process
  3. personnel
  4. environment and
  5. required quality.

1. **Size:** of end product, quantified in terms of the number of source instructions. No. of function points.
2. **Process:** used to produce the end product (rework, bureaucratic delays, communications overheads)
3. **Personnel:** capability of software engineering

4. **Environment:** tools & techniques availability to support of efficient software development and the process.
5. **Required Quality:** product quality including features, performance reliability etc.

<u>The relationships among these parameters and the estimated cost written as:</u>

Effort=(personnel)(environment)(quality)(sizeProcess)

- There are three generations of S/W development are:
  - Conventional, Transition and Modern practices.

- It can overview return on investment (ROI) by three way:
  - Achieving ROI across line of business (successive system) : cost per unit 1st, 2nd ..... nth system.
  - Achieving ROI across a project with multiple iterations (successive iterations): 1st, 2nd, ... nth iteration.
  - Achieving ROI across a life cycle of product releases (successive Releases): 1st, 2nd, ... nth release.

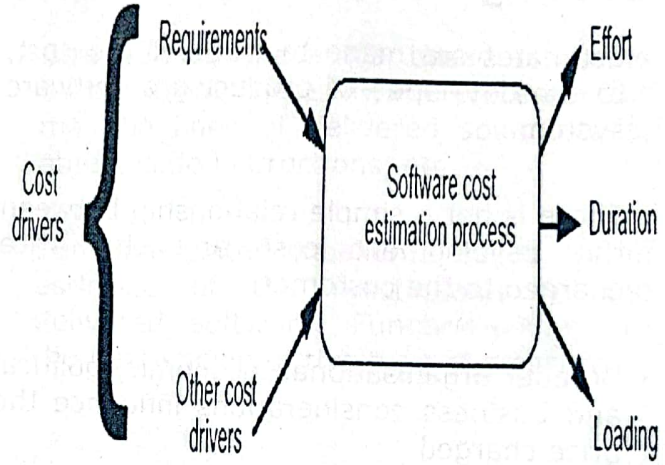
Cost Estimation

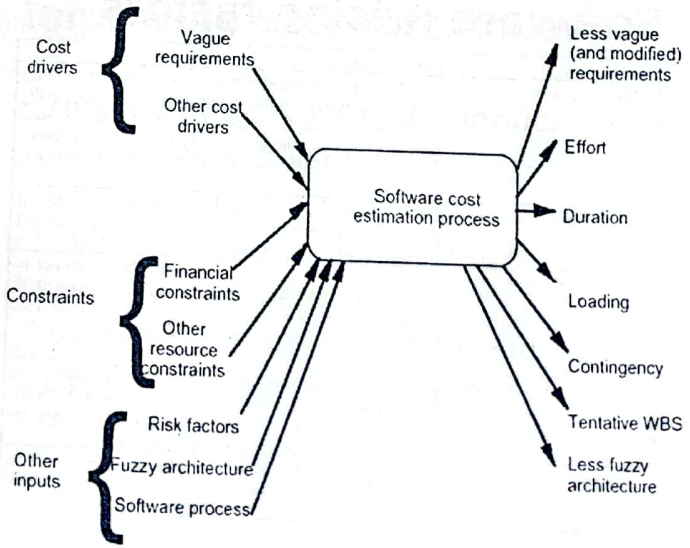
- The **costs** of development are primarily the **costs** of the effort involved, so the effort computation is used in both the **cost** and the schedule **estimate**.

- The initial **cost estimates** may be used to establish a budget for the **project** and to set a price for the **software** for a customer.

Software Cost Estimation Process

- Definition:
  - A set of techniques and procedures that is used to derive the software cost estimate.
  - Set of inputs to the process and then the process will use these inputs to generate the output.





Costing and pricing

- Estimates are made to discover the cost, to the developer, of producing a software system
- There is not a simple relationship between the development cost and the price charged to the customer
- Broader organisational, economic, political and business considerations influence the price charged

Software pricing factors

Factor	Description
Market opportunity	A development organisation may quote a low price because it wishes to move into a new segment of the software market. Accepting a low profit on one project may give the opportunity of more profit later. The experience gained may allow new products to be developed.
Cost estimate uncertainty	If an organisation is unsure of its cost estimate, it may increase its price by some contingency over and above its normal profit.
Contractual terms	A customer may be willing to allow the developer to retain ownership of the source code and reuse it in other projects. The price charged may then be less than if the software source code is handed over to the customer.
Requirements volatility	If the requirements are likely to change, an organisation may lower its price to win a contract. After the contract is awarded, high prices may be charged for changes to the requirements.
Financial health	Developers in financial difficulty may lower their price to gain a contract. It is better to make a small profit or break even than to go out of business.

Productivity measures

- Size related measures based on some output from the software process. This may be lines of delivered source code, object code instructions, etc.
- Function-related measures based on an estimate of the functionality of the delivered software. Function-points are the best known of this type of measure.

Improving Software conomics

- **Improving Software Economics:**
  - Reducing Software product size,
  - improving software processes,
  - improving team effectiveness,
  - improving automation,
  - Achieving required quality, and
  - peer inspections.

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

1. Reducing the **size** or complexity of what needs to be developed
2. Improving the development **process**
3. Using more-skilled **personnel** and better teams (not necessarily the same thing)
4. Using better **environments** (tools to automate the process)
5. Trading off or backing off on **quality** thresholds

- These parameters are given in priority order for most software domains.

- Below Table 3-1 lists some of the technology developments, process improvement efforts, and management approaches targeted at improving the economics of software development and integration.

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

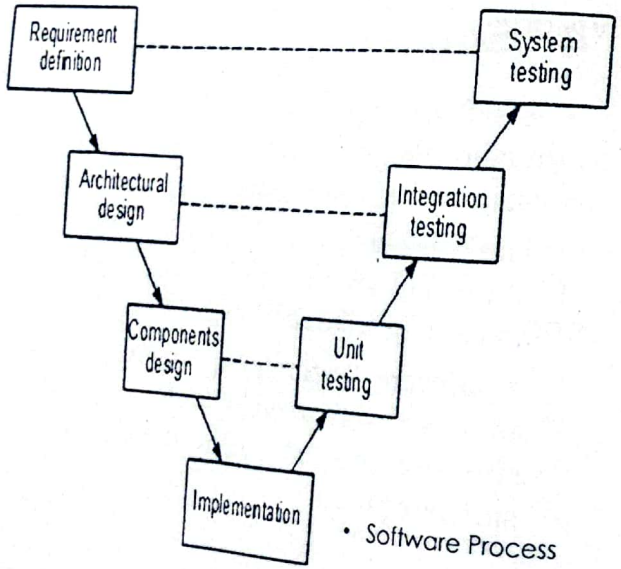
COST MODEL PARAMETERS	TRENDS
Size	Higher order languages (C++, Ada 95, Java, Visual Basic, etc.)
Abstraction and component-based development technologies	Object-oriented (analysis, design, programming)
	Reuse
	Commercial components
Process	Iterative development
Methods and techniques	Process maturity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

4. Software Process

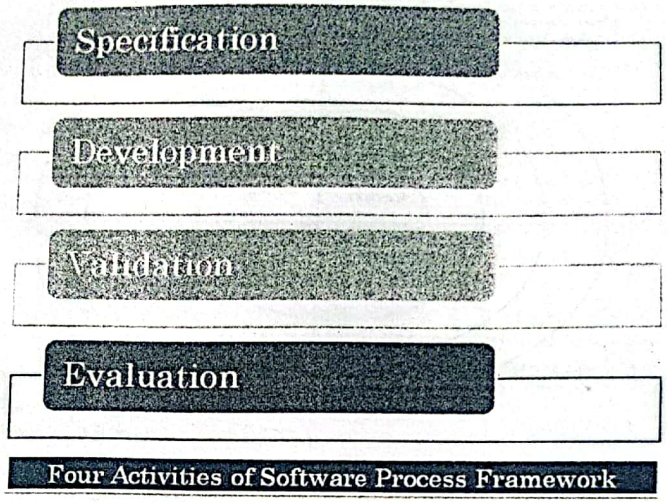
- A software process (also knows as software methodology) is a set of related activities that leads to the production of the software.
- These activities may involve the development of the software from the scratch, or, modifying an existing system.
- A **software development process**, also known as a **software development** lifecycle, is a structure imposed on the **development** of a **software** product.

