Unit-1 Introduction

Reference Books

An Integrated approach to software Engineering(3rd)
- Pankaj Jalote

Software Engineering-A Practitioner's Approach
– Roger S. Pressman

The Problem Domain

- Software engineering doesn't deal with programs that people build to illustrate something of for hobby.
- The problem domain is the software that solves some problem of some users.
- Larger systems or businesses may depend on the software.
- The problems in the software can lead to direct or indirect loss.

Student system & Industrial strength software

- Student system is primarily meant for demonstration purpose.
  - Bugs is not major problem
  - Testing and documentation are not done properly.
- Industrial strength software system is built to solve some problems of a client and is used by the client organizations.

Industrial Strength Software Requirements

- Software should be thoroughly tested before use.
- The development should be broken into phases such that output of each phase is evaluated and reviewed , so bugs can be removed.
- Besides quality requirements, there are requirements of backup and recovery, fault tolerance, following of standards, portability, etc.

What is software ?

IEEE defines software as the collection of computer programs, procedures, rules, and associated documentation and data.

Software Characteristics

- Software is developed or engineered, it is not manufactured in the classical sense.
- Software doesn't "wear out".
- Although the industry is moving toward component-based assembly, most software continues to be custom built.

Software Applications

- Software may be applied in any situation for which a pre-specified set of procedural steps has been defined.
- Following are the areas which uses software applications:
  - **System software**
  - **Real-time software**
  - **Business software**
  - **Engineering and scientific software**
  - **Embedded software**
  - **Personal computer software**
  - **Web-based software**
  - **Artificial intelligence software**

Software is Expensive

- Size of software is measured by Lines of Code (LOC) or thousands of lines of code (KLOC).
- The cost of developing software is generally measured in terms of person-months of effort spent in development.
- Productivity is measured in the industry in terms of LOC per person-month. (Generally ranges from 300 to 1000 LOC per person-month).

Software is Expensive

- Software companies charge the client for whom they are developing the software up to $100000 per person-year or more than $8000 per person-month.
- With the current productivity figures of the industry, this translates into a cost per line of code of approximately $8 to $25.

Classic hardware-software cost reversal chart

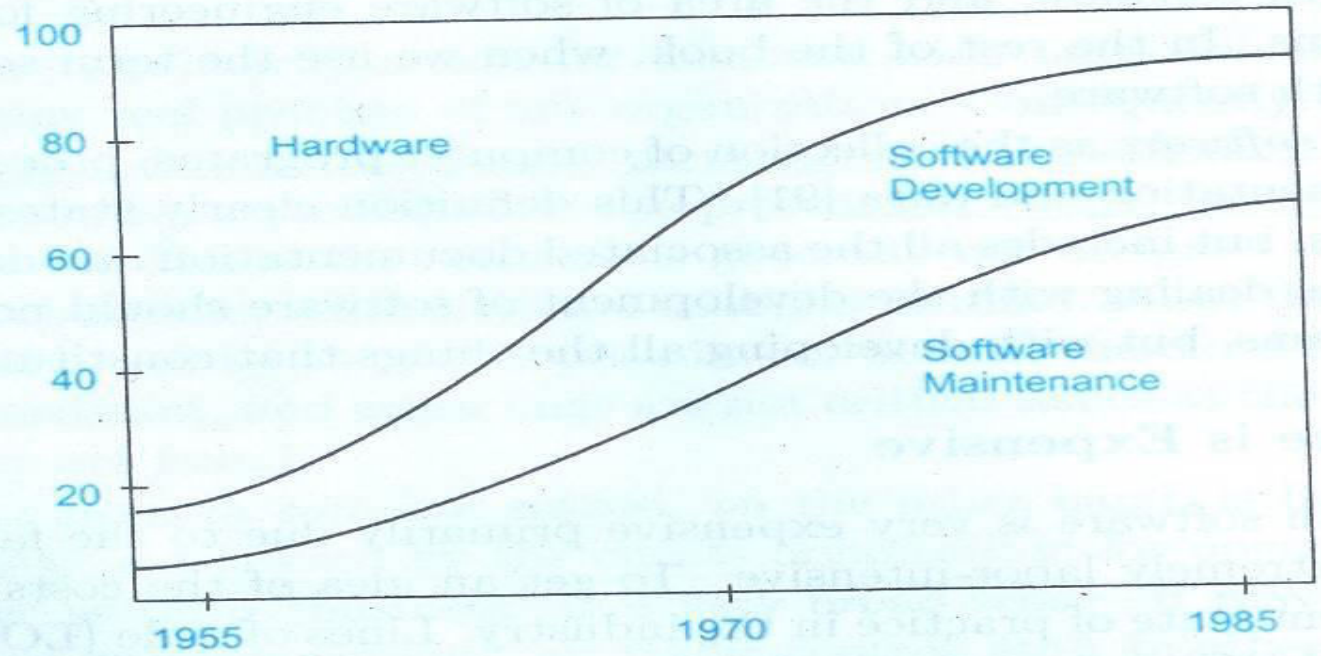

Figure 1.1: Hardware-software cost trend.

Late and Unreliable

- In a survey of over 600 firms, more than 35% reported some projects as a *runaway*.
- *A runaway* is not a project that is somewhat late or somewhat over budget – it is a project where the budget and schedule are out of control.
- Unreliability of software means that the software does not do what it is supposed to do or does something it is not supposed to do.
- The reason for software's unreliability are bugs or errors that get introduced during the design and development process. Hence, even though a software system may fail after operating correctly for some time.

Maintenance and Rework

- Once software is delivered and deployed, it enters the maintenance phase
- Software needs to be maintained not because some of its components wear out and need to be replaced, but because there are often some residual errors remaining in the system that must be removed as they are discovered.
- Many of these surfaces only after the system has been in operation, sometimes for a long time.

Maintenance and Rework

- These errors, once discovered, need to be removed, leading to the software being changed. This is some times called ***Corrective maintenance***.
- Even without bugs, software frequently undergoes change.
- The main reason is that software often must be upgraded and enhanced to include more features and provide more services.

Maintenance and Rework

Law of software evolution

- It has been argued that once a software system is deployed, the environment in which it operates changes.
- Hence, the needs that initiated the software development also change to reflect the needs of the new environment.
- Hence, the software must adapt to the needs of the changed environment.
- The changed software then changes the environment, which in turn requires further change.
- This phenomenon is called the ***law of software evolution***.
- Maintenance due to this phenomenon is sometimes called ***adaptive maintenance***.

Maintenance and Rework

- Maintenance revolves around understanding existing software and maintainers spend most of their time trying to understand the software they have to modify.
- It includes understanding of code and related documents.
- There should not be side effects of maintenance.

Maintenance and Rework

- To test whether those aspects of the system that are not supposed to be modified are operating as they were before modification, regression testing is done.
- Regression testing involves executing old test cases to test that no new errors have been introduced.
- The development proceeds when it is believed that the requirements are well understood.

Maintenance and Rework

- As time goes by and understanding of the system improves, the clients frequently discover additional requirements they had not specified earlier.
- This leads to requirements getting changed.
- This change leads to rework.
- The Requirements, the design, the code all have to be changed.
- It is estimated that rework costs are 30 to 40% of the development cost.

What is Software Engineering ?

- Software Engineering is defined as the systematic approach to the development, operation, maintenance, and retirement of software.
- Systematic approach means that methodologies are used for developing software which are repeatable. If they are applied by different groups of people, similar software will be produced.

The Software Engineering Challenges

- Scale
- Quality and Productivity
- Consistency and Repeatability
- Change

Scale

- Development of a very large system requires a very different set of methods compared to developing a small system.
- In other words, the methods that are used for developing small systems generally do not scale up to large system.
- For example, problem of counting people in a room versus counting people of country.

Scale

- Methods that one can use to develop programs of a few hundred lines cannot be expected to work when software of a few hundred thousand lines needs to be developed.
- Any large project involves the use of engineering and project management.
- In small projects, informal methods for development and management can be used, but for large projects, both have to be much more formal.

