UNIT - I

Introduction: The evolving role of Software – Software characteristics, Software Process: Software process models - The linear sequential model - The prototyping model - The RAD model - Evolutionary software process models - The incremental model - The spiral model - Software applications - Software myths. Planning and Estimation: The Project Planning Process-Software Project Estimation- LOC based-FP- Based-Estimation for Object Oriented Project.

Objectives:

- **To learn about various software development models.**
- **To know about the project estimation techniques.**

Software Engineering is a process of developing and maintaining a software product in a cost effective and efficient way.

1.1 Introduction:

- Software is a set of complete programs which is used to perform a certain task.
- Software is a data structure that enables the programs to adequately manipulate information.
- Software is also defined as documents that describe the operation and use of the programs.
- System engineering is concerned with all aspects of computer based system development including hardware, software and process engineering.
- Software engineering is a part of system engineering.

1.2 The Evolving Role of Software:

- Software takes on a dual role.
- It is a product and a vehicle for delivering a product.

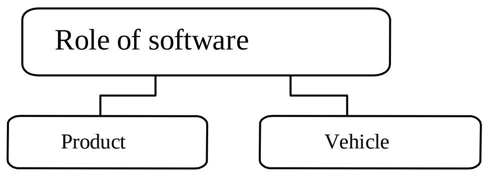


Fig: 1.1 Software classifications

Product:

As a product, it delivers the computing potential embodied by computer hardware or a network of computers that are accessible by local hardware. Whether it resides within a cellular phone or operates inside a mainframe computer, software is an information transformer - producing, managing, acquiring, modifying, displaying, or transmitting information that can be as simple as a single bit or as complex as a multimedia presentation.

Vehicle:

As the vehicle for delivering the product, software acts as the basis for the control of the computer(operating systems), the communication of information (networks), and the creation and control of other programs(software tools and environments).

- Software delivers the information.
- Software transforms personal data (e.g., an individual‘s financial transactions) 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., Internet) and provides the means for acquiring information in all of its forms.

The role of computer software has undergone significant change over a span of little more than 50 years. Dramatic improvements in hardware performance, profound changes in computing architectures, vast increases in memory and storage capacity and a wide variety of exotic input and output options have all precipitated more sophisticated and complex computer-based systems. A huge software industry has become a dominant factor in the economics of the industrialized world. The lone programmer of an earlier era has been replaced by teams of software specialists, each focusing on one part of the technology required to deliver a complex application.

The questions that were asked of the lone programmer are the same questions that are asked when modern computer based systems are built:

- 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 progress as software os being developed and maintained?

These questions and many others demonstrate the industry‘s concern about software and the manner in which it is developed - a concern that has lead to the adoption of software engineering practice.

1.3 Software Characteristics **Software** is:

- Instructions (computer programs) that when executed provide desired function and performance.
- Data structures that enable the programs to adequately manipulate information.
- Documents that describe the operation and use of the programs.

To gain an understanding of software, it is important to examine the characteristics of software that make it different from other things that human beings build. Software is a logical rather than a physical system element. Therefore, software has characteristics that are considerably different than those of hardware:

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

1. Software is developed or engineered; it is not manufactured in the classical sense.

Although some similarities exist between software development and hardware manufacture, the two activities are fundamentally different. In both activities, high quality is achieved through good design, but the manufacturing phase for hardware can introduce quality problems that are nonexistent (or easily corrected) for software. Both activities are dependent on people, but the relationship between people applied and work accomplished is entirely different. Both activities require the construction of a ―product‖, but the approaches are different. Software costs are concentrated in engineering. This means that software projects cannot be managed as if they were manufacturing projects.