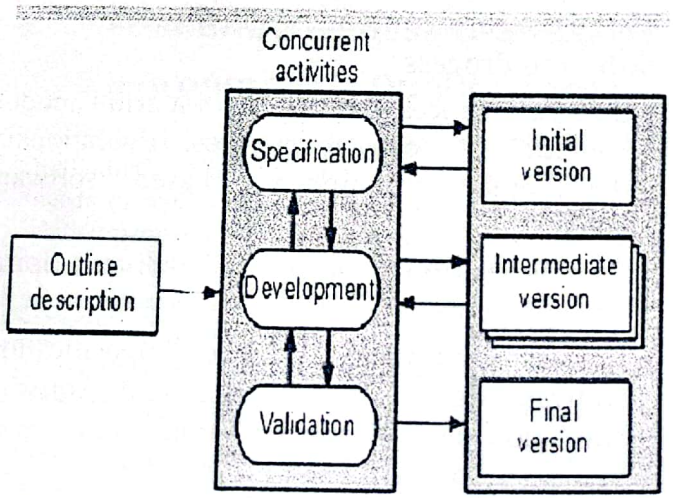
- A **software process** is represented as a set of work phases that is applied to design and build a **software** product.

- The **software** that meets the specification is produced.

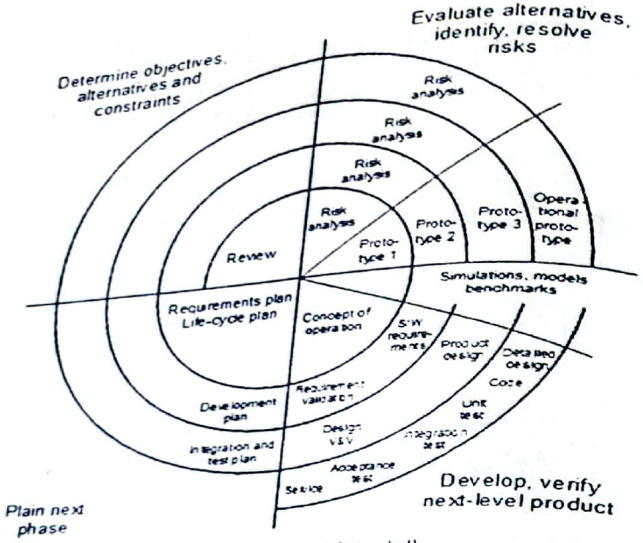


- Software Process





- Evolutionary Development



Spiral Development (Model)

There are four components includes in Software Process

- **Software Process Model:** The waterfall model, V-model of software process, Evolutionary development, Component based software engineering.
- **Process Iteration:** Incremental delivery, Spiral development,
- **Process activities:** Software Specification, software design and development, software validation and software evolution.
- The Rational Unified Process

- **Three level of Software Process:**

Three levels of process and their attributes

ATTRIBUTES	METAPROCESS	MACROPROCESS	MICROPROCESS
Subject	Line of business	Project	Iteration
Objectives	Line-of-business profitability Competitiveness	Project profitability Risk management Project budget, schedule, quality	Resource management Risk resolution Milestone budget, schedule, quality
Audience	Acquisition authorities, customers Organizational management	Software project managers Software engineers	Subproject managers Software engineers
Metrics	Project predictability Revenue, market share	On budget, on schedule Major milestone success Project scrap and rework	On budget, on schedule Major milestone progress Release/iteration scrap and rework
Concerns	Bureaucracy vs. standardization	Quality vs. financial performance	Content vs. schedule
Time scales	6 to 12 months	1 to many years	1 to 6 months

5. Team Effectiveness & Software Environment & Quality Target

- Teamwork is much more important than the sum of the individuals. With software teams, a project manager needs to configure a balance of solid talent with highly skilled people in the leverage positions. Some maxims of team management include the following:
  - A well-managed project can succeed with a nominal engineering team.
  - A mismanaged project will almost never succeed, even with an expert team of engineers.
  - A well-architected system can be built by a nominal team of software builders.
  - A poorly architected system will flounder even with an expert team of builders.

Boehm five staffing principles are -

- The principle of top talent: Use better and fewer people
- The principle of job matching: Fit the tasks to the skills and motivation of the people available.
- The principle of career progression: An organization does best in the long run by helping its people to self-actualize.
- The principle of team balance: Select people who will complement and harmonize with one another
- The principle of phase-out: Keeping a misfit on the team doesn't benefit anyone

- Software project managers need many leadership qualities in order to enhance team effectiveness.
- The following are some crucial attributes of successful software project managers that deserve much more attention:

1. **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.
2. **Customer-interface skill.** Avoiding adversarial relationships among stakeholders is a prerequisite for success.

3. **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.

4. **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.

5. **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

- **Software environment** is the term commonly used to refer to support an application.
- A **software environment** for a particular application could include the operating system, the database system, specific development tools or compiler.

Quality Target

- Software best practices are derived from the development process and technologies.
- Key practices that improve overall software quality include the following:
  - Focusing on driving requirements and critical use cases early in the life cycle, focusing on requirements completeness and traceability late in the life cycle, and focusing throughout the life cycle on a balance between requirements evolution, design evolution, and plan evolution
  - Using metrics and indicators to measure the progress and quality of an architecture as it evolves from a high-level prototype into a fully compliant product
  - Providing integrated life-cycle environments that support early and continuous configuration control, change management, rigorous design methods, document automation, and regression test automation
  - Using visual modeling and higher level languages that support architectural control, abstraction, reliable programming, reuse, and self-documentation
  - Early and continuous insight into performance issues through demonstration-based evaluations

- Conventional development processes stressed early sizing and timing estimates of computer program resource utilization. However, the typical chronology of events in performance assessment was as follows
  - **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.
  - **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.

6. Principles of Conventinal Software Engineering

1. Make quality
2. High-quality software is possible.
3. Give products to customers early.
4. Determine the problem before writing the requirements.
5. Evaluate design alternatives.
6. Use an appropriate process model.
7. Use different languages for different phases.
8. Minimize intellectual distance.
9. Put techniques before tools.
10. Get it right before you make it faster.
11. Inspect code.
12. Good management is more important than good technology.
13. People are the key to success.
14. Follow with care.
15. Take responsibility.
16. Understand the customer's priorities.
17. The more they see, the more they need.
18. Plan to throw one away.
19. Design for change.
20. Design without documentation is not design.
21. Use tools, but be realistic.
22. Avoid tricks.
23. Encapsulate.
24. Use coupling and cohesion.
25. Use the McCabe complexity measure.

26. Don't test your own software.
27. Analyze causes for errors.
28. Realize that software's entropy increases.
29. People and time are not interchangeable.
30. Expect excellence.

7. Principles of Modern Software Management

- Top 10 principles of modern software management are. (The first five, which are the main themes of my definition of an iterative process, are summarized here.

1. **Base the process on an architecture-first approach.** This requires that a demonstrable balance be achieved among the driving requirements, the architecturally significant design decisions, and the life-cycle plans before the resources are committed for full-scale development.

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

- With today's sophisticated software systems, it is not possible to define the entire problem, design the entire solution, build the software, and thentest the end product in sequence.
- Instead, an iterative process that refines the problem understanding, an effective solution, and an effective plan over several iterations encourages a balanced treatment of all stakeholder objectives.
- Major risks must be addressed early to increase predictability and avoid expensive downstream scrap and rework.

