Chapter 1

SOFTWARE ENGINEERING, THE SOFTWARE PROCESS

1.1

SOFTWARE:

"SET OF instructions (computer programs) that when executed provide desired features, function, and performance"

Why we use software:

- Software delivers the most important product of our time—**information.**
- It transforms personal data so that the data can
be more useful in a local context;
- It **manages** business information to enhance competitiveness;
- It provides a **gateway to worldwide information networks** (e.g., the
Internet), and provides the means for acquiring information in all of its form.

- Why does it take so long to get software finished?
- Why are development costs so high?
- Why can't we find all errors before we give the software to our customers?
- Why do we spend so much time and effort maintaining existing programs?
- Why do we continue to have difficulty in measuring progress as software is being developed and maintained?

Nature of Software

Characteristics of software

- Software is developed or engineered; it is not manufactured
- Software doesn't "wear out."

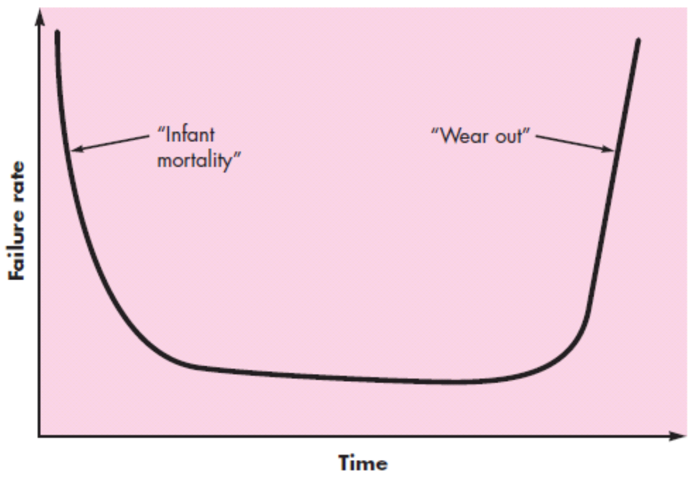


Fig- "bathtub curve," indicates that hardware exhibits relatively high failure rates early in its life hardware wear out

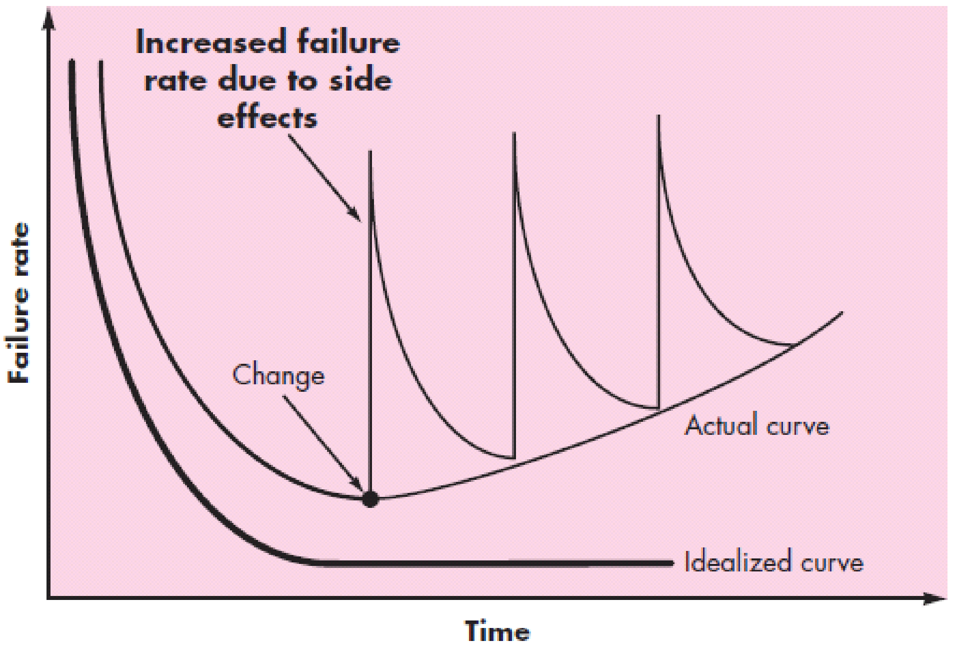


Figure shows During its life,2 software will undergo change. As changes are made, it is likely that errors will be introduced, causing the failure rate curve to spike as shown in the "actual curve" (Figure). Before the curve can return to the original steady-state failure rate, another change is requested, causing the curve to spike again. Slowly, the minimum failure rate level begins to rise—the software is deteriorating due to change

Software Application Domain

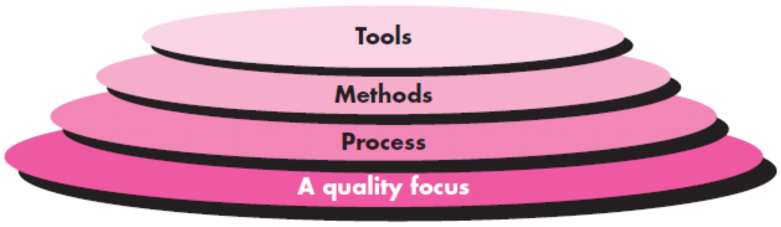
- **System software**—a collection of programs written to service other programs. Some system software (e.g., compilers, editors, and file management utilities)

- **Application software**—stand-alone programs that solve a specific business need e.g., point-of-sale transaction processing, real-time manufacturing process control
- **Engineering/scientific software**—has been characterized by "number crunching" algorithms. Applications range from astronomy to volcanology, from automotive stress analysis to space shuttle orbital dynamics, and from molecular biology to automated manufacturing.
- **Web applications**—called "WebApps"- this network-centric software category spans a wide array of applications.
- **Artificial intelligence software**—makes use of nonnumerical algorithms to solve complex problems that are not amenable to computation or straightforward analysis

Legacy Software
Hundreds of thousands of computer programs fall into one of the seven broad application domains discussed in the preceding subsection. Some of these are state of-the-art software—just released to individuals, industry, and government. But other programs are older, in some cases much older.

Software Engineering

Def: "The application of a systematic, disciplined, quantifiable approach to the development, operate, and maintenance of software; that is, the software engineering"



Software engineering is a layered technology. Referring to Figure any engineering approach (including software engineering) must rest on an organizational commitment to quality.