2. Software doesn't "wear out."

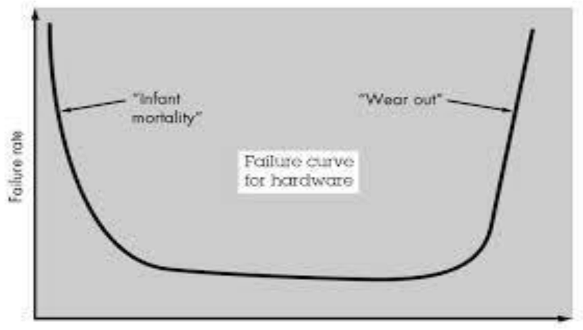


Fig: 1.2 Failure Curve for Hardware

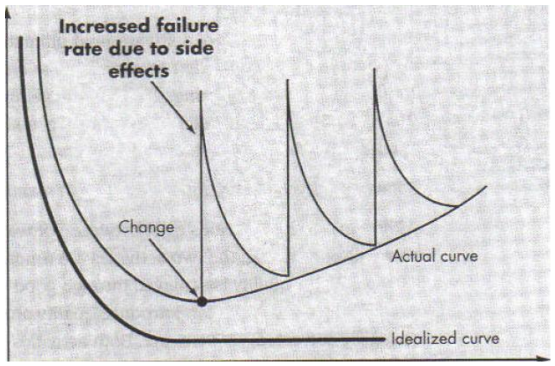


Fig: 1.3 Failure Curve for Software

The relationship, often called the "bathtub curve," indicates that hardware exhibits relatively high failure rates early in its life (these failures are often attributable to design or manufacturing defects); defects are corrected and the failure rate drops to a steady-state level (ideally, quite low) for some period of time. As time passes, however, the failure rate rises again as hardware components suffer from the cumulative effects of dust, vibration, abuse, temperature extremes, and many other environmental maladies. Stated simply, the hardware begins to wear out. Software is not susceptible to the environmental maladies that cause hardware to wear out

3. Although the industry is moving toward component-based assembly, most software continues to be custom built.

Consider the manner in which the control hardware for a computer-based product is designed and built. The design engineer draws a simple schematic of the digital circuitry, does some fundamental analysis to assure that proper function will be achieved, and then goes to the shelf where catalogs of digital components exist. Each integrated circuit (called an IC or a chip) has a part number, a defined and validated function, a well-defined interface, and a standard set of integration guidelines. After each component is selected, it can be ordered off the shelf.

1.4 .Software Process:

1.4.1 Layered Architecture of Engineering

- Software process is a set of activity which produces the software product.
- Software engineering is a **layered technology.**

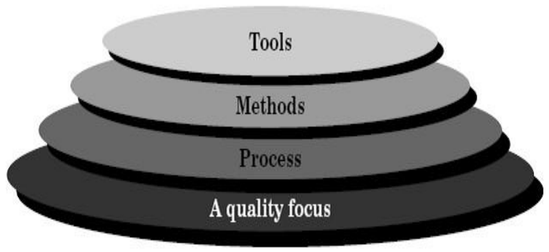


Fig 1.4 Layered technology

- The TQM, six sigma are a continuous process improvement culture, and the bedrock of software engineering is the **quality focus layer.**
- The foundation for software engineering is the **process** layer. Process defines a framework that must be established for effective delivery of software engineering technology.
- Software engineering **methods** provide the technical ideas for building software methods encompass broad array of tasks that include communication, requirement analysis, design modeling, program construction, testing and support.
- Software engineering **tools** provide automated or semi-automated support for the process and the methods.
- **CASE:** When tools are integrated so that information created by one tool can be used by another. This is called as **Computer Aided Software Engineering.**

1.4.2 Generic process Framework

The process frame work establishes the foundation for a complete software process by identifying a small number of framework activities that are applicable to all software projects, regardless of their

Communication:

This framework activity involves heavy communication and collaboration with the customer and encompasses requirements gathering and other related activities.

Planning:

This activity establishes a plan for the software engineering work that follows. It describes the technical tasks to be conducted, the risks that are likely, the resource that will be required, the work products to be produced and a work schedule.

Modeling:

This activity encompasses the creation of models that allow the developer and the customer to better understand software requirements and the design that will achieve those requirements.

Construction:

This activity combines code generation and the testing that is required to uncover errors in the code.

Deployment:

The software is delivered to the customer who evaluates the delivered product and provides feedback based on the evaluation.

These five generic framework activities can be used during the development of small programs, the creation of large web applications and for the engineering of large, complex computer-based systems.

The phase and related steps described in our generic view of software engineering are complemented by a number of umbrella activities. Typical activities in this category include:

- Software project tracking and control.
- Risk management.
- Software quality assurance.
- Software configuration management.
- Format technical reviews.
- Measurement.
- Reusability management.
- Work product preparation and production.

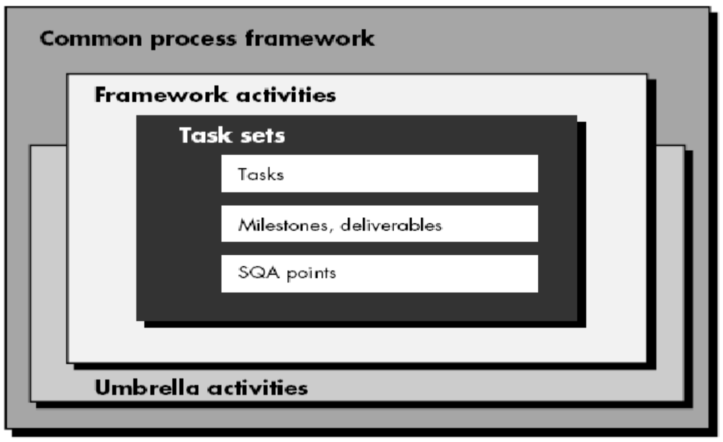


Fig: 1.5 Process frame work

1.5 Software Process Models:

A software process model populates a process framework with explicit set of tasks for software engineering actions. The different types of software engineering process models are:

- Linear sequential model.
- Prototyping model.
- RAD model.
- Evolutionary model.