3. **Transition design methods to emphasize component-based development.** Moving from a line-of-code mentality to a component-based mentality is necessary to reduce the amount of human-generated source code and custom development.
4. **Establish a change management environment.** The dynamics of iterative development, including concurrent workflows by different teams working on shared artifacts, necessitates objectively controlled baselines

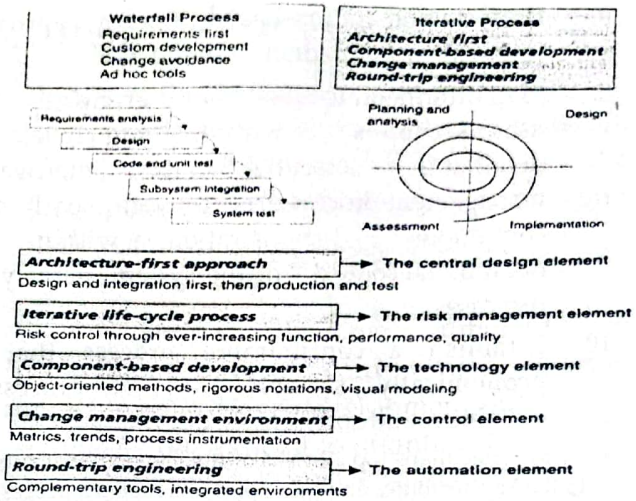


FIGURE 4-1. *The top five principles of a modern process*

5. **Enhance change freedom through tools that support *round-trip engineering*.** Round-trip engineering is the environment support necessary to automate and synchronize engineering information in different formats(such as requirements specifications, design models, source code, executable code, test cases).
6. **Capture design artifacts in rigorous, *model-based notation*.** A model based approach (such as UML) supports the evolution of semantically rich graphical and textual design notations.
7. **Instrument the process for *objective quality control* and progress assessment.** Life-cycle assessment of the progress and the quality of all intermediate products must be integrated into the process.

8. Use a demonstration-based approach to assess intermediate artifacts.

9. Plan intermediate releases in groups of usage scenarios with evolving levels of detail. It is essential that the software management process drive toward early and continuous demonstrations within the operational context of the system, namely its use cases.

10. Establish a configurable process that is economically scalable. No single process is suitable for all software developments.

- Below table maps top 10 risks of the conventional process to the key attributes and principles of a modern process

TABLE 4-1. *Modern process approaches for solving conventional problems*

CONVENTIONAL PROCESS: TOP 10 RISKS	IMPACT	MODERN PROCESS: INHERENT RISK RESOLUTION FEATURES
1. Late breakage and excessive scrap/rework	Quality, cost, schedule	Architecture-first approach Iterative development Automated change management Risk-confronting process
2. Attrition of key personnel	Quality, cost, schedule	Successful, early iterations Trustworthy management and planning
3. Inadequate development resources	Cost, schedule	Environments as first-class artifacts of the process Industrial-strength, integrated environments Model-based engineering artifacts Round-trip engineering
4. Adversarial stakeholders	Cost, schedule	Demonstration-based review Use-case-oriented requirements/testing
5. Necessary technology insertion	Cost, schedule	Architecture-first approach Component-based development
6. Requirements creep	Cost, schedule	Iterative development Use case modeling Demonstration-based review
7. Analysis paralysis	Schedule	Demonstration-based review Use-case-oriented requirements/testing
8. Inadequate performance	Quality	Demonstration-based performance assessment Early architecture performance feedback
9. Overemphasis on artifacts	Schedule	Demonstration-based assessment Objective quality control
10. Inadequate function	Quality	Iterative development Early prototypes, incremental releases

8. Iterative Process

- Modern software development processes have moved away from the conventional waterfall model, in which each stage of the development process is dependent on completion of the previous stage.

- The economic benefits inherent in transitioning from the conventional waterfall model to an iterative development process are significant but difficult to quantify.

- As one benchmark of the expected economic impact of process improvement, consider the process exponent parameters of the COCOMO II model. (Appendix B provides more detail on the COCOMO model) This exponent can range from 1.01 (virtually no diseconomy of scale) to 1.26 (significant diseconomy of scale).

- The parameters that govern the value of the process exponent are application precedentedness, process flexibility, architecture risk resolution, team cohesion, and software process maturity.

- The following paragraphs map the process exponent parameters of CO COMO II to top 10 principles of a modern process.
- **Application precedentedness.**
  - Domain experience is a critical factor in understanding how to plan and execute a software development project.
  - For unprecedented systems, one of the key goals is to confront risks and establish early precedents, even if they are incomplete or experimental.
  - This is one of the primary reasons that the software industry has moved to an ***iterative life-cycle process.***
  - Early iterations in the life cycle establish precedents from which the product, the process, and the plans can be elaborated in ***evolving levels of detail.***

- **Process flexibility.**
  - Development of modern software is characterized by such a broad solution space and so many interrelated concerns that there is a paramount need for continuous incorporation of changes.
  - These changes may be inherent in the problem understanding, the solution space, or the plans. Project artifacts must be supported by efficient ***change management*** commensurate with project needs.
  - A ***configurable process*** that allows a common framework to be adapted across a range of projects is necessary to achieve a software return on investment.

- **Architecture risk resolution.**
  - ***Architecture-first*** development is a crucial theme underlying a successful iterative development process.
  - A project team develops and stabilizes architecture before developing all the components that make up the entire suite of applications components.
  - An ***architecture-first*** and ***component-based development approach*** forces the infrastructure, common mechanisms, and control mechanisms to be elaborated early in the life cycle and drives all component make/buy decisions into the architecture process.

- **Team cohesion.**
  - Successful teams are unified, and integrated teams are successful. Successful teams and unified or integrated teams share common objectives and priorities.
  - Advances in technology (such as programming languages, UML, and visual modeling) have enabled more rigorous and understandable notations for communicating software engineering information, particularly in the requirements and design artifacts that previously were ad hoc and based completely on paper exchange.
  - The ***model-based*** formats have also enabled the ***round-trip engineering*** support needed to establish change freedom sufficient for evolving design representations.

- **Software process maturity.**
  - The Software Engineering Institute's Capability Maturity Model (CMM) is a well-accepted benchmark for software process assessment.
  - One of key themes is that truly mature processes are enabled through an integrated environment that provides the appropriate level of automation to instrument the process for *objective quality control.*

Unit – Important questions

1. Explain briefly Waterfall model. Also explain Conventional s/w management performance?
2. Define Software Economics. Also explain Pragmatic s/w cost estimation?
3. Explain Important trends in improving Software economics?
4. Explain five staffing principal offered by Boehm. Also explain Peer Inspections?
5. Explain principles of conventional software engineering?
6. Explain briefly, principles of modern software management

Thank you

Software Project Management

Course Contents:

Unit 02: Software Process Primitives and Process Management Framework

2.1 Software Process Life Cycle Phases

2.2 Various Elements of the Software Process (Management, Engineering and Pragmatic)

2.3 Technical and Management Perspective of Software Architecture

2.4 Software Process Workflow and Iteration Workflow

2.5 Status Monitoring – Software Process Checkpoints and Milestone

Software Process Life Cycle Phases

- **Software life cycle** models describe **phases** of the **software cycle** and the order in which those **phases** are executed.
- Each **phase** produces deliverables required by the next **phase** in the **life cycle**.
- Requirements are translated into design.
- Code is produced according to the design which is called **development phase**

Life-Cycle Phases

- Engineering and Production Stages
- Inception Phase
- Elaboration Phase
- Construction Phase
- Transition Phase

Life-Cycle Phases [Engineering and Production Phase/Stages]

❑ **Two stages of the life-cycle :**

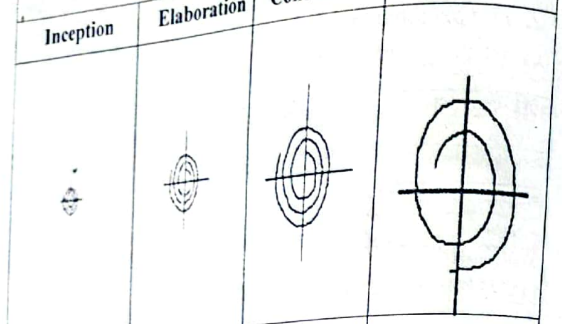
1. ***The engineering stage*** – driven by smaller teams doing design and synthesis activities
2. ***The production stage*** – driven by larger teams doing construction, test, and deployment activities

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

- The most conventional life cycles, the phases are named after the primary activity within each phase :
  - Requirement analysis, design, coding, unit test, integrationtest, and system test

- So,conventional software development efforts sequential process, in which one activity was required to be complete before the next was begun.
- Attributing only two stages to a life cycle is too coarse

The phases of the Life Cycle-Process

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

Inception Phase

- ☐ Overriding goal – to achieve concurrence among stakeholders on the life-cycle objectives
- ☐ Essential activities :
  - ➢ Formulating the scope of the project (capturing the requirements and operational concept in an information repository)
  - ➢ Synthesizing the architecture (design trade-offs, problem space ambiguities, and available solution-space assets are evaluated)
  - ➢ Planning and preparing a business case (alternatives for risk management, iteration planes, and cost/schedule/profitability trade-offs are evaluated).