- Total **quality** management, Six Sigma, effective approaches to software engineering.
- The foundation for software engineering is the **process layer**. The software engineering process is the glue that holds the technology layers together and enables rational and timely development of computer software. Process defines a framework
- Software engineering **methods** provide the technical how-to's for building software. Methods encompass a broad array of tasks that

include communication, requirements analysis, design modeling, program construction, testing, and support.

- Software engineering **tools** provide automated or semiautomated support. When tools are integrated so that information created by one tool can be used by another

Software Process:

A **generic process framework for software engineering encompasses five activities**:

- **Communication.** Before any technical work can commence, it is critically important to communicate and collaborate with the customer (and otherstakeholders11 the intent is to understand stakeholders' objectives for the project and to gather requirements that help define software features and functions.
- **Planning.** Any complicated journey can be simplified if a map exists. A software project is a complicated journey, and the planning activity centesimal" that helps guide the team as it makes the journey. The map—called a software project plan—defines the software engineering work by describing the technical tasks to be conducted, the risks that are likely, the resources that will be required, the work products to be produced, and a work schedule.
- **Modeling**. Whether you're a landscaper, a bridge builder, an aeronautical engineer, a carpenter, or an architect, you work with models every day. You create a "sketch" of the thing so that you'll understand the big picture—what it will look like architecturally, how the constituent parts fit together, and many other characteristics. If required, you refine the sketch into greater and greater detail in an effort to better understand the problem and how you're going to solve it. A software engineer does the same thing by creating models to better understand software requirements and the design that will achieve those requirements.
- **Construction**. This activity combines code generation (either manual or automated) and the testing that is required to uncover errors in the code.
- **Deployment**. The software (as a complete entity or as a partially completed increment) is delivered to the customer who evaluates the delivered product and provides feedback based on the evaluation.

Software engineering process framework activities are complemented by a number of umbrella activities. In general, umbrella activities are applied throughout a software project and help a software team manage and control progress, quality, change, and risk.

Typical **umbrella activities** include:

- **Software project tracking and control**—allows the software team to assess progress against the project plan and take any necessary action to maintain the schedule.
- **Risk management**—assesses risks that may affect the outcome of the project or the quality of the product. Software quality assurance—defines and conducts the activities required to ensure software quality.
- **Technical reviews**—assesses software engineering work products in an effort to uncover and remove errors before they are propagated to the next activity.
- **Measurement**—defines and collects process, project, and product measures that assist the team in delivering software that meets stakeholders' needs; can be used in conjunction with all other framework and umbrella activities.
- **Software configuration management**—manages the effects of change throughout the software process.
- **Reusability management**—defines criteria for work product reuse (including software components) and establishes mechanisms to achieve reusable components.
- **Work product preparation and production**—encompasses the activities required to create work products such as models, documents, logs, forms, and lists.

Software Engineering Practice

Essence of Practice

Understand the problem. (communication and analysis).
- Identify **stakeholders**
- Solve **data, functions, and features are required**
- **Represent in smaller problems**
- Represent problem **graphically**

Plan the solution. (modeling and software design).
- Understand patterns
- Has a similar problem been solved so can be **reusable**
- Define the **subproblems**

Carry out the plan. (code generation).
- Design **Source code** according to model
- Prepare **solution in part wise** and correct

Examine the result

• Test each component part of the solution
• Solution produce results that conform to the data, functions, and features that are required

General Principals

The First Principle: The Reason It All Exists

A software system exists for one reason: **to provide value to its users**.

The Second Principle:(Keep It Simple, Stupid!)

Software design is not a difficult process. There are many factors to considering any design effort. All design should be as simple as possible.

The Third Principle: Maintain the Vision

A clear vision is essential to the success of a software project.

The Fourth Principle: What You Produce, Others Will Consume

Seldom implement knowing someone else will have to understand what you are doing.

The Fifth Principle: Be Open to the Future

A system with a long lifetime has more value. In today’s computing environments, to reuse of an entire system.

The Sixth Principle: Plan Ahead for Reuse

Reuse saves time and effort. The reuse of code and designs have been proclaimed as a major benefit of using technologies. Planning ahead for reuse reduces the cost and increases the value of both the reusable components and the systems into which they are incorporated.

The Seventh principle: Think!

Placing clear, complete thought before action almost always produces better results.

Software Crisis

The word crisis has definition: "the turning point in the course of a disease, when it becomes clear whether the patient will live or die."

This give us a clue about the real nature of the problems that have plagued software development.

Somewhere crisis is also deal with the failures.

Regardless the set of problems that are encountered in the development of computer software is not limited to software that "doesn't function properly.

Rather, it encompasses problems associated with how we develop software

How we support a growing volume of existing software, how we can expect to keep pace with a growing demand for more software.

And yet, things would be much better if we could find and broadly apply a cure

Software Myths

Management myths.

Managers with software responsibility, most disciplines, are often under pressure to maintain budgets, keep schedules from slipping, and improve quality.

- **Myth:** We already have a book that's full of standards and procedures for building software. Won't that provide my people with everything they need to know?

  Reality: The book of standards may very well exist, but is it used? Are software practitioners aware of its existence? Does it reflect modern software engineering practice? Is it complete? Is it adaptable? Is it streamlined to improve time-to-delivery while still maintaining a focus on quality? In many cases, the answer to all of these questions is "no."

- **Myth:** If we get behind schedule, we can add more programmers and catch up.

  Reality: Software development is not a mechanistic process like manufacturing. "adding people to a late software project makes it later." However, as new people are added, people who we're working must spend time educating the newcomers

- **Myth:** If I decide to outsource the software project to a third party, I can just relax and let that firm build it.

  Reality: If an organization does not understand how to manage and control software projects internally, it will invariably struggle when it outsources software projects.

Customer myths. A customer who requests computer software

- **Myth:** A general statement of objectives is sufficient to begin writing programs—we can fill in the details later.

  Reality: Although a comprehensive and stable statement of requirements is not always possible. Unambiguous requirements are developed only through effective and continuous communication between customer and developer.

- **Myth:** Software requirements continually change, but change can be easily accommodated because software is flexible.

  Reality: It is true that software requirements change, but the impact of change varies with the time changes are requested early the cost impact is relatively small than the cost impact of change at commitment level

Practitioner's myths. Myths that are still believed by software practitioners

- **Myth:** Once we write the program and get it to work, our job is done.