- Incremental model.
- Spiral model.

1.6. The Linear Sequential Model:

The **waterfall model** is also called the classic life cycle, suggests a systematic sequential approach to software development that begins with customer specification of requirements and progress through planning, modeling, construction and deployment, culminating in on-going support of the completed software.

It can serve as a useful Process model in situations where requirements are fixed and work is to proceed to completion in a linear manner.

The waterfall model is applied in:

- Real projects rarely follow the sequential flow that the model proposes.
- It is often difficult for the customer to state all requirements explicitly.
- The customer must have patience.
- A working version of the program will not be available until late in the project time-span.

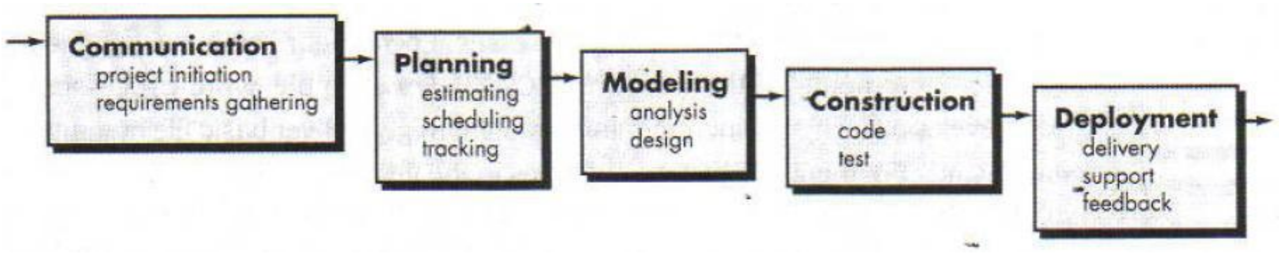


Fig: 1.6 Waterfall Model

1.7 The Prototyping Model:

- Prototype model focuses on producing software product quickly.
- A prototype paradigm may be the best approach in many situations.
- The developer may be ensured with the efficiency of the algorithm.
- The prototyping model begins with communication.
- In this phase the software engineers and customer must define the overall objective for the software and identify whatever requirements are known, outline the areas where further definition is mandatory.

- In quick plan phase, prototyping iteration is planned quickly & modeling occurs.
- The quick design process on a representation and the prototype is constructed.
- In final phase the prototype is developed and then evaluated by the customer loses.

Problem with the prototyping model:

- The customer sees what appears to be a working version of the software product. Hence software development management is necessary.
- The developer often makes the implementation quickly which results in the usage of inefficient algorithm.

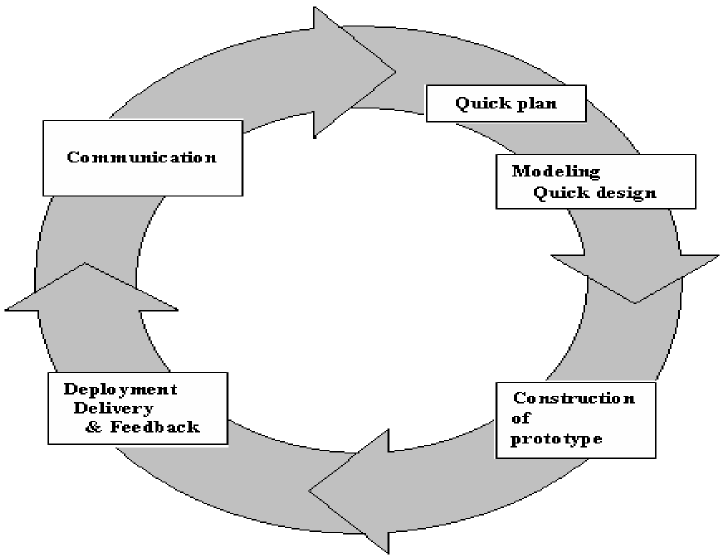


Fig. 1.7. Prototype Model

Advantage:

- Iteration process facilitating enhancements.
- Partial product can be viewed by the customer.
- Flexibility of the product.
- Customer satisfaction of the product.

Disadvantage:

- No optimal solution.
- Time consuming if algorithm used is inefficient.
- Poor documentation resulting with difficulty.

1.8 RAD Model :(Rapid Application Development)

- RAD is an incremental software process models that emphasis a short development cycle.

- In RAD model the rapid development is achieved by using a component based construction approach.
- The first phase is communication phase it works to understand the business problem & information characteristics.
- Planning phase is essential because multiple software teams work in parallel on different software function.

- Modeling encompasses three major phases.
    1. Component reuse.
    2. Automatic code generation.
    3. Testing.