Scale

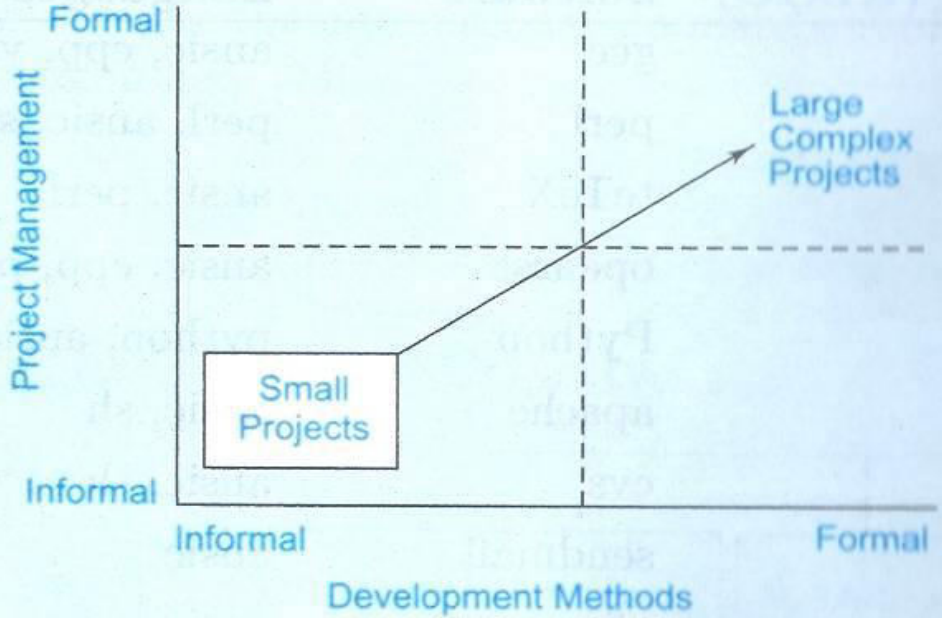

Figure 1.3: The problem of scale.

Scale

- As shown in figure, when dealing with a small software project, the engineering capability required is low and the project management requirement is also low.
- When scale changes to large, to solve such problems properly, both need to be formal.
- We can say that a project is small if its size is less than 10KLOC, medium if its size is less than 100KLOC, large if the size is less than one million LOC, and very large if the size is many million LOC.

Quality and Productivity

- Software Engineering is driven by three major factors : Cost, Schedule and Quality.
- The cost of a software project is often measured in terms of person-months.
- Person-months can be converted into a dollar amount by multiplying it with the average dollar cost, including the cost of overheads like hardware and tools, of one person-month.

Quality and Productivity

- Schedule is also an important factor. The cycle time from concept to delivery should be small.
- For software, this means that it needs to be developed faster.
- Productivity can capture both cost and schedule concerns.
- If productivity is higher, it means that cost in terms of person-months will be lower and development of the software in shorter time.

Quality

- Developing high-quality software is another fundamental goal of software engineering.
- According to the quality model adopted by the international standard, software quality comprises of six main attributes.
- They are also called as characteristics of good quality software.

Characteristics of Good Quality Software

- **Functionality :** The capability to provide functions which meet stated and implied needs when the software is used.
- **Reliability :** The capability to maintain a specified level of performance.
- **Usability :** The capability to be understood, learned, and used.
- **Efficiency :** The capability to provide appropriate performance relative to the amount of resources used.
- **Maintainability :** The capability to be modified for purposes of making corrections, improvements, or adaptation.
- **Portability :** The capability to be adapted for different specified environments without applying actions or means other than those provided for this purpose in the product.

Characteristics of Good Quality Software

- The concept of quality is project-specific.
- For an ultra-sensitive project, reliability may be of utmost importance but not usability, while in a commercial package for playing games on a PC, usability may be of utmost importance and not reliability.
- For each software development project, a quality objective must be specified before the development starts, and the goal of the development process should be to satisfy that quality objective.

Consistency and Repeatability

- An organization involved in software development not only wants high quality and productivity, but it wants these consistently.
- Consistency of performance allows an organization:
  - to predict the outcome of a project with reasonable accuracy,
  - to improve its processes to produce higher-quality products and to improve its productivity.

Consistency and Repeatability

- Within an organization, consistency is achieved by using its chosen methodologies in a consistent manner.
- Frameworks like ISO9001 and the Capability Maturity Model (CMM) are used for this task.

Change

- As the world/business changes faster, software has to change faster.
- Expectation is much more from software for change, because it is easy to change due to its lack of physical properties that may make changing harder.
- The challenge for software engineering is to accommodate and embrace change.
- The software that cannot accept and accommodate change is of little use today.

The Software Engineering Approach

- We can view high quality and productivity (Q&P) as the basic objective of Software project.
- Three main forces that govern Q&P are **People, processes,** and **technology**, often called the Iron Triangle.

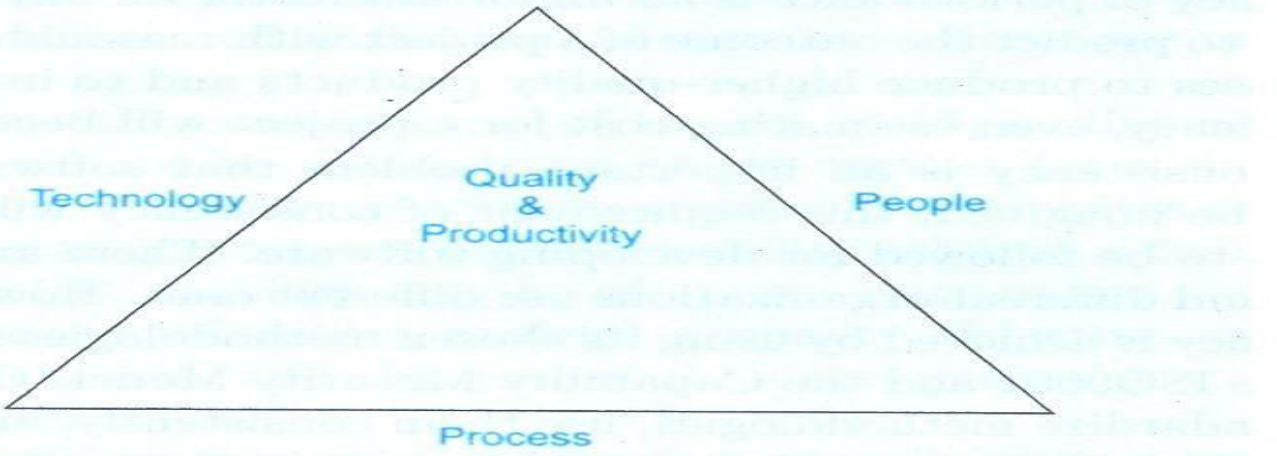

Figure 1.5: The iron triangle.

The Software Engineering Approach

- For high Q&P good technology has to be used, good processes or methods have to be used, and people doing the job have to be properly trained.
- In software engineering, the focus is primarily on processes, which were referred to as systematic approach in the definition given earlier.
- Hence, to tackle the problem domain and successfully face the challenges that software engineering faces, one must focus on the software process.

Phased Development Process

- A development process consists of various phases, each phase ending with a defined output.
- The main reason for having a phased process is that
  - it breaks the problem of developing software into successfully performing a set of phases
  - each handling a different concern of software development
- This ensures that the cost of development is lower than what it would have been if the whole problem was tackled together.
- Furthermore, a phased process allows proper checking for quality and progress at some defined points during the development.

Phased Development Process

- Without this, one would have to wait until the end to see what software has been produced.
- Hence, for managing the complexity, project tracking, and quality, all the development process consist of a set of phases.
- Any problem solving in software must consist of requirement specification for understanding and clearly stating the problem, design for deciding a plan for a solution, coding for implementing the planned solution, and testing for verifying the programs.
- Systematic approaches require that each of these four problem solving activities be done formally.

Requirements Analysis

- Requirements analysis is done in order to understand the problem that the software system is to solve.
- The emphasis in requirements analysis is on identifying ***what*** is needed from the system, not ***how*** the system will achieve its goals.
- The goal of the requirements activity is to document the requirements in a software requirements specification document.
- There are two major activities in this phase : problem understanding or analysis and requirement specification.