  Reality: Someone once said that "the sooner you begin 'writing code,' the longer it'll take you to get done." Industry data indicate 60 and 80% of all effort expended on software will be expended after it is delivered to the customer for the first time.

- **Myth:** Until I get the program "running" I have no way of assessing its quality.

  Reality: the technical review. Software reviews are a "quality filter" that have been found to be more

- **Myth:** The only deliverable work product for a successful project is the working program.

  Reality: A working program is only one part of a software configuration includes many elements. A variety of work products provide a foundation for successful engineering

- **Myth:** Software engineering will make us create voluminous and unnecessary documentation and will invariably slow us down.

  Reality: Software engineering is not about creating documents. It is about creating a quality product. Better quality leads to reduced rework.

1.2

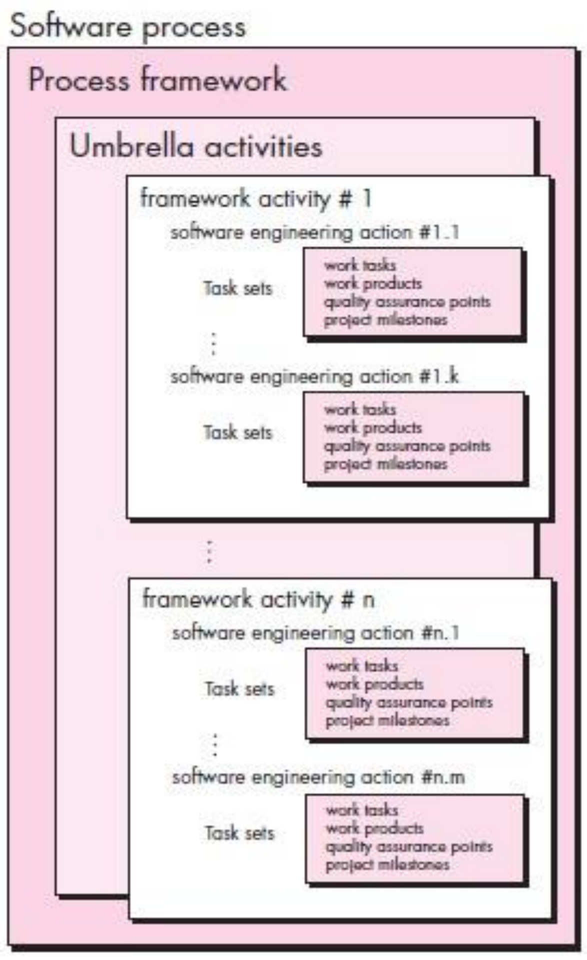
PROCESS MODELS

Generic Process Model

The software process is represented schematically each framework activity is populated by a set of software engineering actions. Each software engineering action is defined by a task set that identifies the work

Defining a Framework Activity

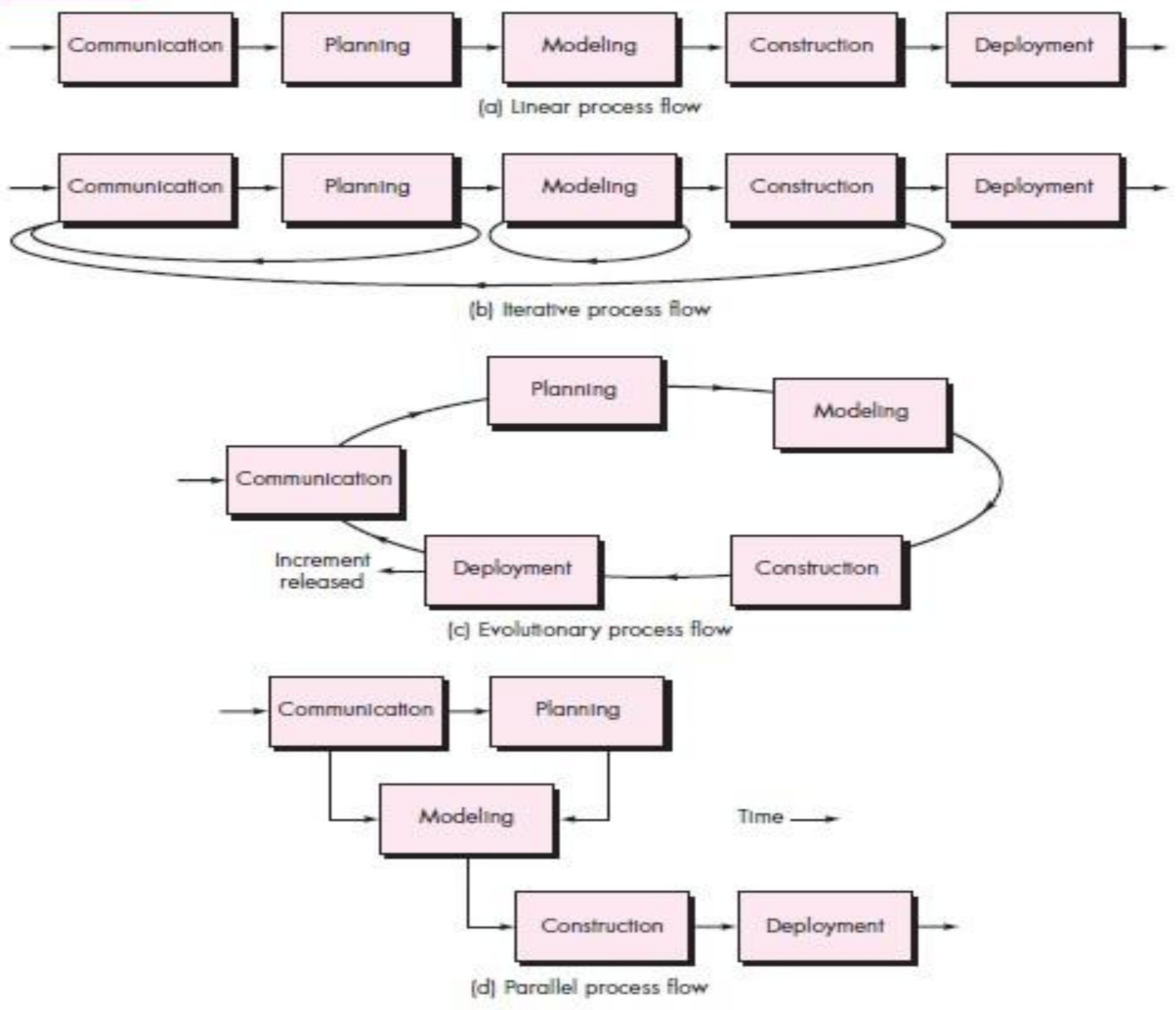
a generic process framework for software engineering defines five framework activities—**communication, planning, modeling, construction,** and **deployment.** In addition, a set of umbrella activities—project tracking and control, risk management, quality assurance, configuration management, technical reviews, and others—are applied throughout the process.