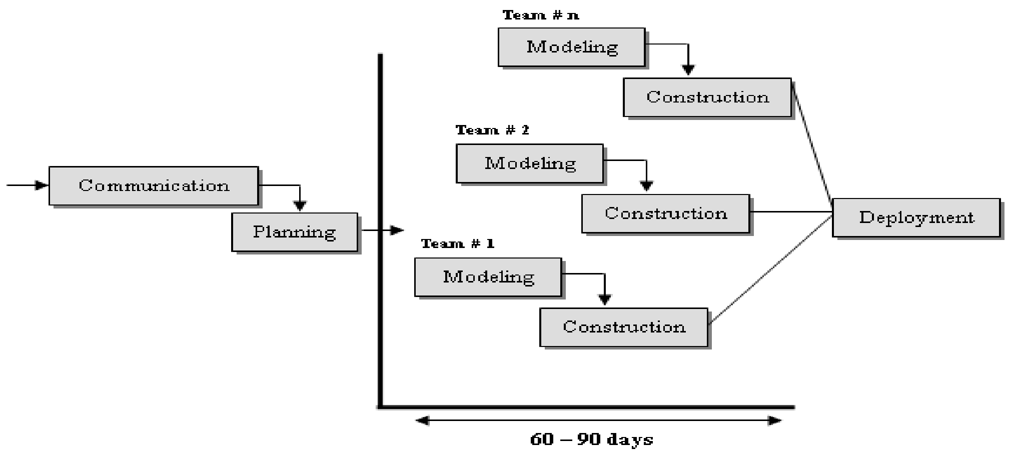


Fig: 1.8 RAD model

Drawbacks of RAD:

- For large but scalable projects RAD requires sufficient human resources to create the right number of RAD team.
- If developer and customer are not committed to the rapid-fire activities necessary to complete the software in a much abbreviated time frame, RAD project will fail.
- If a system cannot be properly modularized, building the component necessary for RAD will be problematic.
- If high performance is an issue & performance is to be achieved through tuning the interface to system component, the RAD approach may not work.
- RAD may not be appropriate when technical risks are high. Eg: when a new application makes necessary use of new technology.

1.9 Evolutionary Model:

- Referred as the successive version model. In evolutionary model the software is first broken into a several model.
- The development team first develops the core module of the software.
- The initial product skeleton is refined into increasing levels of capability by adding new functionality in the successive version.
- This modeling is very popular for projects because the system can easily be partitioned into
  standalone units in terms of the object.

Properties of evolutionary model:

- Continuing change in degradation.
- Increase in complexity.
- Program evaluation.
- Invariant evaluation.
- Incremental growth limit.

Continuing change in degradation:

Software system is continually changing which makes software become very less useful.

Increase in complexity:

Due to continual changes the software complexity increase, integrating various modules will be difficult.

Program evaluation:

Program, process and measure of project and system attribute are statistically self- regulatory with determinable tends.

Invariant work rate:

Rate of activity in large software project is statistically invariant.

During the life cycle of large software system volume of modification in successive release is statistically invariant.

Disadvantage:

- Difficult to divide problem into several unit**.**
- Used only for very large projects.

Advantages:

- Reduce errors

- User get chance to experiment with partially developed software much before the computer version of software, so the changes requested after the delivery are minimized.

1.10. The Incremental Model

- There are many situations in which initial software requirements are reasonably well-defined but the overall scope of the development effort precludes a purely linear process.
- The **incremental model** combines elements of the waterfall model applied in an iterative fashion. The incremental model applies linear sequences in a staggered fashion as calendar time progresses. Each linear sequence produces deliverable ―increments‖ of the software.
- When an incremental model is used, the first increment is often a **core product**. The incremental process model, like prototyping and other evolutionary approaches, is iterative in nature.
- Unlike prototyping, the incremental model focuses on the delivery of an operational product with each increment.

The figure 1.9 shows the linear sequences in a staged fashion as calendar time progresses of the incremental model.