Elaboration Phase

- ☐ During the elaboration phase, an executable architecture prototype is built
- ☐ Essential activities :
  - ➢ *Elaborating the vision* (establishing a high-fidelity understanding of the critical use cases that drive architectural or planning decisions)
  - ➢ *Elaborating the process and infrastructure* (establishing the construction process, the tools and process automation support)
  - ➢ *Elaborating the architecture and selecting components* (lessons learned from these activities may result in redesign of the architecture)

Construction Phase

- ☐ During the construction phase :
  All remaining components and application features are integrated into the application
  All features are thoroughly tested
- ☐ Essential activities :
  - ➢ Resource management, control, and process optimization
  - ➢ Complete component development and testing against evaluation criteria
  - ➢ Assessment of the product releases against acceptance criteria of the vision

Transition Phase

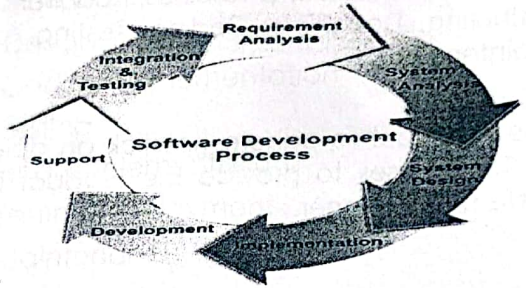
- The transition phase is entered when baseline is mature enough to be deployed in the end-user domain
- This phase could include beta testing, conversion of operational databases, and training of users and maintainers

- Essential activities :
  - Synchronization and integration of concurrent construction into consistent deployment baselines
  - Deployment-specific engineering (commercial packaging and production, field personnel training)

- **Evaluation Criteria :**
  - Is the user satisfied?
  - Are actual resource expenditures versus planned expenditures acceptable?

- Each of the four phases consists of one or more iterations in which some technical capability is produced in demonstrable form and assessed against a set of the criteria.
- The transition from one phase to the nest maps more to a significant business decision than to the completion of specific software activity.

- So, in software engineering, a software development process is the process of dividing software development work into distinct phases to improve design, product management, and project management. It is also known as a software development life cycle.



- SDLC is a process which defines the various stages involved in the development of software for delivering a high-quality product.
- SDLC stages cover the complete life cycle of a software i.e. from inception to retirement of the product.
- Adhering to the SDLC process leads to the development of the software in a systematic and disciplined manner.

Purpose of SDLC:

- Purpose of SDLC is to deliver a high-quality product which is as per the customer's requirement.

- SDLC has defined its phases as, Requirement gathering, Designing, Coding, Testing, and Maintenance.

- It is important to adhere (to stick an object) to the phases to provide the Product in a systematic manner.

For Example:

- A software has to be developed and a team is divided to work on a feature of the product and is allowed to work as they want.
- One of the developers decides to design first whereas the other decides to code first and the other on the documentation part.
- This will lead to project failure because of which it is necessary to have a good knowledge and understanding among the team members to deliver an expected product.

Given below are the various phases of SDLC:

- Requirement gathering
- Analysis
- Design
- Coding/Implementation
- Testing
- Final Implementation
- Testing/Deployment
- Maintenance

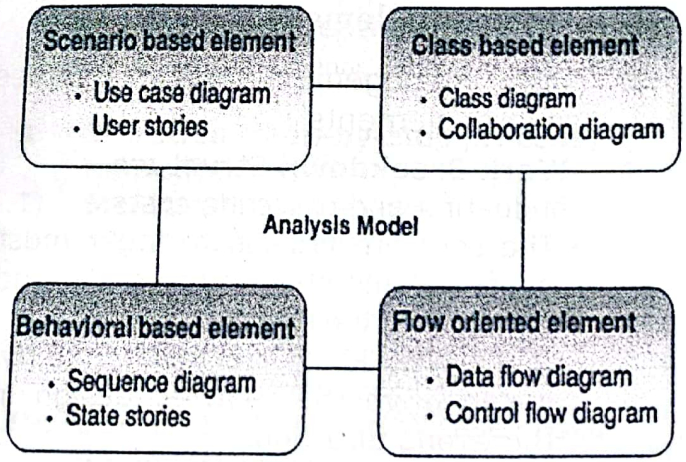


Fig. - Elements of analysis model

Various Elements of the Software Process

- Management elements
- Engineering elements
- Pragmatic elements

Management Elements

- The management set includes several elements :
  - **Work Breakdown Structure** – budgeting and collecting costs.
    - The software project manager must have insight into project costs and how they are expended.
    - If the WBS is structured improperly, it can drive the evolving design in the wrong direction.

- **Business Case** – provides all the information necessary to determine whether the project is worth investing in.
  - It details the expected revenue, expected cost, technical and management plans.

- **Release Specifications**

Typical release specification outlines are:

I. **Iteration (step-by-step process) content**
II. **Measurable objectives**
   A. Evaluation criteria
   B. Follow-through approach
III. **Demonstration plan**
   A. Schedule of activities
   B. Team responsibilities
IV. **Operational scenarios (use cases demonstrated)**
   A. Demonstration procedures

➢**Software Development Plan** – the defining document for the project's process.

- It must comply with the contract, comply with the organization standards, evolve along with the design and requirements.

➢**Deployment** – depending on the project, it could include several document subsets for transitioning the product into operational status.

- It could also include computer system operations manuals, software installation manuals, plans and procedures for cutover etc.

➢**Environment** – A robust development environment must support automation of the development process.

- It should include : requirements management visual modeling document automation automated regression testing

Engineering Element of Software Process

- ❑ **Software process**
- ❑ **Reuse:** Not just of software, but also sets of requirements, parts of designs, or group of test scripts.
- ❑ **Measurement:** Trying to quantify project goals to evaluate progress (for example – number of bugs per 100 lines of code)
- ❑ **Tools and Integrated Environment:** For example, CASE (Computer-aided software engineering) tools.

➢ **Architecture Description** – it is extracted from the design model and includes views of the design, implementation, and deployment sets sufficient to understand how the operational concept of the requirements set will be achieved.

➢ **Software User Manual** – it should include installation procedures, usage procedures and guidance, operational constraints, and a user interface description. (writteb by test team)

Pragmatic Elements

- Over the past 30 years, the quality of documents become more important than the quality of the engineering information they represented.
  - The reviewer must be knowledgeable in the engineering notation.
  - Human-readable engineering artifacts should use rigorous notations that are complete, consistent, and used in a self-documenting manner.
  - Paper is tangible, electronic artifacts are too easy to change.
  - Short documents are more useful than long ones.

Artifacts of the Process

Requirements Set	Design Set	Implementation Set	Deployment Set
1.Vision document 2.Requirements model(s)	1.Design model(s) 2.Test model 3.Software architecture description	1.Source code baselines 2.Associated compile-time files 3.Component executables	1.Integrated product executable baselines 2.Associated run-time files 3.User manual

Management Set	
Planning Artifacts	**Operational Artifacts**
1.Work breakdown structure 2.Bussines case 3.Release specifications 4.Software development plan	5.Release descriptions 6.Status assessment 7.Software change order database 8.Deployment documents 9.Enviorement

Technical and Management Perspective of Software Architecture

Software Architectures: A Management Perspective

- From a management perspective, there are three different aspects of an architecture :
  - An ***architecture*** (the intangible design concept) is the design of software system, as different or similar to design of a component.
  - An ***architecture baseline*** (the tangible artifacts) is a slice (portion) of information across the engineering artifact sets sufficient to satisfy all stakeholders that the vision can be achieved within the parameters of the business case (cost, profit, time, people).

➢ An ***architecture description*** (a human-readable representation of an architecture) is an organizes subsets of information extracted from the design set model.