Requirements Analysis

- In problem analysis, the aim is to understand the problem and its context, and the requirements of the new system that is to be developed.
- Understanding the requirements of a system that does not exist is difficult and requires creative thinking.
- Once the problem is analyzed and the essentials understood, the requirements must be specified in the requirement specification document.

Requirements Analysis

- The requirements document must specify
  - all functional and performance requirements;
  - the formats of inputs and outputs; and
  - all design constraints that exist due to political, economic, environmental, and security reasons.
- In other words, besides the functionality required from the system, all the factors that may effect the design and proper functioning of the system should be specified in the requirements document.

Software Design

- The purpose of the design phase is to plan a solution of the problem specified by the requirements document.
- This phase is the first step in moving from the problem domain to the solution domain. Starting from what is needed, design takes us toward how to satisfy the needs.
- The design activity often results in three separate outputs – architecture design, high level design and detailed design

Software Design

- Architecture focuses on looking at a system as a combination of many different components, and how they interact with each other to produce the desired results.
- High level design identifies the modules that should be built for developing the system and the specifications of these modules.
- In detailed design, the internal logic of each of the modules is specified.
- A ***design methodology*** is a systematic approach to create a design by application of a set of techniques and guidelines.

Coding

- The goal of the coding phase is to translate the design of the system into code in a given programming language.
- The coding phase affects both testing and maintenance profoundly. Well written code can reduce the testing and maintenance effort.
- The goal of coding should be to reduce the testing and maintenance effort.
- During coding the focus should be on developing programs that are easy to read and understand, and not simply on developing programs that are easy to write.

Testing

- Testing is the major quality control measure used during software development. Its basic function is to detect defects in the software.
- The goal of testing is to uncover requirement, design and coding errors in the programs.
- The starting point of testing is ***unit testing***, where the different modules or components are tested individually.
- As modules are integrated into the system, ***integration testing*** is performed, which focuses on testing the interconnection between modules.

Testing

- When system is put together, **system testing** is performed. Here the system is tested against the system requirements to see if all the requirements are met and if the system performs as specified by the requirements.
- Finally, **acceptance testing** is performed to demonstrate to the client, on the real-life data of the client, the operation of the system.
- Testing process starts with a test plan. It specifies conditions that should be tested, different units to be tested, and the manner in which the modules will be integrated.
- Then for different test units, a ***test case specification document*** is produced, which lists all the different test cases, together with the expected outputs.
- The final output of the testing phase is the ***test report*** and the ***error report***, or a set of such reports.

Managing the Process

- The management activities typically revolve around a plan.
- A software plan forms the baseline that is heavily used for monitoring and controlling the development process of the project.
- Without proper project planning a software project is very unlikely to meet its objectives.
- For effectively managing a process, objective data is needed. For this, software metrics are used.

Software Metrics

- Software metrics are quantifiable measures that could be used to measure different characteristics of a software system or the software development process.
- There are two types of metrics used for software development: Product metrics and process metrics.
- Product metrics are used to quantify characteristics of the product being developed. (i.e. software)
- Process metrics are used to quantify characteristics of the process being used to develop the software
- Process metrics aim to measure such considerations as productivity, cost and resource requirements, and the effect of development techniques and tools.

Software Processes

- In software engineering, the phrase software process refers to the methods of developing software.
- A software process is a set of activities, such that if the activities are performed properly and in accordance with the ordering constraints, the desired result is produced.
- The process that deals with the technical and management issues of software development is called a software process.
- Many different types of activities need to be performed to develop software. All these activities together comprise the software process.

Processes and Process Models

- A successful project is the one that satisfies the expectations on all the three goals of cost, schedule, and quality.
- A process model specifies a general process, usually as a set of stages in which a project should be divided, the order in which the stages should be executed, and any other constraints and conditions on the execution of stages.
- A process is a means to reach the goals of high quality, low cost and low cycle time, and a process model provides generic guidelines for developing a suitable process for a project.

Component Software Processes

- There are two major components in a software process – A development process and A project management process.
- The development process specifies the development and quality assurance activities that need to be performed, whereas the management process specifies how to plan and control these activities so that cost, schedule, quality and other objectives are met.
- To handle evolution and changes software configuration control process is often used

Component Software Processes

- These three constituent processes focus on the projects and the products and can be considered as comprising the product engineering process, as their main objective is to produce the desired product.
- The basic objective of the process management process is to improve the software process. By improvement, we mean that the capability of the process to produce quality goods at low cost is improved.

Component Software Processes

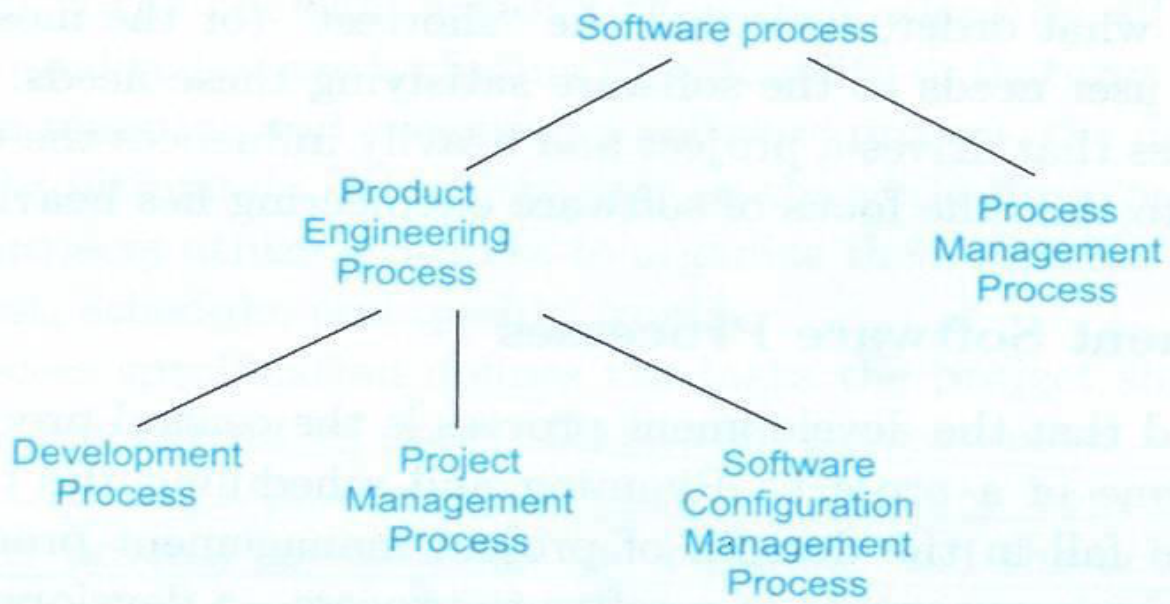

Figure 2.1: Software processes.

ETVX Approach for Process Specification

- A process has a set of phases, each phase performing a well-defined task which leads a project towards satisfaction of its goals.
- Outputs of a development process, which are not the final output, are called as work products. In software, a work product can be the requirements document, design document, code etc.

ETVX Approach for Process Specification

- As a process typically contains a sequence of steps, it is important to distinguish that when a phase should be initiated and terminated.
- This is frequently done by specifying the entry criteria and exit criteria for a phase.
- The entry criteria of a phase specifies the conditions that the input to the phase should satisfy to initiate the activities of that phase.
- The exit criteria specifies the conditions that the work product of this phase should satisfy to terminate the activities of the phase.
- The entry criteria of a phase should be consistent with the exit criteria of the previous phase.