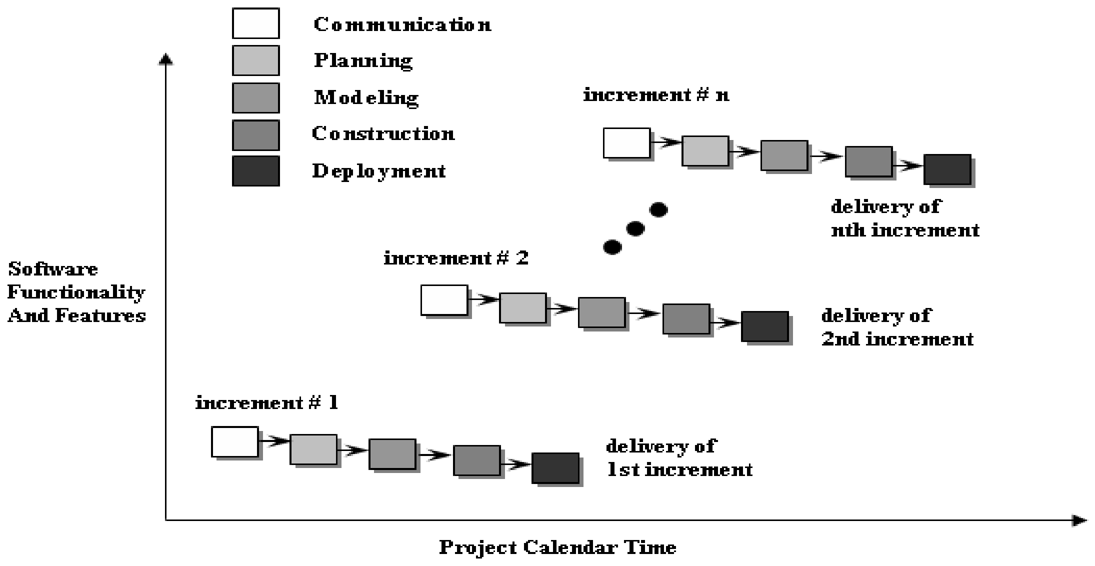


Figure: 1.9 Incremental Models

Advantages:

- Requirements are identified in every phase.
- Cost is distributed with less human power & staffing.
- Errors are simultaneously corrected.

- Feedback requirement is clear.
- Testing is easy.

Disadvantages:

- Requires proper planning to distribute the work.
- Total cost required for the development of the product is high.
- Interface model should be well-defined.

1.11 The Spiral Model

The **spiral model** is an evolutionary software process model that couples the iterative nature of prototyping with the controlled and systematic aspects of the waterfall model. It provides the potential for rapid development of increasingly more complete versions of the software.

The spiral model is a realistic approach to the development of large scale systems and software. The spiral development model is a risk-driven process model generator that is used to guide multi-stakeholder concurrent engineering of software intensive systems.

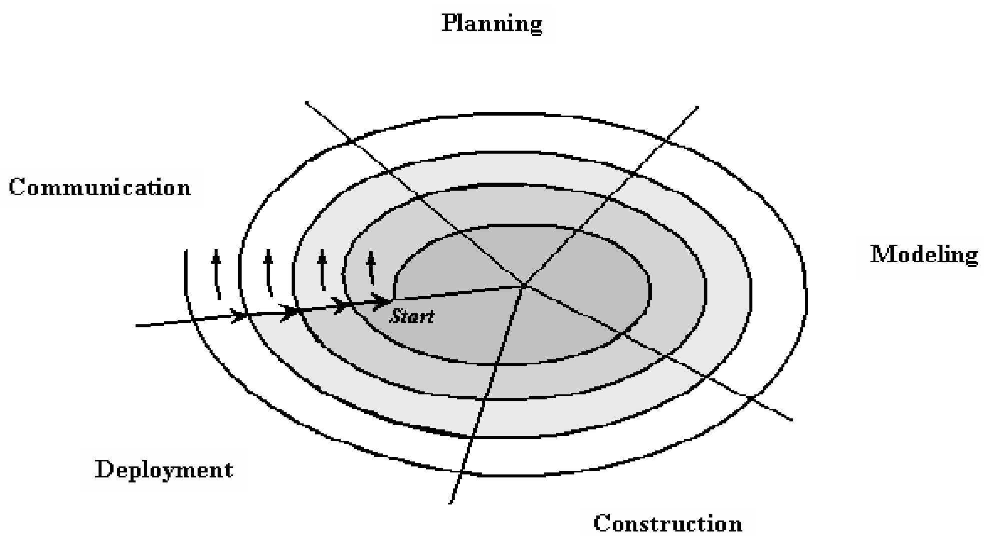


Figure: 1.11 Spiral model

Task set:

In this model each of the regions with the set of work tasks is called as task set. The size of the task set will vary according to the project.

Task region:

Task region is a region where set of task is achieved.

Principle of spiral model:

- Elaborate software entity object and find out constraints and alternatives.
- Elaborate definition of the software entity for the project.
- Spiral model terminate a project, if it is too risky.

Customer communication:

- Task requires establishing effective communication between developer and customer.

Planning:

- Task requires defining resources, time& other related information.

Risk information:

- Task requires to access both technical and management risk.

Engineering:

- Task required building one or more representation of the application.

Construction &release:

- The task requires constructing the project & releasing the project to the customer.

Customer evaluation:

- Task required obtaining customer feedback based on evolution of the software representation created during the installation stage.

Advantage:

- User will be able to see the project development cycle.
- Risk analysis which resolves higher priority error.
- Project is very much refined.
- Reusability of the software.