Defining a Framework Activity

What actions are appropriate for a framework activity, given the nature of the problem to be solved, the characteristics of the people doing the work, and the stakeholders who are sponsoring the project

FIGURE 2.2 Process flow



Identifying a Task Set

Select a task set that best accommodates the needs of the project and the characteristics of team. This implies that a software engineering action can be adapted to the specific needs of the software project

Process Patterns

1. Stage pattern—defines a problem associated with a framework activity for the process. Ex: **Establishing Communication.**

2. Task pattern—defines a problem associated with a software engineering action or work task and relevant to successful software engineering practice (e.g., **Requirements Gathering** is a task pattern).

3. Phase pattern—define the sequence of framework activities that occurs within the process, even when the overall flow of activities is **iterative** in nature. Ex: **Spiral Model**

Process Assessment and Improvement

A number of different approaches to software process assessment and improvement have been proposed

- **Standard CMMI Assessment Method for Process Improvement (SCAMPI)**— provides a five-step process assessment model that incorporates five phases: **initiating, diagnosing, establishing, acting, and learning.**
- **CMM-Based Appraisal for Internal Process Improvement (CBA IPI)**— provides a diagnostic technique for assessing the relative maturity of a software organization.
- **SPICE (ISO/IEC15504)** — The intent of the standard is to assist organizations in developing an objective evaluation of the efficacy of any defined software process [ISO08].
- **ISO 9001:2000 for Software**—a generic standard that applies to any organization that wants to improve the overall quality of the products, systems, or services that it provides. Therefore, the standard is directly applicable to software organizations and companies

The Waterfall Model

There are times when the requirements for a problem are well understood—when work flows from **communication** through **deployment** in a reasonably linear fashion.

The waterfall model, sometimes called the classic life cycle, suggests a systematic, sequential approach to software development that begins with customer specification of requirements and progresses through planning, modeling, construction, and deployment, culminating in ongoing support of the completed software.

A variation in the representation of the waterfall model is called the V-model. Represented in Figure 2.4, the V-model [Buc99] depicts the relationship of quality's

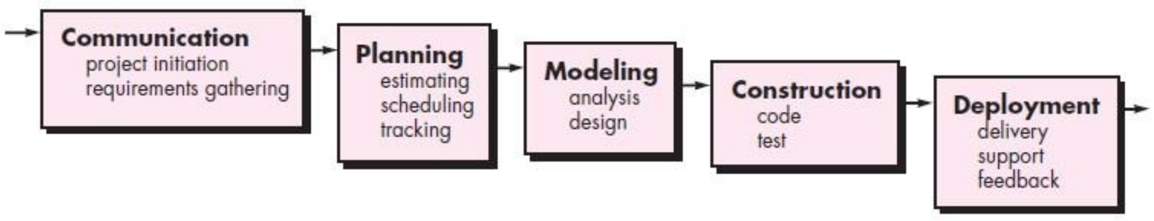


Fig: Waterfall model

Incremental Process Models

The incremental model combines elements of linear and parallel process flows the incremental model applies linear sequences in a staggered fashion as calendar time progresses. Each linear sequence produces deliverable "increments" of the software.

Example: word-processing software developed using the incremental paradigm

- **1st** Increment: Deliver basic file management, editing, and document production functions
- 2nd Increment: more sophisticated editing and document production capabilities
- 3rd Increment: spelling and grammar checking;
- 4th Increment: Advanced page layout capability.

When an incremental model is used, the first increment is often a core product. The core product is used by the customer plan is developed for the next increment. The plan addresses the modification of the core product to better meet the needs of the customer
This process is repeated following the delivery of each increment, until the complete product is produced.
The incremental process model focuses on the delivery of an operational product with each increment. Early increments are stripped-down versions of the final product,