ETVX Approach for Process Specification

- The specification of a step with its input, output, and entry and exit criteria is shown in figure.
- This approach for process specification is called the ETVX (Entry criteria, Task, Verification, and eXit

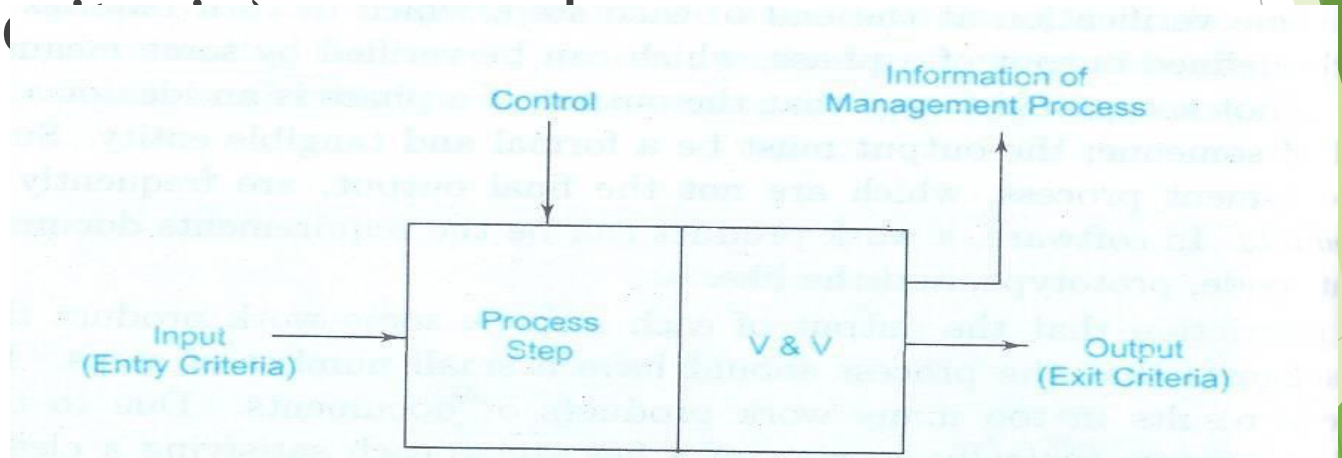

Figure 2.2: A step in a development process.

Desired Characteristics of software process

- Predictability
- Support Testability and Maintainability
- Support Change
- Early Defect Removal
- Process Improvement and Feedback

Predictability

- Predictability of a process determines how accurately the outcome of following that process in a project can be predicted before the project is completed.
- Predictability can be considered a fundamental property of any process. If a process is not predictable, it is of limited use.
- A predictable process is also said to be under statistical control.
- A process is under statistical control if following the same process produces similar results – results will have some variation, but the variation is mostly due to random causes and not due to process issues.

Predictability

- It should be clear that if one hopes to consistently develop software of high quality at low cost, it is necessary to have a process that is under statistical control.
- A predictable process is an essential requirement for ensuring good quality and low cost.

Support Testability and Maintainability

- The goal of development should be to reduce the maintenance effort. That is, one of the important objectives of the development project should be to produce software that is easy to maintain.
- The process used should ensure this maintainability.
- Overall, we can say that the goal of the process should to reduce the cost of testing and maintenance.
- Both testing and maintenance depend heavily on the quality of design and code, and these costs can be reduced if the software is designed and coded to make testing and maintenance easier.

Support Change

- Here we focus on changes due to requirement changes.
- As organizations and businesses change, the software supporting the business has to change. Hence, any model that builds software and makes change very hard will not be suitable in many situations.
- If project is of any significant duration, considerable changes can be expected.
- Changes may occur simply because people may change their minds as they think more about possibilities and alternatives.
- So, the process that can handle change easily is desirable.

Early Defect Removal

- The error occurrences can be distributed like 20% in Requirements, 30% in Design and 50% in Coding.
- The cost of correcting errors of different phases is not the same and depends on when the error is detected and corrected.
- If there is an error in the requirements, then the design and the code will be affected by it. To correct the error after the coding is done, would require both the design and the code to be changed, thereby increasing the cost of correction.

Cost of correcting errors

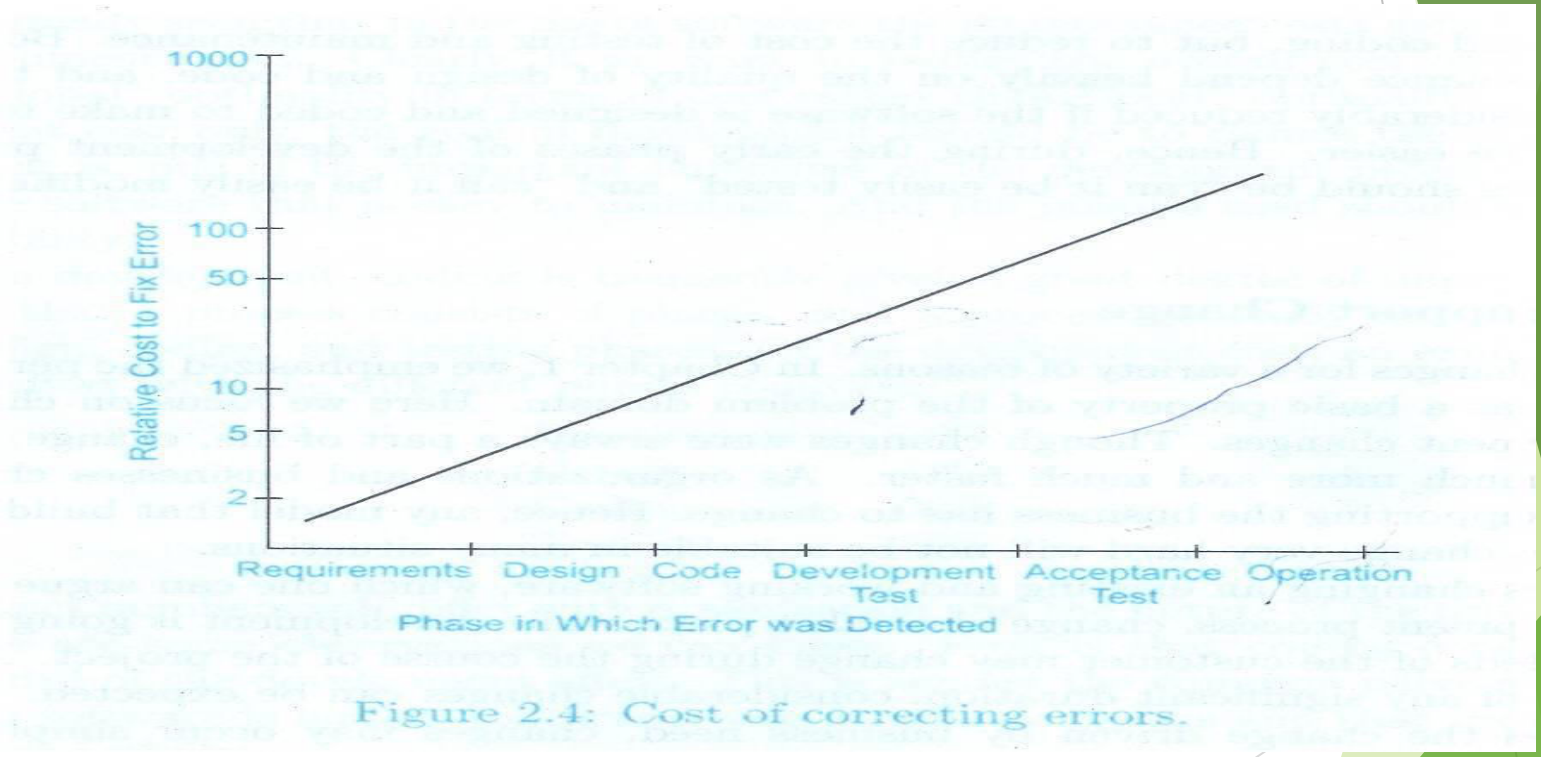

Figure 2.4: Cost of correcting errors.

Early Defect Removal

- We should attempt to detect errors that occur in a phase during that phase itself and should not wait until testing to detect errors.
- Error detection and corrections should be a continuous process that is done throughout software development.
- This means that we should try to verify the output of each phase before starting with the next.
- In other words, a process should have quality control activities spread through the process and in each phase.
- A quality control (QC) activity is one whose main purpose is to identify and remove defects.