Disadvantages:

- It is only suitable for large size project.
- Model is more complex to use.
- Management skill is necessary so as to analyze the risk factor.

1.12 Software Applications:

- Software may be applied in any situation for which a pre-specified set of procedural steps (i.e.,
  an algorithm) has been defined.
- Information content and determinacy are important factors in determining the nature of a software application.
- Content refers to the meaning and form of incoming outgoing information.
- Information determinacy refers to the predictability of the order and timing of information.
- An engineering analysis program accepts data that have a predefined order executes the analysis algorithm without interruption and produces resultant data in report or graphical format.
- A multiuser operating system on the other hand accepts inputs that have varied content and arbitrary timing executes algorithms that can be interrupted by external conditions and produces output that varies as a function of environment and time.
- It is somewhat difficult to develop meaningful generic categories for software applications.
- As software complexity grows neat compartmentalization disappears.

The following software areas indicate the breadth of potential applications:

- System Software.
- Real-Time Software.
- Business Software.
- Engineering and Scientific Software.
- Embedded Software.
- Personal Computer Software.
- Artificial Intelligence Software.

System Software:

System software is a collection of programs written to service other programs. Some system software (e.g., compiler s editors and file management utilities) processes complex but determinate information structures. Other systems application (e.g., operating system components drivers telecommunications processors) process largely indeterminate data. In either case the systems software area is characterized by heavy interaction with computer hardware heavy usage by multiple users; concurrent operation that requires scheduling resource sharing and sophisticated process management.

Real-Time Software:

Programs that monitor/analyze/ control real world events as they occur are called real-time software. Elements of real-time software include a data gathering component that collects and formats information from an external environment an analysis component that transforms information as required by the application a control / output component that responds to the external environment so that real-time response (typically ranging from 1 millisecond to 1 minute) can be maintained. It should be noted that the term ―real-time‖ differs from ―interactive‖ or timesharing‖. A real-time system must respond within strict time constraints.

Business Software:

Business information processing is the largest single software application area. Discrete ―systems‖ have evolved into management information system (MIS) software that accesses one or more large databases containing business information. Applications in this area restructure existing data in a way that facilitates business operation or management decision making. In addition to conventional data processing applications, business software applications also encompass interactive and client/server computing.

Engineering and Scientific Software:

Engineering and Scientific software has been characterized by ―number crunching‖ algorithms. Application range from astronomy to volcanologist from automotive stress analysis to space shuttle orbital dynamics and from molecular biology to automated manufacturing. New applications with the engineering/scientific area are moving away from conventional numerical

algorithms. Computer aided design system simulation and other interactive applications have begun to take on real-time and even system software characteristics.

Embedded Software:

Intelligent products have become commonplace in nearly every consumer and industrial market. Embedded software resides in read only memory and is used to control products and systems for the consumer and industrial markets. Embedded software can perform very limited and esoteric functions (e.g., digital functions in an automobile such as fuel control, dashboard displays, braking systems, etc.).

Personal Computer Software:

The personal computer software market has burgeoned over the past decade. Word processing, spreadsheets, computer graphics, multimedia entertainment, database management personal and business financial applications and external network or database access are only a few of hundreds of application.

Artificial Intelligence Software:

Artificial Intelligence (AI) software makes use of non numerical algorithms to solve complex problems that are not amenable to computation or straight forward analysis. An active AI area is expert systems also called knowledge-based systems. However other application areas for AI software are pattern recognition (image and voice) theorem proving and game playing. In recent years a new branch of AI software called artificial neural networks, has evolved. A neural network simulates the structure of brain processes (the functions of the biological neuron) and may ultimately lead to a new class of software that can recognize complex patterns and learn from past experience.

1.13 Software myths:

Software myths are defined as the beliefs about the software and the process used to build it. It is also defined as the misleading altitudes which caused serious problem for managers & technical people. In software industry myths are classified as:

- Management myths.
- Customer myths.
- Practitioner‘s myths.

Management myths:

- Managers with software responsibility are often under pressure ,like maintaining budgets &keeping schedules.
- So they do believe in software myths to lessen their pressure. An example of management myths is given below:

Myths: we already have a book that‘s full standards and procedures for building software. Won‘t that provide my people for everything?

Reality: The reality is the book of standards very well exists, but is it used? Are practitioner‘s aware of its existence?

Customer myths:

A customer who requests computer software may be a person at the next desk or anybody else. The problem about customer myths arises from the misleading of the software managers and practitioners to the customers**.**

An example for customer myths is given below:

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

Reality: An ambiguous statement of objectives is a disaster. Unambiguous requirement are developed through effectives and continuous communication between customer & developer.

Practitioner's myths:

Myths that are still believed by software practitioners.

An example for practitioner's myths is given below:

Myths: Once we write the program and get it to work our job is done.