❑ **The importance of software architecture can be summarized as follows:**

➢Architecture representations provide a basis for balancing the trade-offs between the problem space and the solution space.

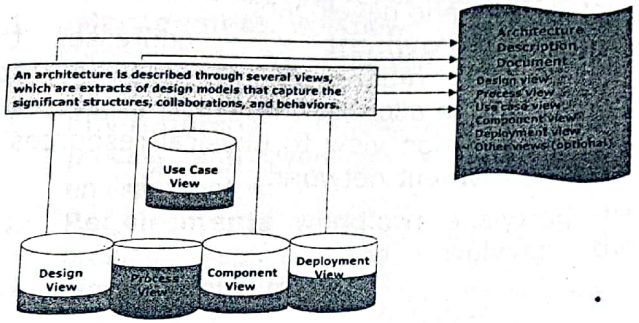
➢ Poor architectures and immature processes are often given as reasons for project failures.

➢ A mature process, an understanding of the primary requirements, and a demonstrable architecture are important prerequisites for predictable planning.

➢Architecture development and process definition are the intellectual steps that map the problem to a solution without violating the constraints.

Software Architectures: A Technical Perspective

The model which draws on the foundation of architecture developed at *Rational Software Corporation* and particularly on Philippe Kruchten's concepts of software architecture :



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

➢ The ***design view*** addresses the basic structure and the functionality of the solution.

➢ The ***process view*** addresses the run-time collaboration issues involved in executing the architecture on a distributed deployment model, including the logical software network topology, interprocess communication and state management.

➢The ***component view*** describes the architecturally significant elements of the implementation set and addresses the software source code realization of the system from perspective of the project's integrators and developers.

➢ The ***deployment view*** addresses the executable realization of the system, including the allocation of logical processes in the distribution view to physical resources of the deployment network.

Software Process Workflow and Iteration Workflow

Software Process Workflows

❑ The term *workflow* is used to mean a thread of cohesive and most sequential activities.

➢ **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

❑ Four basic key principles:

1.Architecture-first approach:
Implementing and testing the architecture must precede full-scale development and testing and must precede the downstream focus on completeness and quality of the product features.

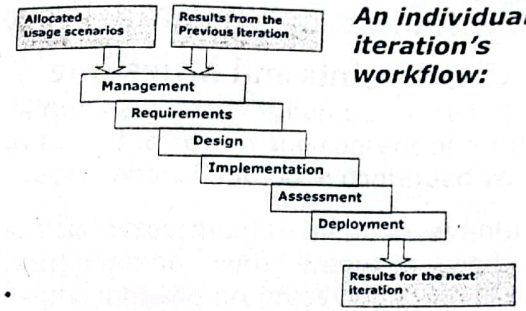
2.Iterative life-cycle process:
The activities and artifacts of any given workflow may require more than one pass to achieve adequate results.

3.Roundtrip engineering:
Raising the environment activities to a first-class workflow is critical; the environment is the tangible embodiment of the project's process and notations for producing the artifacts.

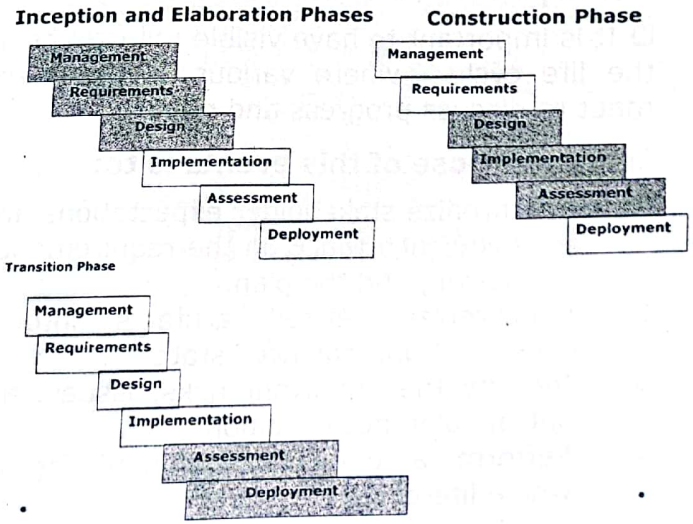
4.Demonstration-based approach:
Implementation and assessment activities are initiated nearly in the life-cycle, reflecting the emphasis on constructing executable subsets of the involving architecture.

Iteration Workflow

❑ **An iteration consist of sequential set of activities in various proportions, depending on where the iteration is located in the development cycle.**



An individual iteration's workflow:



Status Monitoring – Software Process Checkpoints and Milestone

Checkpoints of the Process

- It is important to have visible milestones in the life cycle , where various stakeholders meet to discuss progress and planes.

- **The purpose of this events is to:**

  - Synchronize stakeholder expectations and achieve concurrence on the requirements, the design, and the plan.
  - Synchronize related artifacts into a consistent and balanced state
  - Identify the important risks, issues, and out-of-rolerance conditions
  - Perform a global assessment for the whole life-cycle.

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

1. Major milestones –provide visibility to systemwide issues, synchronize the management and engineering perspectives and verify that the aims of the phase have been achieved.

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

3. Status assessments – periodic events provide management with frequent and regular insight into the progress being made.

Thank you

Software Project Management

Course Contents:

Unit 03: Techniques of Planning, Controlling and Automating Software Process

3.1 Iterative Process Planning (Process Work Breakdown Structure, Planning Guidelines, Cost and Schedule Estimation Process, Iteration Planning Process)

3.2 Project Organization and Responsibilities

3.3 Process Automation – Tools and Environment

3.4 Project Control and Process Automation

3.5 Process Customization

- **Iterative Process Planning**
  - Work Breakdown Structures
  - Planning Guidelines
  - The Cost and Schedule Estimating Process
  - The Iteration Planning Process
  - Pragmatic Planning
- **Project Organizations and Responsibilities**
  - Line-of-Business organizations
  - Project Organizations
  - Evolution Organizations
- **Process Automation**
  - Tools: Automation Building Blocks
  - The Project Environment

- **Project Control and Process Instrumentation**
  - The Seven Core Metrics
  - Management Indicators
  - Quality Indicators
  - Life-Cycle Expectations
  - Pragmatic Software Metrics
  - Metrics Automation
- **Process Atomization**
  - Process Discriminants
  - Example: Small-Scale Project Versus Large-scale Project

- **Iterative Process Planning**
  - Work Breakdown Structures
  - Planning Guidelines
  - The Cost and Schedule Estimating Process
  - The Iteration Planning Process
  - Pragmatic Planning

work breakdown structure (WBS)

- A **work breakdown structure (WBS)** is a key project deliverable that organizes the team's **work** into manageable sections.
- The **work breakdown structure** visually defines the scope into manageable chunks that a project team can understand, as each level of the **work breakdown structure** provides further definition and detail.

- The development of a work breakdown structure is dependent on the project management style, organizational culture, customer preference, financial constraints and several other hard-to-define parameters.

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

- A WBS provides the following information

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

What is process breakdown structure?

- Product **Breakdown Structure** (BBS) is a project management tool and important part of the project planning.
- The product **breakdown structure** defines subtasks or work packages and describes the relationship between work packages.
- This **process** helps to organize the projects gut and to define the project frameworks.

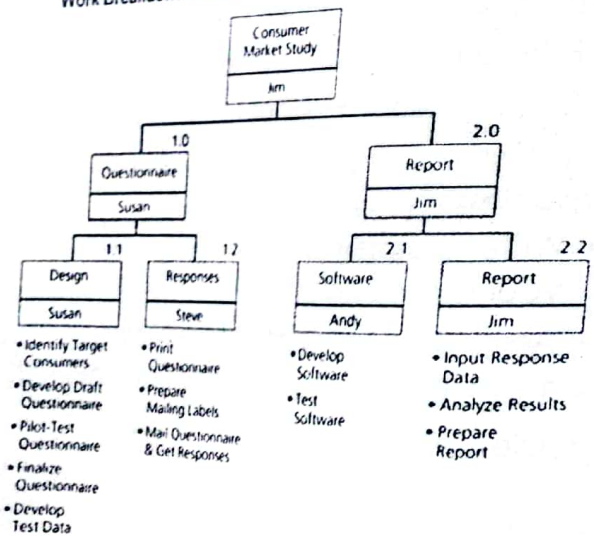
Why use a WBS in project management?