Process Improvement and Feedback

- In the context of software, as the productivity and quality are determined largely by the process, to satisfy the objectives of quality improvement and cost reduction the software process must be improved.
- The project must be improved based on previous experiences, and each project done using the existing process must feed information.
- Process improvement is also an objective in a large project where feedback from the early parts of the project can be used to improve the execution of the rest of the project. (used in iterative model)

Software Development Process Models

- Waterfall Model
- Prototyping
- Iterative Enhancement Model
- Spiral model
- Time boxing Model

Waterfall Model

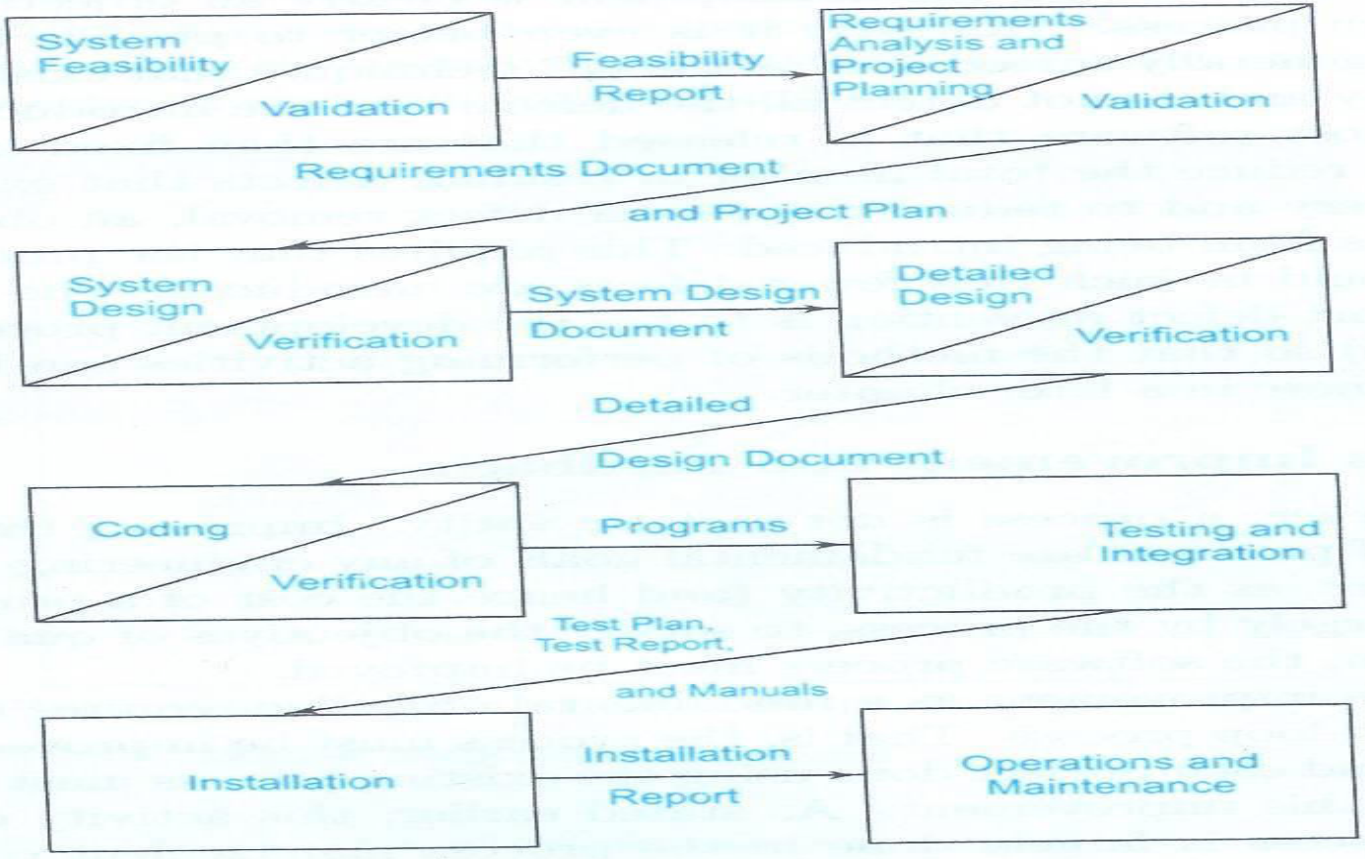

Figure 2.5: The waterfall model.

Waterfall Model

- The simplest process model is the waterfall model, which states the phases are organized in a linear order.
- The model was proposed by Royce.
- In this model, a project begins with feasibility analysis. Upon successfully demonstrating the feasibility of a project, the requirements analysis and project planning begins.
- The design starts after the requirements analysis is complete, and coding begins after the design is complete.
- Once the programming is completed, the code is integrated and testing is done.
- Upon successful completion of testing, the system is installed. After this, the regular operation and maintenance of the system takes place.

Waterfall Model

- Linear ordering of activities has some important consequences. First, to clearly identify the end of a phase and the beginning of the next, some certification mechanism is needed.
- This is usually done by some verification and validation means that will ensure that the output of a phase is consistent with its input.
- Following is the set of documents those are generally produced in a project.
- Requirement document , Project plan, Design document, Test plan and Test reports, Final code, etc.
- In addition of these work products, there are various other documents like Review reports and status reports, that are produced in a typical project.

Advantages of Waterfall Model

- The main advantage of this model is its simplicity.
- It is also easy to administer in a contractual setup – as each phase is completed and its work product produced, some amount of money is given by the customer to the developing organization.

Limitations of Waterfall Model

- It assumes that the requirements of a system can be frozen, which is not possible for new systems. Having unchanging requirements is unrealistic for such system.
- Freezing the requirements usually requires choosing the hardware. A large project might take a few years to complete. If the hardware is selected early, then due to the speed at which hardware technology is changing, it is likely that the final software will use a hardware technology which would be out dated at that time.

Limitations of Waterfall Model

- It follows “big – bang” approach. The entire software is delivered in one shot at the end. This has heavy risks, as the user does not know until the very end what they are getting. Furthermore, if the project runs out of money in the middle, then there will be no software.
- It is a document-driven process that requires formal documents at the end of each phase.

Despite these limitations, the waterfall model has been the most widely used process model. It is well suited for routine types of projects where the requirements are well understood.

Prototyping

- The goal of a prototyping-based development process is to counter the first two limitations of the waterfall model.
- The basic idea here is that instead of freezing the requirements before any design or coding can proceed, a throwaway prototype is built to help understand the requirements.
- This prototype is developed based on the currently known requirements. Development of the prototype undergoes design, coding and testing, but each of these phases is not done very formally.
- By using this prototype, the client can get an actual feel of the system, because the interactions with the prototype can enable the client to better understand the requirements of the desired system.

The Prototyping Model

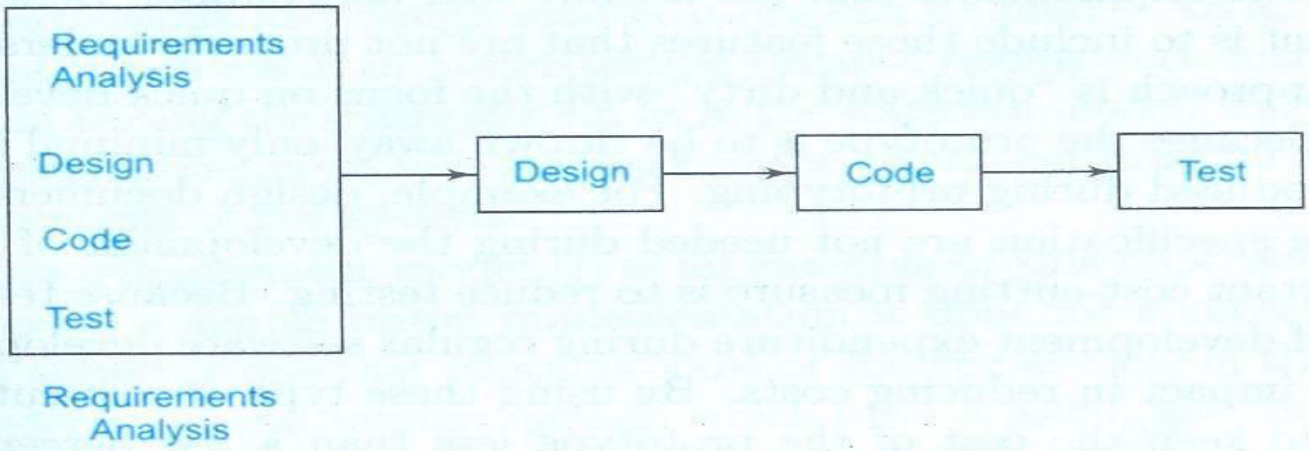

Figure 2.6: The prototyping model.