Reality: Industry date indicates that between 60 and 80% of all effort expended on software will be expended after it is derived to the customer for the first time.

1.14 Planning and Estimation:

Software project planning actually encompasses all estimation, risk analysis, scheduling, and SQA/SCM planning. However, in the context of set of resources, planning involves estimation
- your attempt to determine how much money, how much effort, how many resources, and how much time it will take to build a specific software-based system or product.

1.15 Project Planning Process:

- The objective of software planning is to provide a framework that enables the manager to make reasonable estimation of resources, cost and schedule.
- In addition estimation should attend to define best case &cost scenarios so that project outcome can be bounded.
- The project plan must be adopted and updated as the project proceeds.

1.15.1 Task set for project planning:

- Establish project scope.
- Determine feasibility.
- Analysis risk.

- Define resources:
- Determine human resources.
- Identify environment resources.
- Estimate cost and effort.
- Develop project schedule.

1.16 Software project estimation:

- Software project estimation can be transformed from a black arc to a series of systematic steps that provide estimates with acceptable risk.
- To achieve reliable cost and effort estimate the no of option arise.
- Delay estimation until late in project.
- Base estimates on similar project that have already been completed.
- Use relatively simple decomposition techniques to generate project cost and effort estimates.
- Use one or more empirical models for software cost & effort estimation. **eg: COCOMO MODEL.**
- Unfortunately the first option however alternative is not practical cost estimates must be provide ―up front‖.
- The second option can work reasonably well and if the current project is quite familiar to past.
- Unfortunately past experience as not always be a good indicator of future results.
- Decomposition techniques take a divide and conquer approach to software project estimation by decomposing a project into major function and related software engineering activities.
- Empirical estimation model can be used to complement decomposition technique and offer a potentially valuable estimation approach in their own right production to LOC, FP based estimation.
- Line of code and functional point described as measures from which productivity metrics can be completed.
- LOC, FP estimation are distinct estimation techniques but characteristic is common.
- The project planner begins with a bounded statement of software box. From this statement attempts to decomposed software into problem function that can be estimated individually.
- The base line productivity matrices are then applied to the appropriate estimation

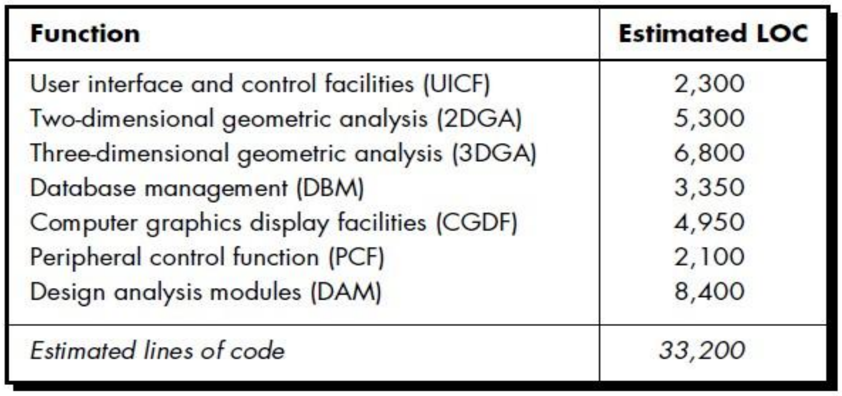
variables and cost or effort of the function is derived.

- Function estimates are combined to produce overall estimates for the entire project.
- When a new project is estimated it should first be allocated to a domain and then appropriate domain average for productivity should be used in generating the estimates.
- LOC, FP estimation techniques differs in the level of detail requires for decomposition &target of the portioning.
- The relevant estimate can be used to derive an FP value that can be tied to pass data and used to generate and estimate.

S=(Sopt+4Sm+Spess)/6
S- Estimation value.
Sopt - optimistic (lower).
Sm -most likely.
Spess - pessimistic (higher).

1.17 LOC based Estimation:

- **Loc** is used as the estimation variables, decomposition is absolutely essential.
- The greater degree of portioning codes to a development of accurate estimates of Loc.
- An example of **Loc-based** estimation.
- Let us consider a software package to be developed for a computer-aided design application for mechanical component.
- The software is to be executed on an engineering workstation and must interface with various peripherals including a mouse, digitizer, high-resolution, color display & laser printer.



