Software Engineering

UNIT - 1

Software → Def :- Software is a Computer programs that When executed provide desired function and performance

Software Engineering →

Def: IEEE → Software Engineering is the application of a Systematic, disciplined and quantifiable approach to the development, operation and maintenance of Software.

Def: Stephen Schach → Software Engineering is a disciplin Whose aim is the production of quality Software that is delivered On time, within budget and that satisfies its requirements.

Def: Fritz Bauer → Software engineering is the establishment and use of Sound engineering principles in order to obtain economical Software that is ~~relia~~ reliable and works efficiently on real Machines.

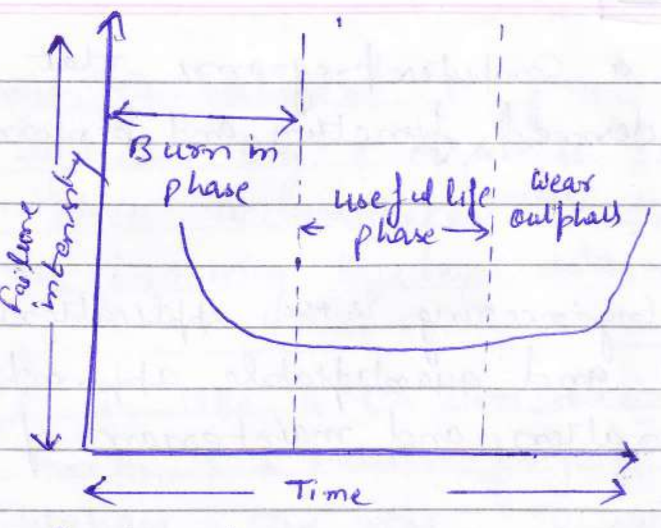
Software Characteristics

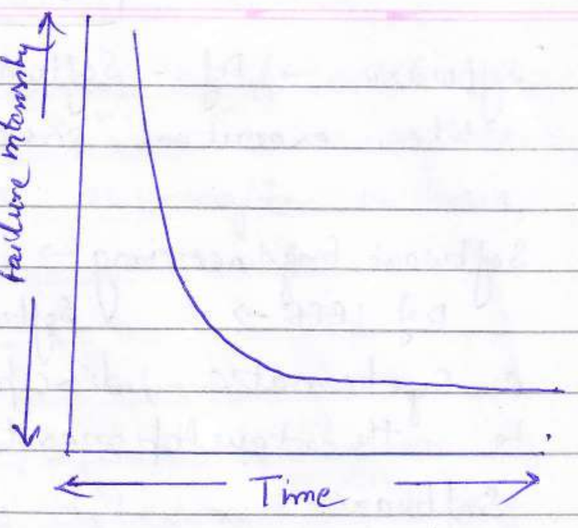
Software has Characteristics that are Considerably different than those of hardware.

1) Software is developed or engineered, It is not manufactured in the classical Sense. In the case of hardware product every product costs us due to raw Material and Other processing expenses. There is no assembly line in Software development. Development of a Software is a One time effort but it requires a Continuous Maintenance effort in order to keep it operational.

2) Software does not wear out : normal product or Manufactured products life depends as Bath tube Curve which have three phases Initial phase,

[illegible]





→ Bath tube Curve
[→ failure curve for manufacture product or Hardware product]

Software Curve

failure Intensity is increased due to wear out phase failure intensity high at initial place or Burn in phase. So Hardware product is operational for short term life cycle. But software does not have this phase, as it does not wear out. Therefore, Software becomes reliable over time instead of wearing out. Hence Software may be retired due to new requirements, new expectations, environmental changes etc.

3) Software is flexible → This statement is not always true for all the situations. Most of us feel that Software is flexible. A program can be developed to do "almost anything". Sometimes, this characteristics 'almost anything' may be the best and may help us to accomodate any kind of change

4) Reusability of Components → every project is a new project and we start from the ~~Seact~~ Sketch and design of every unit of a Software product. Huge effort is required to develop a software which further increases the cost of the Software product. However effort has been made to design standard Components that may be

used in new projects. Software reusability has introduced another area and is known as component based software engineering. e.g- user interfaces are build using reusable components that enable the creation of graphics windows, pulldown menus, and a wide variety of interaction mechanisms.

Software Crisis → Computer industry has progressed at a break-neck speed through the computer revolution, and recently, the network revolution triggered and/or accelerated by the explosive spread of the internet and most recently the web. but the problems with software have not been decreasing. As per IBM report 31% of projects get cancelled before completion, 53% over run their cost estimates etc. History has seen many software failures. Some of there are:

i) Y2K problem — The Y2K problem was the most crucial problem of last century. It was simply the ignorance about the adequacy or otherwise of using only last two digits of the year. The 4-digit date format like 1964, was shortened to 2-digit format, like 64. The developers could not visualize the problem of year 2000. Millions of rupees have been spent to handle this practically non existent problem.

ii) The "star wars" program of USA produced "patriot Missile" and was used first time in Gulf war. patriot Missiles were used as a defence for Iraqi Scud Missiles. The patriot Missiles failed several times to hit Scud Missiles including one that killed 28 US soldiers in Dhahran, Saudi Arabia. A review team was Constituted to find the reason and result was Software bug, A small timing error in the system's clock accumulated to the point that after 14 hours the tracking System was no longer accurate.