The Prototyping Model

- The development of the prototype typically starts when the preliminary version of the requirements specification document has been developed.
- At this stage, there is a reasonable understanding of the system and its needs and which needs are unclear or likely to change.
- After the prototype has been developed, the end users and clients are given an opportunity to use the prototype and play with it.
- Based on their experience, they provide feedback to the developers regarding the prototype.
- Based on the feedback, the prototype is modified to incorporate some of the suggested changes that can be done easily, and then the users and the clients are again allowed to use the system.

The Prototyping Model

- This cycle repeats until, in the judgment of the prototypers and analysts, the benefit from further changing the system and obtaining feedback is outweighed by the cost and time.
- To reduce the cost of prototyping, we can do following two things: 1. The focus of the development is to include those features that are not properly understood. 2. Another important cost-cost cutting measure is to reduce testing.
- Prototyping is often not used, as it is feared that development costs may become large.
- Prototyping is well suited for projects where requirements are hard to determine and the confidence in the stated requirements is low.

Iterative Development

- The iterative development process model counters the third limitation of the waterfall model and tries to combine the benefits of both prototyping and the waterfall model.
- The basic idea is that the software should be developed in increments, each increment adding some functional capability to the system until the full system is implemented.
- An advantage of this approach is that it can result in better testing because testing each increment is likely to be easier than testing the entire system as in the waterfall model.
- Furthermore, as in prototyping, the increments provide feedback to the client that is useful for determining the final requirements of the system.

The iterative enhancement model

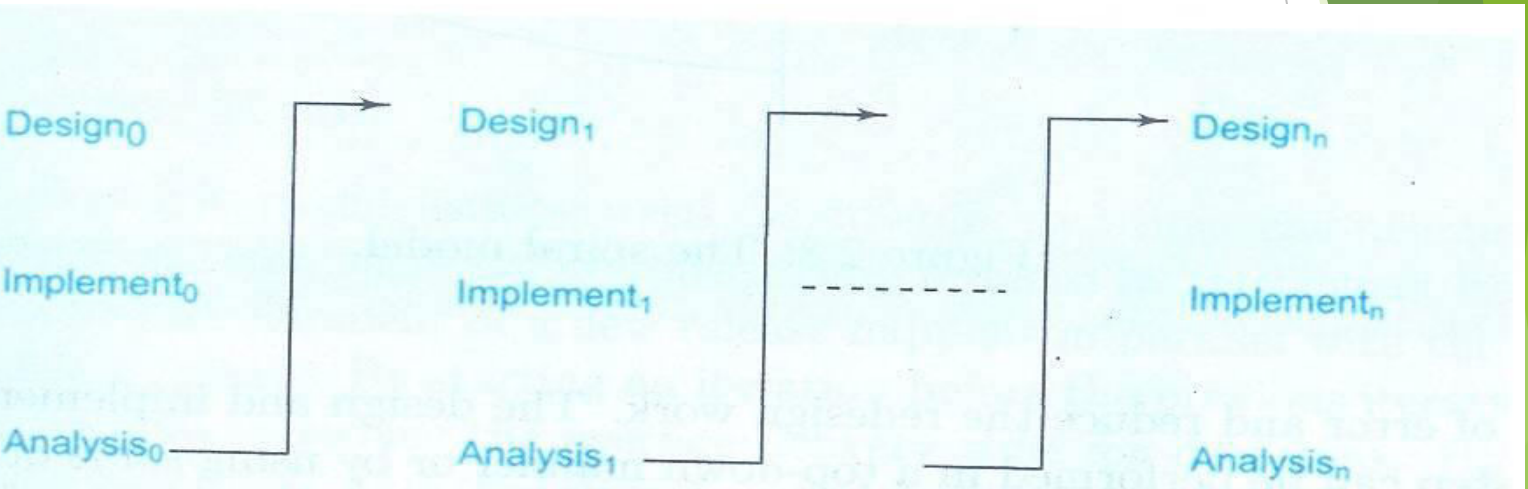

Figure 2.7: The iterative enhancement model.

Iterative Development

- In the first step of this mode, a simple initial implementation is done for a subset of the overall problem.
- This subset is one that contains some of the key aspects of the problem that are easy to understand and implement and which form a useful and usable system.
- A project control list is created that contains, all the tasks that must be performed to obtain the final implementation.
- This project control list gives an idea of how far along the project is at any given step from the final system

Iterative Development

- Each step consists of removing the next task from the list, designing the implementation for the selected task, coding and testing the implementation, performing an analysis of the partial system obtained after this step, and updating the list as a result of the analysis.
- These three phases are called the design phase, implementation phase, and analysis phase.
- The process is iterated until the project control list is empty, at which time the final implementation of the system will be available.
- Each entry in the list is a task that should be performed in one step of the iterative enhancement process and should be simple enough to be completely understood.

Spiral model

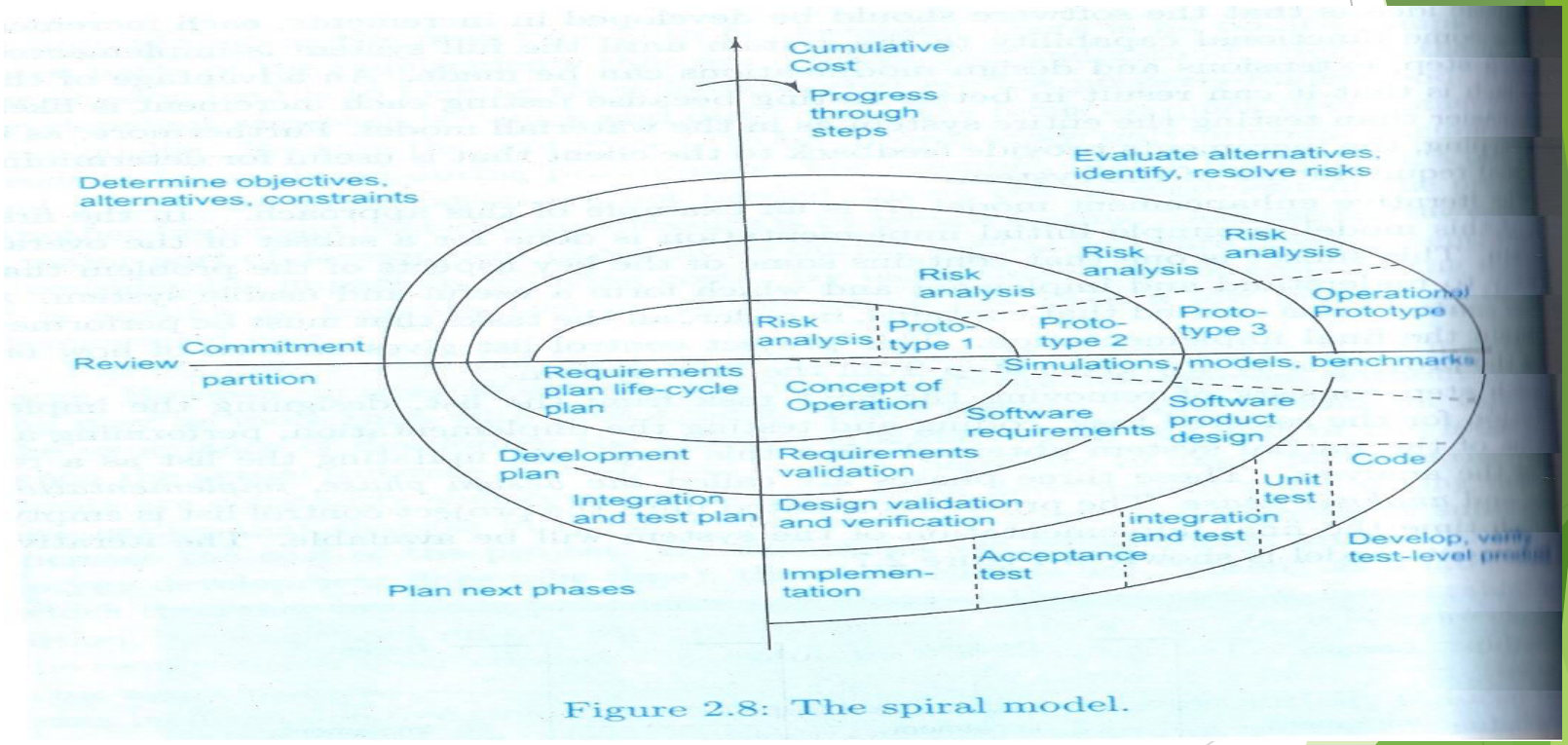

Figure 2.8: The spiral model.