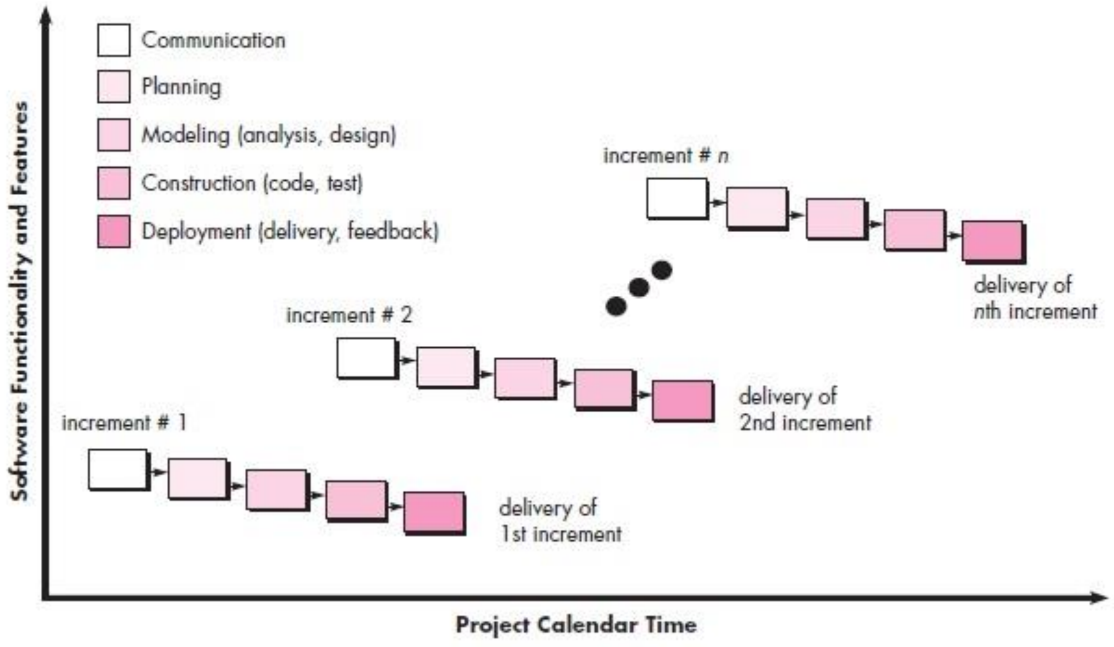
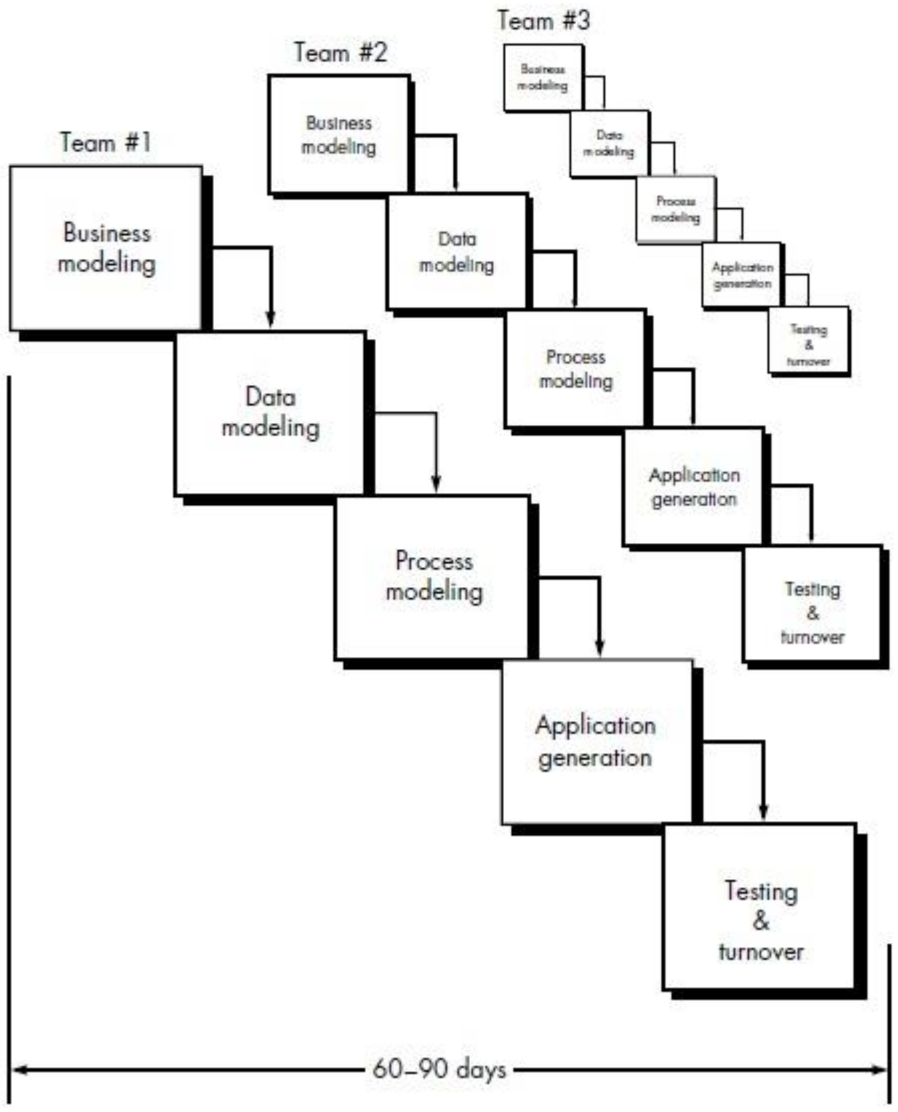

Communication
Planning
Modeling (analysis, design)
Construction (code, test)
Deployment (delivery, feedback)
increment # n
delivery of nth increment
increment # 2
delivery of 2nd increment
increment # 1
delivery of 1st increment
Software Functionality and Features
Project Calendar Time

Fig: Incremental Process Model

RAD Model (Rapid Application Development)



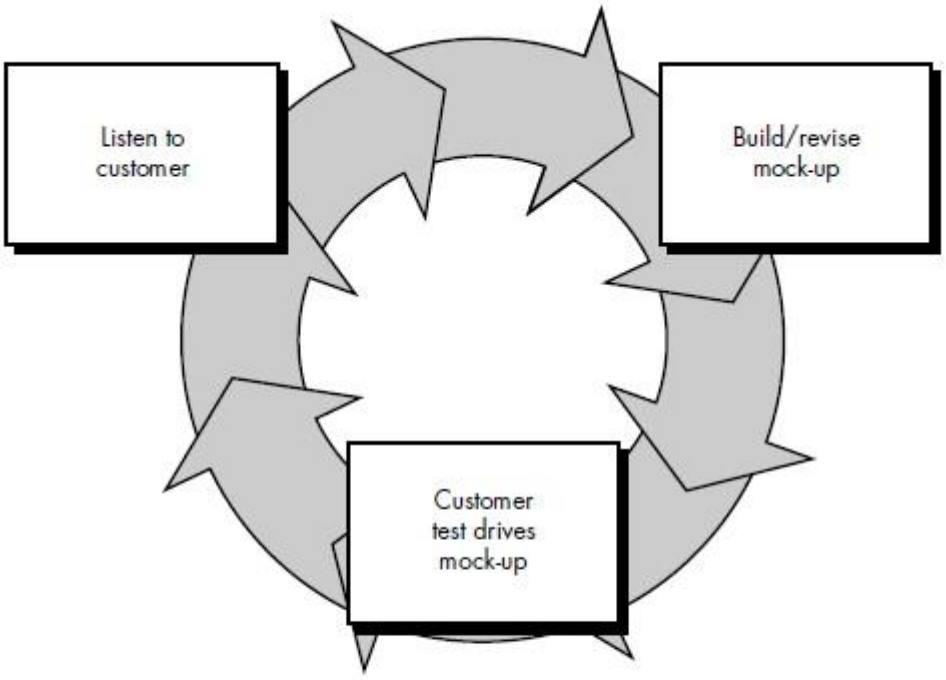
- Rapid application development (RAD) is an incremental software development process model that emphasizes an extremely short development cycle.
- The RAD model is a "high-speed" adaptation of the linear sequential model in which rapid development is achieved by using component-based construction.
- RAD process enables a development team to create a "fully functional system" within very short time periods (e.g., 60 to 90 days)

Business modeling. The information flow among business functions is modeled in a way that answers the following questions: What information drives the business process? What information is generated? Who generates it? Where does the information go? Who processes it?
Data modeling. Set of data objects that are needed to support the business. The attributes of each object are identified and the relationships between these objects defined.