- Estimate the cost of a project.
- Establish dependencies.
- Determine a project timeline and develop a schedule.
- Write a statement of work (or SOW, one of your other acronyms).
- Assign responsibilities and clarify roles.
- Track the progress of a project.
- Identify risk.

How to Create a WBS: The High-Level View

- Determine and describe the project statement.
- Highlight all the necessary phases of the project.
- Create and list the deliverables (as well as how success will be measured)
- Divide the deliverables into manageable tasks.

Work Breakdown Structure for Consumer Market Study Project



Planning Guidelines

❑ Two simple planning guidelines should be considered when a project plan is being initiated or assessed.

FIRST-LEVEL WBS ELEMENT	DEFAULT BUDGET
Management	10%
Environment	10%
Requirements	10%
Design	15%
Implementation	25%
Assessment	25%
Deployment	5%
Total	100%

The first guideline prescribes a default allocation of costs among the first-level WBS elements

DOMAIN	INCEPTION	ELABORATION	CONSTRUCTION	TRANSITION
Effort	5%	20%	65%	10%
Schedule	10%	30%	50%	10%

The second guideline prescribes the allocation of effort and schedule across the life-cycle phases

- So, WBS will be help to our project's efficiency and effectiveness, how do we go about it? First, let's look at what all we need to get started.
- There are several inputs you will need to get you off on the right foot:
  - The Project Scope Statement
  - The Project Scope Management Plan
  - Organizational Process Assets
  - Approved Change Requests - (PMBOK Guide)

PMBOK stands for Project Management Body of Knowledge and it is the entire collection of processes, best practices, terminologies, and guidelines that are accepted as standards within the project management industry.

- These above four inputs should give you all the information you and your team needs to create your WBS. Along with these inputs, you will use certain tools as well:
  - Work Breakdown Structure Templates
  - Decomposition - (PMBOK Guide)
- Finally, using these inputs and tools you will create the following outputs:
- Work Breakdown Structure
- WBS Dictionary
- Scope Baseline
- Project Scope Statement (updates)
- Project Scope Management Plan (updates)
- Requested Changes - (PMBOK Guide)

The Cost and Schedule Estimating Process

❑ Project plans need to be derived from two perspectives.

➢ **Forward-looking:**

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

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

3. The software project manager partitions the estimate for the effort into a top-level WBS, also partitions the schedule into major milestone dates and partitions the effort into a staffing profile

4. At this point, subproject managers are given the responsibility for decomposing each of the WBS elements into lower levels using their top-level allocation, staffing profile, and major milestone dates as constraints.

➢ ***Backward-looking:***

1. The lowest level WBS elements are elaborated into detailed tasks, for which budgets and schedules are estimated by the responsible WBS element manager.

2. Estimates are combined and integrated into higher level budgets and milestones.

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

Iterative Process Planning (The Iteration Planning Process)

Engineering stage planning emphasis:

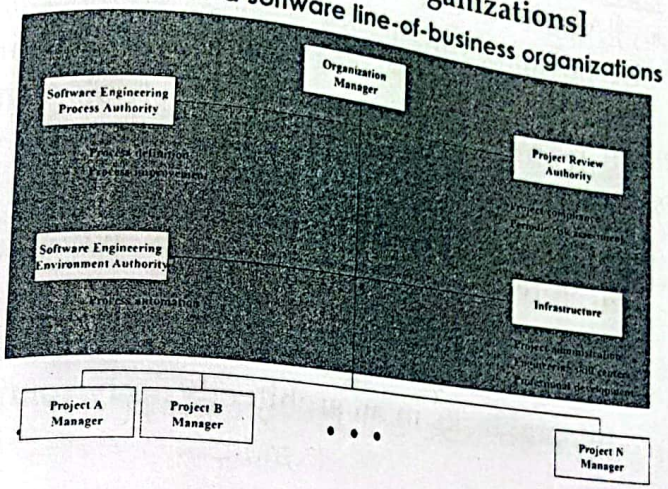
- Macro-level task estimation for production-stage artifacts
- Micro-level task estimation for engineering artifacts
- Stakeholder concurrence
- Analysis of actual vs. planned expenditures
- Tuning the top-downconcurrence
- project-independent planning guidelines into project-specific planning guidelines.

Production stage Planning emphasis:

- Micro-level task estimation for production-stage artifacts
- Macro-level task estimation for engineering artifacts
- Stakeholder concurrence
- Fine-grained variance analysis of actual vs. planned expenditures

Project Organizations and Responsibilities [Line-of-Business Organizations]

Default roles in a software line-of-business organizations



Project Organizations and Responsibilities

Project Organizations

Software Management Team

- Systems Engineering
- Financial Administration
- Quality Assurance

Artifacts

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

Responsibilities

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

Life-Cycle Focus

Inception	Elaboration	Construction	Transition
Elaboration phase planning Team formulating Contract base lining Architecture costs	Construction phase planning Full staff recruitment Risk resolution Product acceptance criteria Construction costs	Transition phase planning Construction plan optimization Risk management	Customer satisfaction Contract closure Sales support Next-generation planning

Software Architecture Team:

- "A **software architect** is a **software** expert who makes high-level design choices and dictates technical standards, including **software** coding standards, tools, and platforms."
- The most common understanding is that an **architect** is a person who takes action and dictates decisions.
- Designing, **planning** and developing are integral tasks in an architect's daily routine

Software Architecture Team

- Demonstrations
- Use-case modelers
- Design modelers
- Performance analysts

Artifacts

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

Responsibilities

- Requirements trade-offs
- Design trade-offs
- Component selection
- Initial integration
- Technical risk solution

Life-Cycle Focus

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

Component Team:

- A **team** that focuses on the creation of one or more **components** of a larger product that a customer would purchase.
- **Team** that is cross-functional (multi-disciplinary), single **component** focused. Contrast with feature **team.**
- **Component teams** : to build, deploy, and ultimately release.
- CT is capable of delivering end-to-end user value
- Each **team** has all the skills necessary to deliver a feature.

Software Development Team

- **Component teams**

Artifacts

- Design set
- Implementation set
- Deployment set

Responsibilities

- Component design
- Component implementation
- Component stand-alone test
- Component maintenance
- Component documentation

Life-Cycle Focus

Inception	Elaboration	Construction	Transition
• Prototyping support • Make/buy trade-offs	• Critical component design • Critical component implementation and test • Critical component base line	• Component design • Component implementation Component stand-alone test • Component maintenance	• Component maintenance • Component documentation

Software Assessment Team:

- 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 Assessment Team

- Release testing
- Change management
- Deployment
- Environment support

Artifacts

- Deployment set
- SCO database
- User manual
- Environment
- Release specifications
- Release descriptions
- Deployment documents

Responsibilities

- Project infrastructure
- Independent testing
- Requirements verification
- Metrics analysis
- Configuration control
- Change management
- User deployment

Life-Cycle Focus

Inception	Elaboration	Construction	Transition
• Infrastructure planning • Primary scenario prototyping	• Infrastructure base lining • Architecture release testing Change management • Initial user manual	• Infrastructure upgrades • Release testing • Change management • User manual base line • Requirements verification	• Infrastructure maintenance • Release base lining • Change management • Deployment to users • Requirements verification

Process Automation

[Computer-aided software engineering]

- Computer-aided software engineering (CASE) is software to support software development and evolution processes.
- Activity automation
  - Graphical editors for system model development; Data dictionary to manage design entities;
  - Graphical UI builder for user interface construction;
  - Debuggers to support program fault finding;
  - Automated translators to generate new versions of a program.

Computer-aided software engineering (CASE) Technology

- Case technology has led to significant improvements in the software process. However, these are not the order of magnitude improvements that were once predicted
  - Software engineering requires creative thought - this is not readily automated;
  - Software engineering is a team activity and, for large projects, much time is spent in team interactions. CASE technology does not really support these.