Spiral model

- The spiral model is another iterative model that has been proposed. As the name suggests, the activities in this model can be organized like a spiral that has many cycles as shown in figure.
- Each cycle in the spiral begins with the identification of objectives for that cycle, the different alternatives that are possible for achieving the objectives, and the constraints that exist.
- The next step in the cycle is to evaluate these different alternatives based on the objectives and constraints.
- The focus of evaluation in this step is based on the risk perception for the project.

Spiral model

- The next step is develop strategies that resolve the uncertainties and risks. This step may involve activities such as benchmarking, simulation and prototyping.
- Next, the software is developed, keeping in mind the risks.
- Finally the next stage is planned.
- In a customized software development, where the client has to provide and approve the specifications, these process models are becoming more popular.
- Businesses are changing very rapidly today, they never really know the complete requirements for the software, and they do not want to invest too much for a long time without seeing returns.

Timeboxing model

- To speed up development, parallelism between the different iterations can be employed.
- By starting an iteration before the previous iteration has completed, it is possible to reduce the average delivery time for iterations.
- In the timeboxing model, the basic unit of development is a time box, which is of fixed duration.
- Since the duration is fixed, a key factor in selection the requirements or features to be built in a time box is what can be fit into the time box. This is in contrast to regular iterative approaches where the functionality is selected and then the time to deliver is determined.

Timeboxing model

- Each time box is divided into a sequence of stages, like in the waterfall model. Each stage preforms some clearly defined task for the iteration and produces a clearly defined output.
- The model also requires that the duration of each stage, the time it takes to complete the task of that stage, is approximately the same. Furthermore, the model requires that there be a dedicated team for each team for each stage.

Timeboxing model

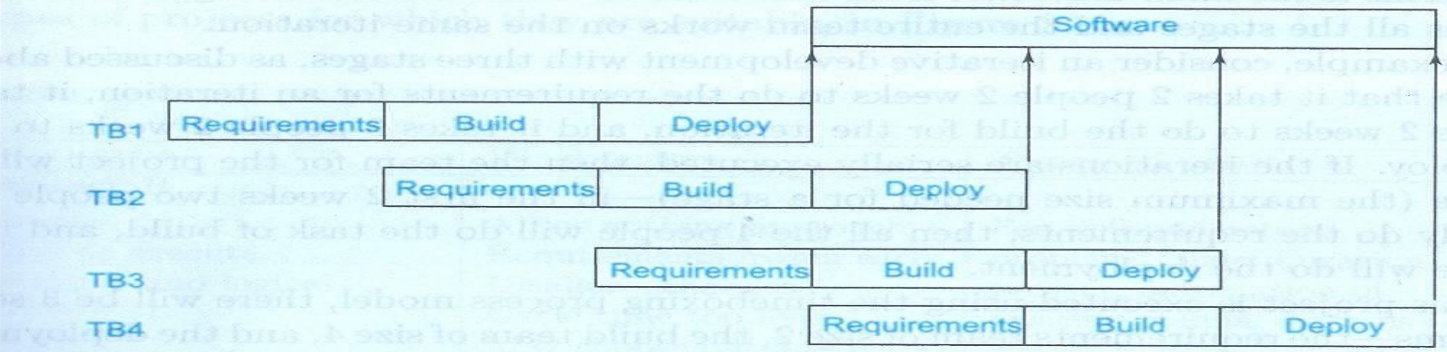

Figure 2.9: Executing the timeboxing process model.

after 12 weeks, the third after 15 weeks, and so on. Contrast this with a linear executio of iterations, in which the first delivery will be made after 9 weeks, the second will b made after 18 weeks, the third after 27 weeks, and so on.

There are three teams working on the project—the requirements team, the buil team, and the deployment team. The team-wise activity for the 3-stage pipelin discussed above is shown in Figure 2.10 [99].

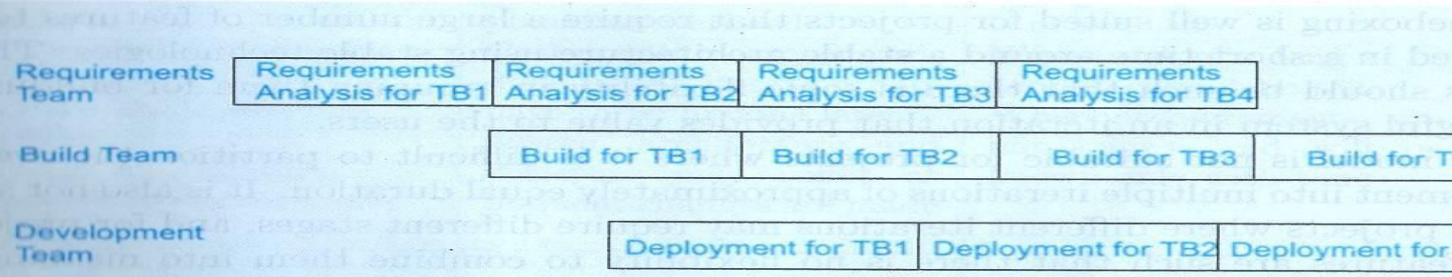

Figure 2.10: Tasks of different teams.

Timeboxing model

- As an example, consider a time box consisting of three stages : requirement specification, build, and deployment.
- The requirement stage is executed by its team of analysts.
- The build team develops the code for implementing the requirements, and performs the testing.
- The tested code is then handed over to the deployment team, which performs pre-deployment tests, and then installs the system for production use.

Timeboxing model

- When the requirement team has finished requirements for timebox-1, the requirements are given to the build team for building the software.
- The requirements team then goes on and starts preparing the requirements for timebox-2.
- When the build for the timebox-1 is completed, the code is handed over to the deployment team, and the build team moves on to build code for requirements for timebox-2, and the requirements team moves on to do requirements for timebox-3.
- This pipelined execution of the timeboxing process is shown in figure.

Comparison of models

Strengths	Weaknesses	Types of projects
Waterfall Simple Easy to execute Intuitive and logical	All or nothing approach Requirements frozen early Disallows changes Cycle time too long May choose outdated hardware technology User feedback not allowed Encourages req. bloating	For well understood problems, short duration project, automation of existing manual systems
Prototyping Helps in requirements elicitation Reduces risk Leads to a better system	Front heavy process Possibly higher cost Disallows later changes	Systems with novice users When uncertainities in requirements When UI very important
Iterative Regular/quick deliveries Reduces risk Accommodates changes Allows user feedback Allows reasonable exit points Avoids req. bloating Prioritizes requirements	Each iteration can have planning overhead Cost may increase as work done in one iteration may have to be undone later System architecture and structure may suffer as frequent changes are made	For businesses where time is of essence Where risk of a long project cannot be taken Where requirements are not known and will be known only with time
Timeboxing All strengths of iterative Planning and negotiations somewhat easier Very short delivery cycle	Project management is complex Possibly increased cost Large team size	Where very short delivery times needed Flexibility in grouping features exists

Figure 2.11: Comparison of process models.