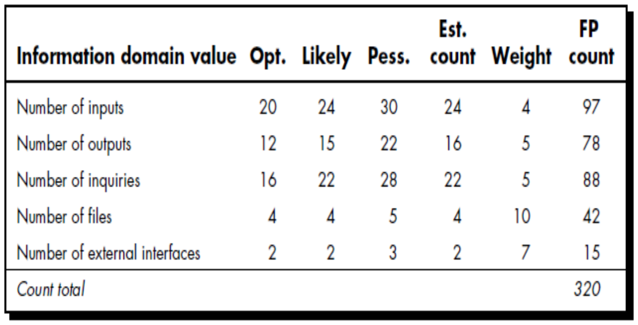
Function	Estimated LOC
User interface and control facilities (UICF)	2,300
Two-dimensional geometric analysis (2DGA)	5,300
Three-dimensional geometric analysis (3DGA)	6,800
Database management (DBM)	3,350
Computer graphics display facilities (CGDF)	4,950
Peripheral control function (PCF)	2,100
Design analysis modules (DAM)	8,400
Estimated lines of code	*33,200*

Table 1.11 LOC Estimation Table

- For example the range of loc estimates for the 3D geometric analysis function is optimistic -4600loc most-likely-6900loc, pessimistic-8600loc.
- Applying equation we can get exact answer.
- Estimation can be done for calculating labor state,projectcost,estimated effort.

1.18 FP BASED ESTIMATION:

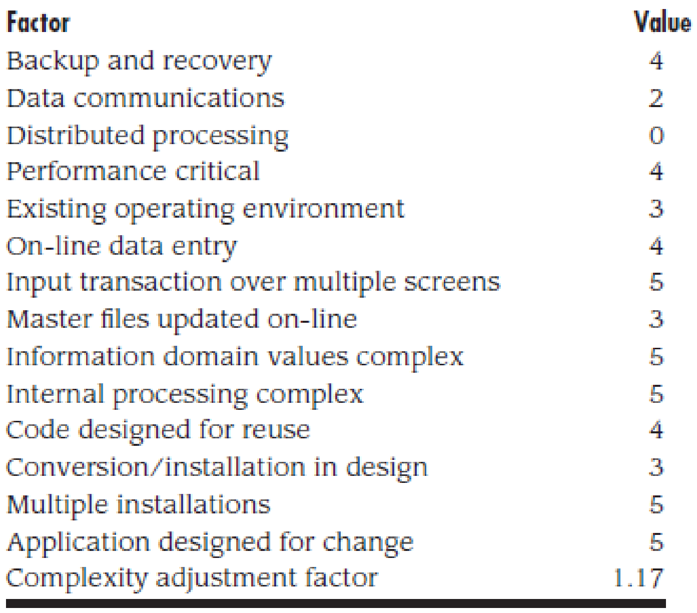
Decomposition for FP-based estimation focuses on information domain values rather than software functions. Referring to the function point calculation the project planner estimates inputs, outputs, inquiries, files, and external interfaces for the CAD software. For the purposes of this estimate, the complexity weighting factor is assumed to be average.



Information domain value	Opt.	Likely	Pess.	Est. count	Weight	FP count
Number of inputs	20	24	30	24	4	97
Number of outputs	12	15	22	16	5	78
Number of inquiries	16	22	28	22	5	88
Number of files	4	4	5	4	10	42
Number of external interfaces	2	2	3	2	7	15
Count total						*320*

Table: 1.12 FP Based Estimation Table

Each of the complexity weighting factors is estimated and the complexity adjustment factor is computed.



Factor	Value
Backup and recovery	4
Data communications	2
Distributed processing	0
Performance critical	4
Existing operating environment	3
On-line data entry	4
Input transaction over multiple screens	5
Master files updated on-line	3
Information domain values complex	5
Internal processing complex	5
Code designed for reuse	4
Conversion/installation in design	3
Multiple installations	5
Application designed for change	5
Complexity adjustment factor	1.17

Table: 1.13 Factor Values

Finally, the estimated number of FP is derived: ***FP*estimated = count-total x [0.65 + 0.01 x (*Fi*)]** ***FP*estimated = 375**

1.19 Object Oriented Estimation

- It is worthwhile to supplement conventional software cost estimation methods with an approach that has been designed explicitly for object oriented software.
- Develop estimates using effort decomposition FP analysis and any other method that is applicable for conventional application.
- Using object-oriented analysis modelling develop use case and determine a count that recognize that the number of use-case may change as the project program
- From 50 analyses, determine the number of key classes.
- Categorize the type of its input for the application develops multiples for support classes.

Interface Type Multiples

No GUI	20
Test-based user i/p	2.25
GUI	2.5
Complex GUI	3.0

- Multiply the number of key classes (step3) by the multiples to obtain an estimate for the number of support classes.
- Multiply the total no. of classes [key support] by the average no. of work units per class.
- Cross check the class based estimate by multiplying the avg.no of work unit/per usecase.
- Develop estimates using decomposition, FP, Loc.
- Use 50 analysis model usecase & determine a count.
- Build analysis model determine no of key classes
- Categorize the type of i/p for application & develop a multiples for support classes.
- Multiply no of key classes by the multiples to obtain estimates for the no.of suppor classes.

K*S=avg.no of work units per class

- Cross check=multiply avg.no.of work unit per use case.