Process modeling. The Processing descriptions are created for adding, modifying, deleting, or retrieving a data object.
Application generation. AD assumes the use of fourth generation techniques RAD process works to reuse existing program components. Automated tools are used to facilitate construction of the software.
Testing and turnover. Since the RAD process emphasizes reuse, many of the program components have already been tested. This reduces overall testing time.

Prototype model

- The prototyping paradigm assists you and other stakeholders/customer to better understand what is to be built when requirements are fuzzy
- The quick design leads to the construction of a prototype.
- The prototype is deployed and evaluated by stakeholders, who provide feedback that is used to further refine requirements.
- Iteration occurs as the prototype is tuned to satisfy the
  needs of various stakeholders, while at the same time enabling you to better understand what needs to be done
- Both stakeholders and software engineers like the prototyping paradigm. Users get a feel for the actual system, and developers get to build something immediately.

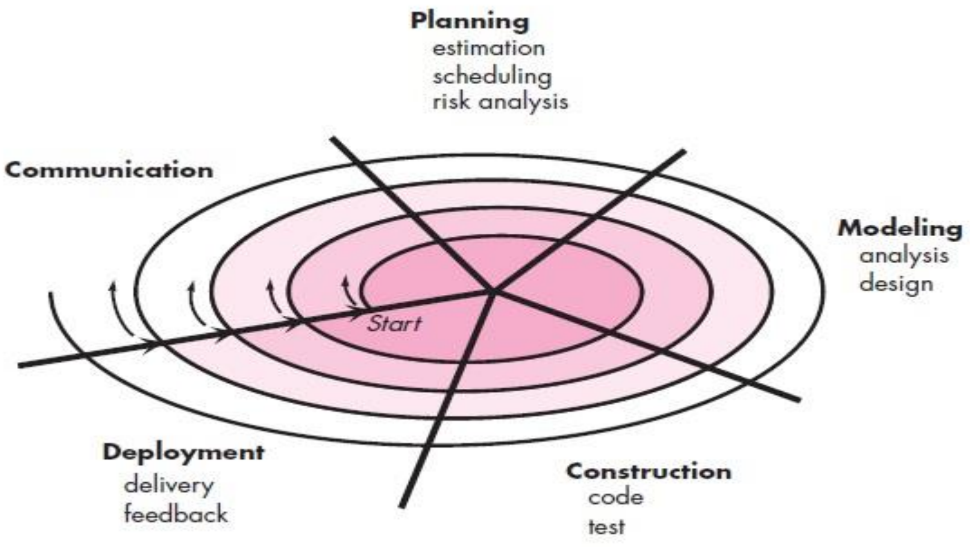


- The quick design focuses on a representation of those aspects of the software that will be visible to the customer/user

- The prototype is evaluated by the customer/user and used to refine requirements for the software to be developed.
- Iteration occurs as the prototype is tuned to satisfy the needs of the customer, while at the same time enabling the developer to better understand what needs to be done
- Ideally, the prototype serves as a mechanism for identifying software requirements. If a working prototype is built

Spiral Model

- The spiral model, originally proposed by Boehm is an evolutionary software.
- It provides the potential for rapid development of incremental versions of the software.
- Using the spiral model, software is developed in a series of incremental releases.
- During early iterations, the incremental release might be a paper model or prototype.
- During later iterations, increasingly more complete versions of the engineered system are produced.
- A spiral model is divided into a number of framework activities, also called task
- The software team performs activities that are implied by a circuit around the spiral in a clockwise direction, beginning at the center
- first circuit around the spiral might represent a "concept development project" that starts at the core of the spiral and continues for multiple iterations



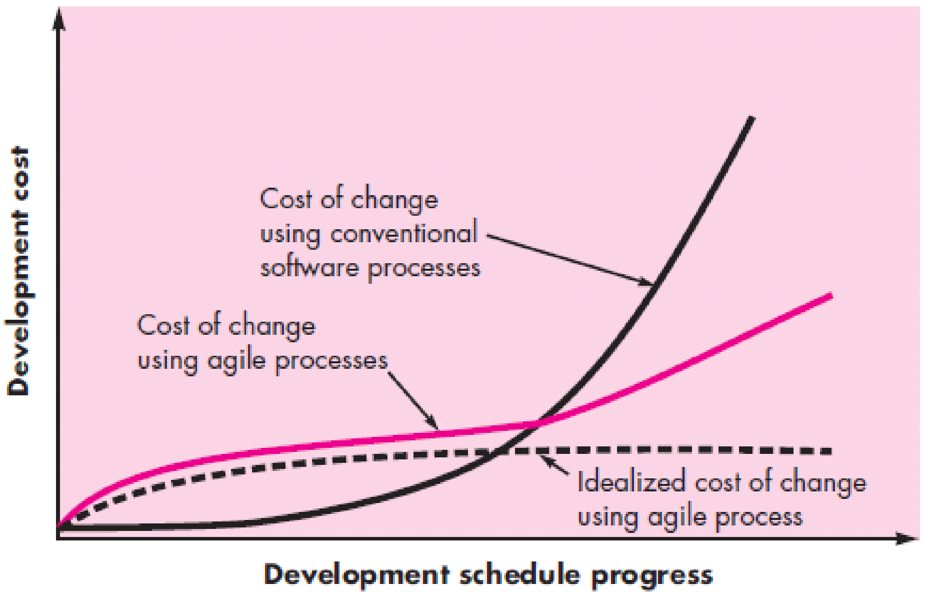
1.3

Agile Development

What is Agility...?