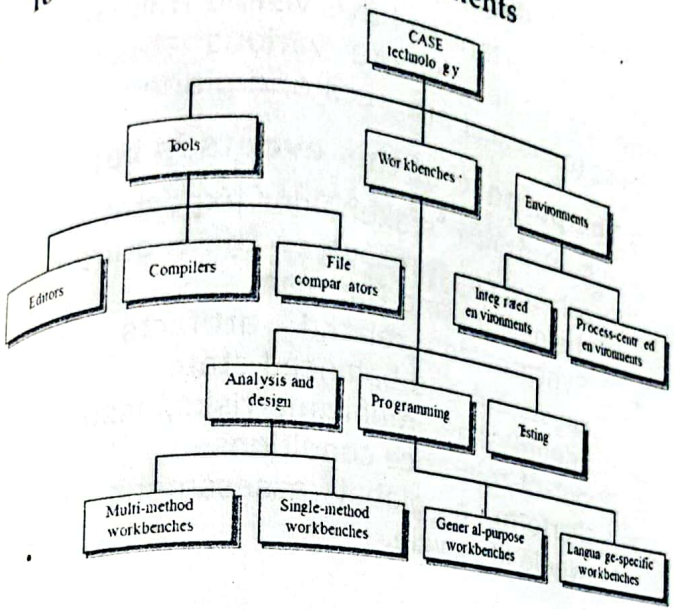
CASE Classification

- Classification helps us understand the different types of CASE tools and their support for process activities.
- Functional perspective
  - Tools are classified according to their specific function.
- Process perspective
  - Tools are classified according to process activities that are supported.
- Integration perspective
  - Tools are classified according to their organisation into integrated units.

CASE Integration

- Tools
  - Support individual process tasks such as design consistency checking, text editing, etc.
- Workbenches
  - Support a process phase such as specification or design, Normally include a number of integrated tools.
- Environments
  - Support all or a substantial part of an entire software process. Normally include several integrated workbenches.

Tools, Workbenches, Environments



Project Control and Process Instrumentation

The Core Metrics

METRIC	PURPOSE	PERSPECTIVES
Work and progress	Iteration planning, plan vs. actuals, management indicator	Source Lines Of Code, function points, object points, scenarios, test cases, Supply Chain Operating System
Budget cost and expenditures	Financial insight, plan vs. actuals, management indicator	Cost per month, full-time staff per month, percentage of budget expended
Staffing and team dynamics	Resource plan vs. actuals, hiring rate, attrition rate	People per month added, people per month leaving
Change traffic and stability	Iteration planning, management indicator of schedule convergence	Software changes
Breakage and modularity	Convergence, software scrap, quality indicator	Reworked SLOC per change, by type, by release/ component/subsystem
Rework and adoptability	Convergence, software rework, quality indicator	Average hours per change, by type, by release/ component/subsystem

Process Customization

- It is important to have visible milestones in the life cycle , where various stakeholders meet to discuss progress and planes.

- **The purpose of this events is to:**
  - Synchronize stakeholder expectations and achieve concurrence on the requirements, the design, and the plan.
  - Synchronize related artifacts into a consistent and balanced state
  - Identify the important risks, issues, and out-of-rolerance conditions
  - Perform a global assessment for the whole life-cycle.

Thank you

Software Project Management

Course Contents:

Unit 04: Modern Approach to Software Project and Economics

3.1 Elements of Modern Software Projects and Management Principles

3.2 Next-Generation Software Economics and Cost Models

3.3 Modern Process Transition – Paradigm Shifts

Modern Software Development Project

- In short, software development is the overall process involved when taking a software project from beginning to production delivery.
- This process incorporates the design, documentation, programming, testing and ongoing maintenance of a software deliverable.

Modern Software Project Management

- Modern S/W Project Management that makes this a perfect time to reflect on what modern Project Management has become and why it is such a critical issue for business organizations of all types.
- Project Management is a discipline for planning, leading, organizing and controlling a well-defined collection of work.

Modern Software Project Management

- Software is said to be an intangible product.
- Software development is a kind of all new stream in world business and there's very little experience in building software products.
- Most software products are adapt/modify made to fit client's requirements.
- The most important is that the underlying technology changes and advances so frequently and rapidly that experience of one product may not be applied to the other one.
- All such business and environmental constraints bring risk in software development hence it is essential to manage software projects efficiently.

- The image above shows triple constraints for software projects.
- It is an essential part of software organization to deliver quality product, keeping the cost within client's budget constrain and deliver the project as per scheduled.

- There are several factors, both internal and external, which may impact this triple constrain triangle.

- Therefore, software project management is essential to incorporate user requirements along with budget and time constraints.

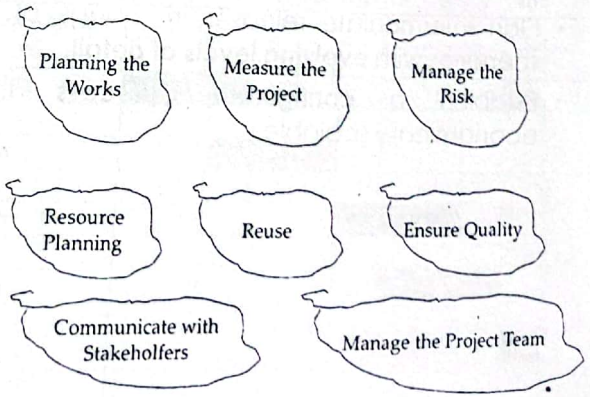
Five Essential Elements for Successful Software Development

- Integrated Development Environment (IDE).
- Source Control.
- Automated Testing.
- Automated Build.
- Defect Management.

The 5 Most Important Elements of Successful Project Management

- Have Clear Project Goals. Make sure you have all the details in front of you before you start.
- Be Dynamic. Once you have your plan in place, remain flexible.
- Communication. You need to ensure your team clearly communicates with one another.
- Stay on Track.
- Review The Project to Improve for The Next Time

Management Activities



Top 10 Management Principles of Iterative Development

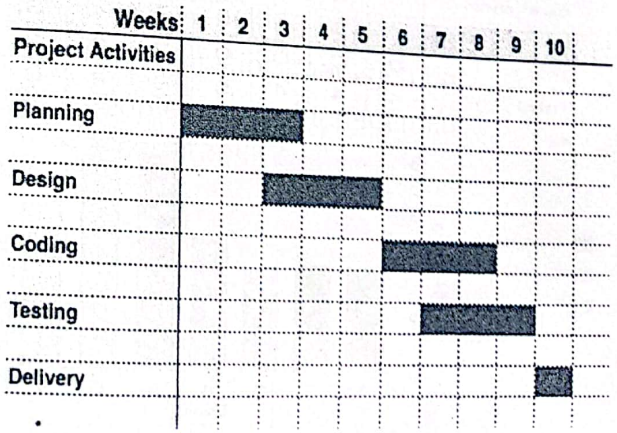
- Base the process on an **architecture-first** approach.
- Establish an **iterative lifecycle process** that confronts risk early.
- Transition design methods to emphasize **component-based development.**
- Establish a **change-management environment.**
- Enhance **change freedom** through tools.
- Capture design artifacts in rigorous, **model-based notation.**
- Instrument the process for **objective quality control and progress assessment.**
- Use a **demonstration-based approach** to assess intermediate artifacts.
- Plan intermediate releases in groups of usage scenarios with **evolving levels of detail.**
- Establish a **configurable process** that is economically scalable.

Project management principles are necessary assets when charting a path to completion.