Other software Processes

- Though the development process is the central process in software process, other processes are needed to properly execute the development process and to achieve the desired characteristics of software processes.
- Following are the important process that are involved when developing software.
- 1. Project Management Process 2. The inspection Process 3. Software Configuration Management Process 4. Requirements Change Management Process 5. Process Management Process

Software Reuse

- Reuse-based software engineering is a software engineering strategy where the development process is geared to reusing existing software.
- The move to reuse-based development has been in response to demands for lower software production and maintenance costs, faster delivery of systems and increased software quality.
- This approach tries to maximize the reuse of existing software.
- Systems in the same application domain are similar and have potential for reuse, that reuse is possible at different levels, and that standards for reusable components facilitate reuse.

Software Reuse

- The software units that are reused may be of different sizes.
- ***Application system reuse***: the whole of an application system may be reused by incorporating it without change into other system, by configuring the application for different customers or by developing application families that have a common architecture but are tailored for specific customers.
- ***Component reuse*** : components of an application ranging in size from sub-systems to single objects may be reused.
- ***Object and function reuse***: This form of reuse, based around standard libraries, has been common for the past many years.

Advantages of Software Reuse

- ***Development cost reduced***
- ***Increased dependability***: reused software is already tried and tested in working systems.
- ***Reduced process risk***: the cost of existing software is already known, while the cost of development is matter of judgment.
- ***Effective use of specialists***
- ***Standards compliance***
- ***Accelerated development***

Disadvantages of Reuse

- ***Increased maintenance costs***: if the source code of a reused software system or component is not available.
- ***Lack of tool support***: new tools may not work with existing code.
- ***Creating and maintaining a component library***: populating a reusable component library and ensuring the software developers can use this library can be expensive.
- ***Finding, understanding and adapting reusable components***:

Examples of Software Reuse

- Software product lines: Enterprise Resource Planning (ERP)
- Class and function libraries implementing commonly used abstractions are available for reuse.
- Application System reuse: Commercial-Off-The-Shelf (COTS) product is a software system that can be used without change by its buyer. DBMS and APIs are best examples of it.

Web Engineering

- In early days of the WWW, web sites consisted of little more than a set of linked hypertext files that presented information using text and limited graphics.
- As time passed, HTML was augmented by development tools that enabled web engineers to provide computing capability along with information.
- Web based systems and applications have evolved into sophisticated computing tools that not only provide standalone function to the end-user, but also have been integrated with corporate databases and business applications.

Attributes of Web Apps

- Network intensiveness: internet or intranet
- Concurrency:
- Unpredictable load:
- Performance: users may have to wait
- Availability: 24/7/365
- Data driven: text, graphics, audio, video
- Content sensitive: the quality and aesthetic nature of content remains an important determinant of the quality of a WebApp.
- Continuous evolution:
- Immediacy: web engineers must use methods for planning, analysis, design, implementation, and testing that have been adapted to the compressed time schedules required for WebApp development.
- Security:
- Look and feel:

Categories of Webapp

- Informational: read only content
- Download: user downloads information
- Customizable: the user customizes content to specific needs.
- Interaction: like chat room, bulletin boards or instant messaging.
- User input: forms-based input
- Transaction-oriented: user request is fulfilled by WebApp
- Service-oriented: application assists user in his task
- Portal
- Database access
- Data warehousing

Characteristics of Process for WebE

Note: Any one of the agile process models like XP, Adaptive Software Development (ASD), SCRUM can be applied successfully as a WebE process.

- Embraces change
- Encourages the creativity and independence of development staff and strong interaction with WebApp stakeholders
- Builds systems using small development teams
- Emphasizes evolutionary or incremental development using short development cycles.

Web Engineering Team

- Following roles should be distributed among the members of the WebE team:
- **Content developers/providers** -> because WebApps are content driven, one role on the WebE team must focus on the generation and collection of content. Content providers may come from non-software background.
- **Web publisher** -> the diverse content generated by content developers must be organized for inclusion within the WebApp. This role is filled by the Web publisher, who must understand both content and WebApp technology.
- **Web engineer** -> involved in all activities and must have knowledge of technologies, client-server architecture, database, multimedia and hardware/software platforms, network security, and support.
- **Business Domain Experts**
- **Support Specialist**
- **Administrator** -> often called “Web Master”

Web Engineering Options

- Once formulation has occurred and basic WebApp requirements have been identified, a business must choose from one of two Web engineering options:

1. Outsourcing: the WebApp is outsourced-Web engineering is performed by a third party vendor who has the expertise, talent, and resources that may be lacking within the business. Or
2. In House Web Engineering: the WebApp is developed in-house using web engineers that are employed by the business.

- Third alternative, doing some web engineering work in-house and outsourcing other work is also an option.

Testing of WebApps

- ***Content*** is evaluated at both a syntactic and semantic level.
- ***Function*** is tested to uncover errors that indicate lack of conformance to customer requirements.
- ***Structure*** is assessed to ensure that it properly delivers WebApp content and function, and it can be supported as new content or functionality is added.
- ***Usability*** is tested to ensure that each category of user is supported by the interface.
- ***Navigability*** is tested to ensure that all navigation syntax and semantics are exercised to uncover any navigation errors like dead links, improper links etc.
- ***Performance*** is tested under a variety of operating conditions, configurations, and loading.
- ***Compatibility*** is tested by executing the WebApp in a variety of different host configurations on both the client and server side.
- ***Interoperability*** is tested to ensure that the WebApp properly interfaces with other applications and databases.
- ***Security*** testing.

Reengineering

- If custom built software is old, then it breaks too often and takes longer time to repair.
- Hardware can be thrown away and purchased newer one.
- Software reengineering is the examination and alteration of a system to reconstitute it in a new form.
- This process includes:
  - Reverse engineering
  - Restructuring
  - Re-documentation
  - Forward engineering

Agile Software Development

- Agile software development describes a set of principles for software development under which requirements and solutions evolve through the collaborative effort of self-organizing cross-functional teams.
- A cross-functional team is a group of people with different functional expertise working toward a common goal.
- It may include people from finance, marketing, operations, and human resources departments.
- Typically, it includes employees from all levels of an organization.
- Members may also come from outside an organization (in particular, from suppliers, key customers, or consultants).

Agile Computing

- Agile Software Development emphasises
  - adaptive planning
  - evolutionary development
  - early delivery
  - continuous improvement
  - rapid and flexible response to change

Agile Manifesto

- ***Individuals and interactions***: self-organization and motivation are important, as are interactions like co-location and pair programming.
- ***Working software***: working software is more useful and welcome than just presenting documents to clients in meetings.
- ***Customer collaboration***: requirements cannot be fully collected at the beginning of the software development cycle, therefore continuous customer or stakeholder involvement is very important.
- ***Responding to change***: agile methods are focused on quick responses to change and continuous development.

Agile v/s Traditional Computing

- Compared to traditional software engineering, agile software development mainly targets complex systems and product development with dynamic, non-deterministic and non-linear characteristics, where accurate estimates, stable plans, and predictions are often hard to get in early stages—and big up-front designs and arrangements would probably cause a lot of waste, i.e., are not economically sound.
- These basic arguments and previous industry experiences, learned from years of successes and failures, have helped shape agile development's favour of adaptive, iterative and evolutionary development.

Agile v/s Traditional Computing

- Agile methods are the adaptive .
- One key of adaptive development methods is a "Rolling Wave" approach to schedule planning, which identifies milestones but leaves flexibility in the path to reach them, and also allows for the milestones themselves to change.
- Adaptive methods focus on adapting quickly to changing realities.
- When the needs of a project change, an adaptive team changes as well.
- An adaptive team has difficulty describing exactly what will happen in the future.

Agile v/s Traditional Computing

- Traditional software computing methods are mostly predictive.
- *Predictive* methods focus on analysing and planning the future in detail and cater for known risks.
- In the extremes, a predictive team can report exactly what features and tasks are planned for the entire length of the development process.
- Predictive methods rely on effective early phase analysis and if this goes very wrong, the project may have difficulty changing direction.
- Predictive teams often institute a change control board to ensure they consider only the most valuable changes.

Agile Process models

- Extreme Programming (XP)
- Adaptive Software Development (ASD)
- Dynamic Systems Development Method (DSDM)
- Scrum
- Crystal
- Feature Driven Development (FDD)
- Agile Modeling (AM)