- But agility is effective response to change.
- It encourages team structures and attitudes that make communication among team members, between technologists and business people, between software engineers and their managers more facile.
- It emphasizes rapid delivery of operational software
- It adopts the customer as a part of the development team
- Agility can be applied to any software process emphasize an incremental delivery strategy that gets working software to the customer as rapidly as feasible for the product type

Agility and cost of change



- It is relatively easy to accommodate a change when a software team is gathering requirements (early in a project).
- Costs escalate quickly, and the time and cost required to ensure that the change is made.
- A well-designed agile process “flattens” the cost of change curve
- a software team to accommodate changes late in a software project without cost and time impact.

- agile process encompasses incremental delivery. When incremental delivery is coupled with other agile practices(module) the cost of making a change is attenuated.
- significant reduction in the cost of change can be achieved

What is an Agile Process...?

- It is difficult to predict in advance which software requirements will persist and which will change.
- It is equally difficult to predict how customer priorities will change as the project proceeds
- It is difficult to predict how much design is necessary before construction is used to prove the design.
- Analysis, design, construction, and testing are not as predictable (from a planning point of view) as we might like.
- How do we create a process that can manage unpredictability?
  in process adaptability (to rapidly changing project and technical conditions). An agile process, therefore, must be adaptable.
- An agile software process must adapt incrementally. To accomplish incremental adaptation,
- An agile team requires customer feedback (so that the appropriate adaptations can be made). Hence, an incremental development strategy should be effective.
- This iterative approach enables the customer to evaluate the software increment regularly, provide necessary feedback to the software team,

Agility Principles

The agility principles for those who want to achieve agility:

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Business people and developers must work together daily throughout the project.
- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Working software is the primary measure of progress.

- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Continuous attention to technical excellence and good design enhances agility.
- Simplicity—the art of maximizing the amount of work not done is essential.
- The best architectures, requirements, and designs emerge from self–organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Human Factors

Competence. In an agile development (as well as software engineering) context, “competence” encompasses innate talent, specific software-related skills, and overall knowledge of the process that the team has chosen to apply. Skill and knowledge of process can and should be taught to all people who serve as agile team members?

Common focus. Although members of the agile team may perform different tasks and bring different skills to the project, all should be focused on one goal—to deliver a working software increment to the customer within the time promised. To achieve this goal, the team will also focus on continual adaptations (small and large) that will make the process fit the needs of the team.

Collaboration. Software engineering (regardless of process) is about assessing, analyzing, and using information that is communicated to the software team; creating information that will help all stakeholders understand the work of the team; and building information that provides business value for the customer. To accomplish these tasks, team members must collaborate—with one another and all other stakeholders.

Decision-making ability. Any good software team (including agile teams) must be allowed the freedom to control its own destiny. This implies that the team is given autonomy—decision-making authority for both technical and project issues.

Fuzzy problem-solving ability. Software managers must recognize that the agile team will continually have to deal with **ambiguity** and will continually be buffeted by change. In some cases, the team must accept the fact that the problem they are solving today may not be the problem that needs to be solved tomorrow. However, lessons learned from any problem-solving activity (including those that solve the wrong problem) may be of benefit to the team later in the project.

Mutual trust and respect. The agile team must become what a “**jelled” team**. A jelled team exhibits the trust and respect that are necessary to make them “so strongly knit that the whole is greater than the sum of the parts.”

Self-organization. In the context of agile development, self-organization implies three things:

(1) the agile team organizes itself for the work to be done,

(2) the team organizes the process to best accommodate its local environment,

(3) the team organizes the work schedule to best achieve delivery of the software increment. Self-organization has a number of technical benefits, but more importantly, it serves to improve collaboration and boost team morale.

1.4

Project Management Concepts

PEOPLE

Managers argue (as the preceding group had) that people are primary, but their actions sometimes belie their words. In this section we examine the players who participate in the software process and the manner in which they are organized to perform effective software engineering

The Players

1. **Senior managers** who define the business issues that often have significant influence on the project.

2. **Project (technical) managers** who must plan, motivate, organize, and control the practitioners who do software work.

3. **Practitioners** who deliver the technical skills that are necessary to engineer a product or application.

4. **Customers** who specify the requirements for the software to be engineered and other stakeholders who have a peripheral interest in the outcome.

5. **End-users** interact with the software once it is released for production use.

Team Leaders

Leadership includes:

- **Motivation.** The ability to encourage (by “push or pull”) technical people to produce to their best ability.

- **Organization.** The ability to mold existing processes (or invent new ones) that will enable the initial concept to be translated into a final product.

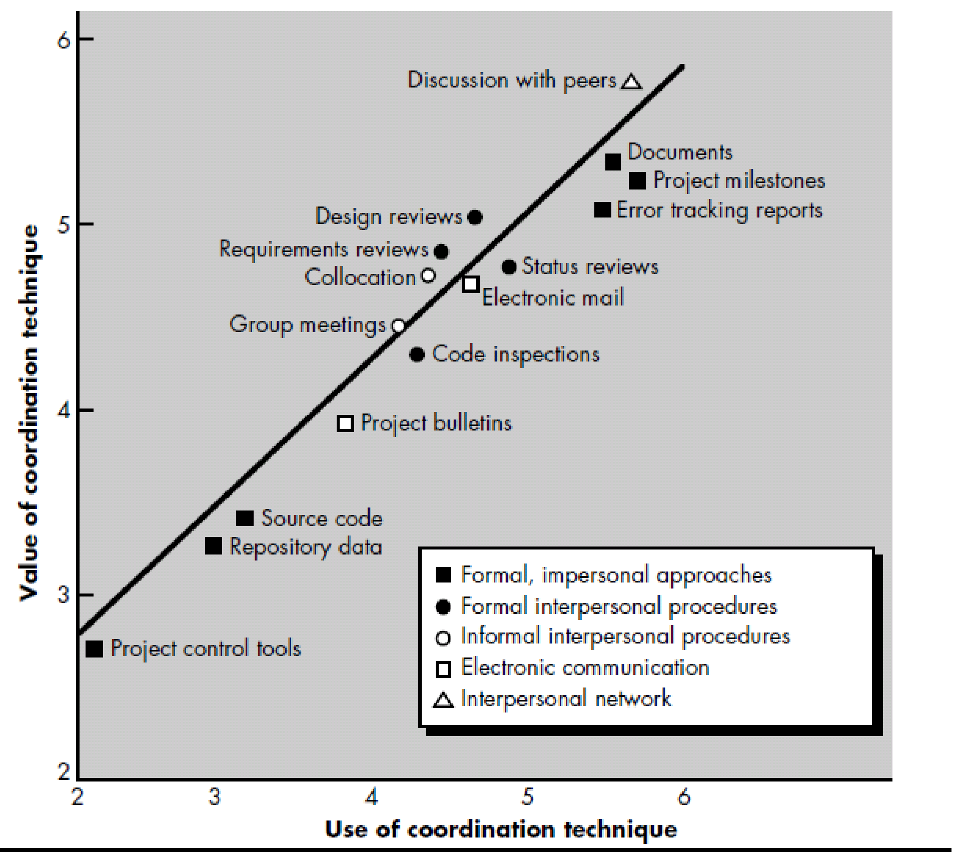
- **Ideas or innovation.** The ability to encourage people to create and feel creative even when they must work within bounds established for a particular software product or application.