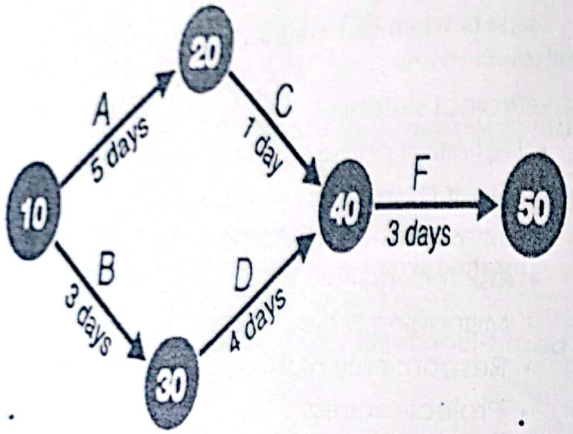
- Project structure
- Definition phase
- Clear goals
- Transparency about project status
- Risk recognition
- Managing project disturbances
- Responsibility of the Project Manager
- Project success

Project Management Tools

1. Gantt Chart

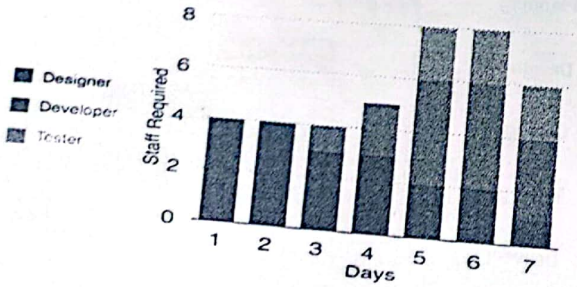


2. PERT Chart



3. Resource Histogram

Staff	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7
Designer	4	4	3	3	2	2	1
Developer	0	0	1	2	4	4	3
Tester	0	0	0	0	2	2	2
Total	4	4	4	5	8	8	6



Software Management Best Practices

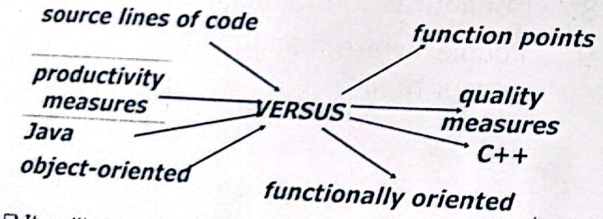
There is nine best practices:

1. Formal risk management
2. Agreement on interfaces
3. Formal inspections
4. Metric-based scheduling and management
5. Binary quality gates at the inch-pebble level
6. Program-wide visibility of progress versus plan
7. Defect tracking against quality targets
8. Configuration management
9. People-aware management accountability

Next-Generation Software Economics & Cost Models

Next-Generation Cost Models

- Software experts hold widely different opinions about software economics and its symptom in software cost estimation models:



- It will be difficult to improve based on estimation models while the project data going into these models are highly lacking a mutual relationship , and are based on differing process and technology foundations.

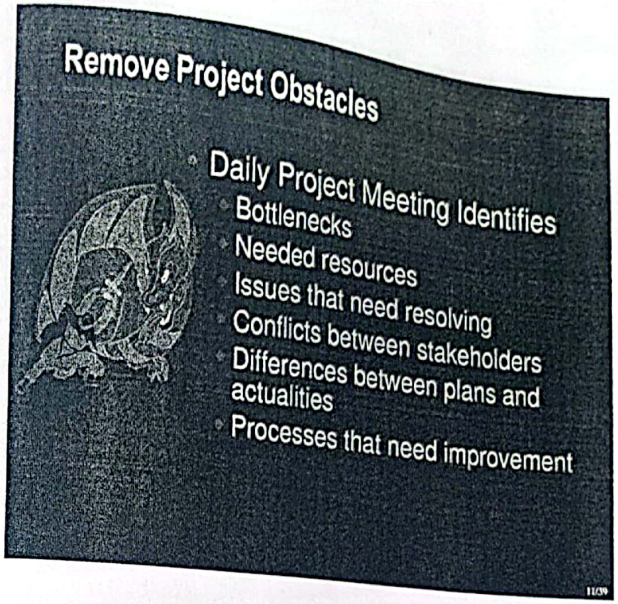
- Some of today's popular software cost models are not well matched to an computational software process focused an architecture-first approach
- Many cost estimators are still using a traditional or common process experience base to estimate a modern project profile
- Then, a next-generation software cost model should clearly separate architectural engineering from application production, just as an architecture-first process does.

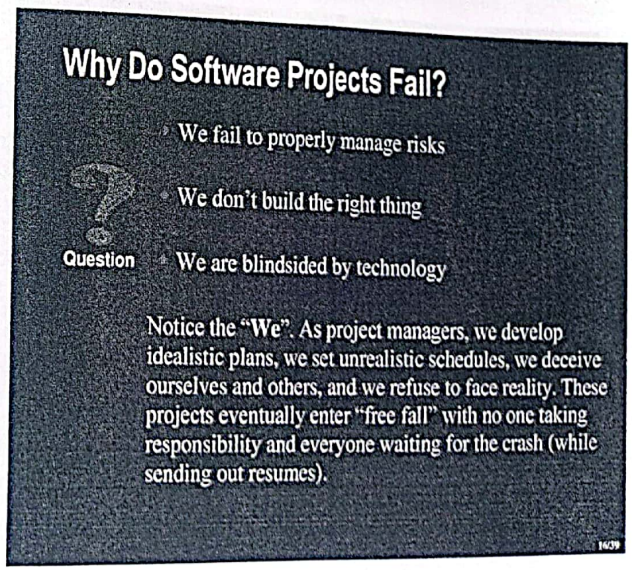
- There are two major improvements in next-generation software cost estimation models:-
  - Separation of the engineering stage from the production stage: which will force estimators to differentiate between architectural scale and implementation size.
  - Detailed design notations such as UML will offer an opportunity to define units of measure for scale that are more standardized and therefore can be automated and tracked.

Modern Software Economics

- Changes: that provide a good description of what an organizational manager should attampt for in making the transition to a modern process:

1. Finding and fixing: a software problem after delivery costs is 100 times more than fixing the problem in early design phases.
2. We can compress software development schedules 25% of nominal, but no more.
3. For every $100 you spend on development, you will spend $200 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 differences in software productivity.
6. Only about 15% of software development effort is devoted to programming.
7. Software systems and products cost 3 times as much per SLOC as individual software programs.
8. 80% of the contribution comes from 20% of the contributors.





Two Types of Risk

- Project and Process Risk
  - What could go wrong with this project?

Define

- Product or Requirements Risk
  - Which faults would be most damaging to the stakeholders?

31/39

What Do Project Managers Do?

Direct

- Team Management
- Plan, Schedule, Track
- Resource Allocation
- Project Direction
- Politics
- Remove Project Obstacles

38/39

Assignment 04:

- Discussed in following:
  1. Modern Process Transitions.
  2. Denouement

Modern Process Transitions

Culture Shifts

- Several culture shifts must be overcome to transition successfully to a modern software management process:
  - Lower level and mid-level managers are performers
  - Requirements and designs are fluid and tangible
  - Good and bad project performance is much more obvious earlier in the life cycle
  - Artifacts are less important early, more important later

- Real issues are surfaced and resolved systematically
- Quality assurance is everyone's job, not a separate discipline
- Performance issues arise early in the life cycle
- Investments in automation is necessary
- Good software organization should be more profitable

Denouement

- Good way to transition to a more mature iterative development process that supports automation technologies and modern architectures is to take the following shot:

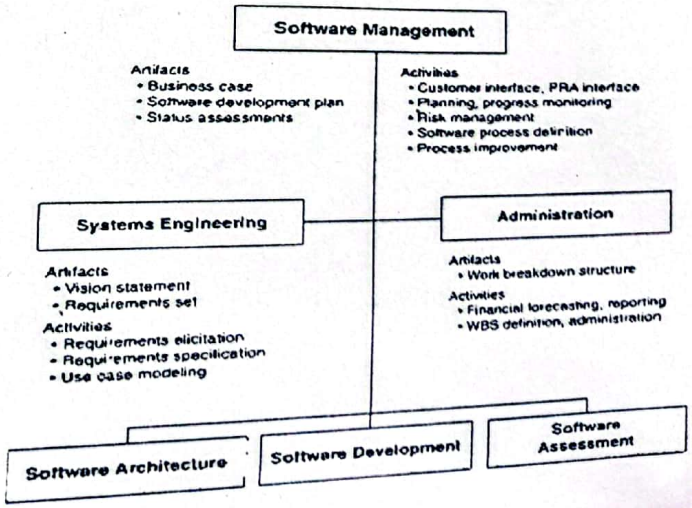
- Ready.

  Do your homework. Analyze modern approaches and technologies. Define your process. Support it with mature environments, tools, and components. Plan thoroughly.

- Aim.

  Select a critical project. Staff it with the right team of complementary resources and demand improved results.

- Fire.

  Execute the organizational and project-level plans with vigor and follow-through.

- This structure can be tailored to the size and circumstances of the specific project organization.