- **Problem solving.** An effective software project manager can diagnose the technical and organizational issues that are most relevant, systematically

To achieve a high-performance team:
• Team members must have trust in one another.
• The distribution of skills must be appropriate to the problem.

Coordination and Communication Issues

- Formal, impersonal approaches
- Formal, interpersonal procedures
- Informal, interpersonal procedures
- Electronic communication
- Interpersonal networking



THE PRODUCT

Product is the problem it is intended to solve at the very beginning of the project. At a minimum, the scope of the product must be established and bounded

Software Scope

- **Context.** How does the software to be built fit into a larger system?
- **Information objectives.** What customer-visible data objects are produced as output from the software What data objects are required for input
- **Function and performance.** What function does the software perform to
  transform after giving input data into output.
  Software project scope must be unambiguous and understandable at the management and technical levels. A statement of software scope must be

Problem Decomposition
Problem decomposition, sometimes called partitioning or problem elaboration, is an activity that sits at the core of software requirements analysis

Rather, decomposition is applied in two major areas:
(1) the functionality that must be delivered
(2) the process that will be used to deliver it.

Human beings tend to apply a divide and conquer strategy when they are confronted with a complex problem. Stated simply, a complex problem is partitioned into smaller problems that are more manageable.

PROCESS:

The generic phases that characterize the software process—definition, development, and support—are applicable to all software

• the linear sequential model

• the prototyping models

• the RAD model

• the incremental model

• the spiral model

• the WINWIN spiral model

• the component-based development model

• the concurrent development models

• the formal methods model

• the fourth-generation techniques model

Melding the Product and the Process

• Customer communication—tasks required to establish effective requirements
elicitation between developer and customer.

• Planning—tasks required to define resources, timelines, and other project related information.

• Risk analysis—tasks required to assess both technical and management risks.

• Engineering—tasks required to build one or more representations of the application.

• Construction and release—tasks required to construct, test, install, and provide user support (e.g., documentation and training).

• Customer evaluation—tasks required to obtain customer feedback based on evaluation of the software representations created during the engineering activity and implemented during the construction activity.

Process Decomposition

Once the process model has been chosen, the common process framework (CPF) is adapted to it. In every case, communication, planning, risk analysis, engineering, construction and release, customer evaluation—can be fitted to the paradigm.

It will work for linear models, for iterative and incremental models, for evolutionary models, and even for concurrent or component assembly models.

Example:

Following work tasks requires for the customer communication activity
1. Develop list of clarification issues.
2. Meet with customer to address clarification issues.
3. Jointly develop a statement of scope.
4. Review the statement of scope with all concerned.
5. Modify the statement of scope as required.

THE PROJECT

In order to manage a successful software project, we must understand what can go wrong (so that problems can be avoided) and how to do it right.

1. Start on the right foot. This is accomplished by working hard (very hard)

- to **understand the problem** that is to be solved
- set **realistic objects** and expectations for everyone who will be involved in the project.
- building the **right team**
- giving the **team the autonomy, authority**, and technology needed to do the job.

2. Maintain momentum. Many projects get off to a **good start** and then slowly disintegrate. To maintain momentum,

- the project manager must provide **incentives** to keep turnover of personnel to an absolute minimum,
- the team should emphasize quality in every task it performs, and senior management should do everything possible to stay out of

3. Track progress. For a software project, progress is tracked as work products
are produced and approved (using formal technical reviews) as part of a quality assurance activity.

4. Make smart decisions. In essence, the decisions of the project manager and the software team should be to **"keep it simple."** Whenever possible, decide to use commercial off-the-shelf software or existing software components,

5. Conduct a postmortem analysis. Establish a consistent mechanism for extracting lessons learned for each project. Evaluate the planned and actual schedules, collect and analyze software project metrics, get feedback from team members and customers, and record findings in written form.

<u>THE W5HH PRINCIPLE</u>

Boehm's W5HH principle is applicable regardless of the size or complexity of a software project

Why is the system being developed?
The answer to this question enables all parties to assess the validity of business reasons for the software work. Stated in another way, does the business purpose justify the expenditure of people, time, and money?

What will be done, by when?
The answers to these questions help the team to establish a project schedule by **identifying key project tasks** and the **milestone**s that are required by the customer.

Who is responsible for a function?
Earlier in this chapter, we noted that the role and responsibility of each member of the **software team** must be defined. The answer to this question helps accomplish this.

Where are they organizationally located?
Not all roles and **responsibilities reside within the software team** itself. The customer, users, and other stakeholders also have responsibilities.

How will the job be done technically and managerially?
Once product scope is established, a management and technical strategy for the project must be defined.

How much of each resource is needed?
The answer to this question is derived by developing estimates based on answers to earlier questions.

<u>CRITICAL PRACTICES</u>

Formal risk management. What are the top ten risks for this project? For each of the risks, what is the chance that the risk will become a problem and what is the impact if it does?

Empirical cost and schedule estimation. What is the current **estimated** size?
of the application software (excluding system software) that will be delivered into operation? How was it derived?

Metric-based project management. Do you have in place a metrics program
to give an early indication of evolving problems? If so, what is the **current requirements** volatility?

Earned value tracking. report **monthly earned value** metrics If so,
are these metrics computed from an activity network of tasks for the entire?
effort to the **next deliver**y

Defect tracking against quality targets. Do you track and periodically report?
the number of defects found by each inspection (formal technical review) and execution test from program inception and the number of defects currently closed and open?

People-aware program management. What is the **average staff turnover?**
for the past three months for each of the suppliers/developers involved in the development of software for this system.