3) In 1996 a US Consumer group embarked on an 18-month, $ 1 Million project to replace its customer database. The new System was delivered on time but not works as promised handling routine transactions smoothly but tripping over more Complex ones. Within three weeks the database was shut down / transactions were processed by hand and a new team was brought in to rebuild the System.

possible reasons for such a failure may be that the design team was over optimistic in agreeing to requirements and developers became fixated on deadlines; allowing errors to be Ignored.

4) "One little bug, One big Crash" of ariane - 5 space rocket developed at a cost of $7000 M over a 10 years period. The space rocket was destroyed after 39 seconds of its launch, at an altitude of two and a half Miles alongwith its payload of four expensive and Uninsured scientific satellites.

The reason was Very Simple, When the guidance system's own Computer tried to Convert one piece of data - sideways velocity of the rocket - from a 64-bit format to a 16-bit format; the number was too big, and an overflow error resulted after 36.7 seconds. When the guidance system shutdown it passed Control to an Identical, redundant Unit, Which was there to provide backup in case of Just such a failure. Unfortunately, the Second Unit had failed in the Identical Manner a few Milliseconds before.

Software project planning →

planning may be most important management activity. without a proper plan, no real monitoring or controlling of the project is possible. No amount of technical effort later can compensate for lack of careful planning. Lack of proper planning is a sure ticket to failure for a large software project.

The objective of software project planning is to provide a framework that enables the manager to make reasonable estimates of resources, cost and schedule.

The major issues project planning addresses are:

1) process planning
2) Effort estimation
3) Resource estimation
4) Quality plans
5) Configuration management plans
6) Risk management plans
7) project monitoring plans

1) process planning →

In process planning stage a key activity is to plan and specify the process that should be used in the project. Established process model may be used as a standard process and tailored to suit the needs of the project. In an organization, often standard processes are defined and a project can use any of these standard processes and tailor it to suit the specific needs of the project. Hence the process planning activity mostly entails selecting a standard process and tailoring it for the project. (Tailoring based on the size, complexity and nature of the project, as well as the characteristics of the team, like the experience of the team members with the problem domain as well as the technology being used, the standard process is tailored.

Effort estimation

For a given set of requirements it is desirable to know how much it will cost to develop the software and how much time the development will take. These estimates are needed before development is initiated. primary reason for cost and schedule estimation is cost benefit analysis and project Monitoring and control.

Most of cost estimation procedure focus on estimating effort in terms of person-month (PM). By properly including the overheads in the cost of a person month effort estimates can be converted into cost. (overheads - cost of hardware, software, office spaced)
for software project development, effort and schedule estimates are essential prerequisites for managing the project.

Effort for a project is a function of many parameters, It is generally agreed that the primary factor that controls the effort is the size of the project that is the larger the project, the greater the effort requirement. One common approach therefore for estimating effort is to make it a function of project size and the equation of effort is considered as

$$\boxed{\text{Effort} = a * \text{size}^b}$$

where a and b are constants(s), and project size is generally in KLOC or function points. Values for these constants for a particular process are determined through regression analysis, which is applied to data about the projects that has been performed in the past.

Decomposition techniques take a "divide and conquer" approach to software project estimation. By decomposing a project into major functions and related software engineering activities, cost and effort estimation can be performed in a stepwise fashion. Empirical estimation models can be used to complement decomposition techniques and offer a potentially valuable estimation approach in their own right. A model is based on experience (historical data) and takes the form $d = f(V_i)$

where d is one of a number of estimated values (e.g cost, project duration) and V_i are selected independent parameters (e.g estimated LOC or FP)

Automated estimation tools also implement one or more decomposition techniques or empirical models

Size estimation →

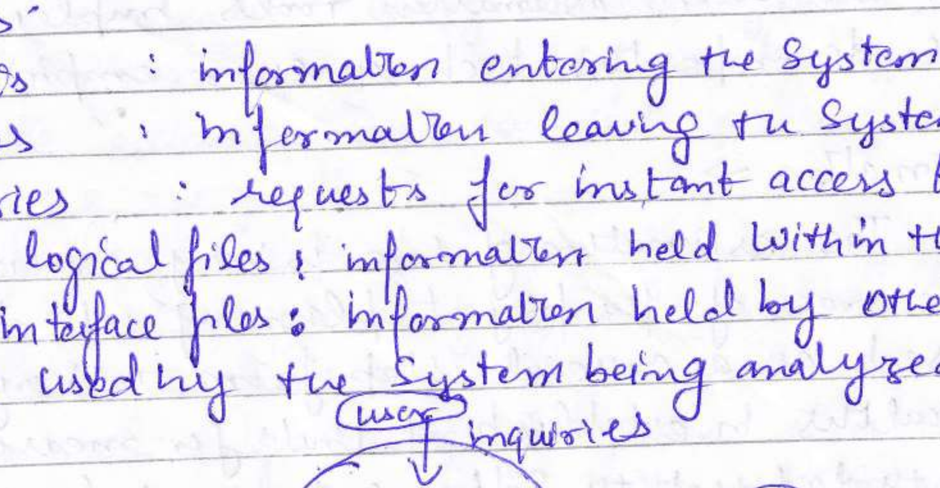
The estimation of size is very critical and difficult area of the project planning. It has been recognised as a crucial step from the very beginning. The difficulties in establishing units for measuring size lie in the fact that the software is essentially abstract. It is difficult to identify the size of the system.

LOC → A line of code is any line of program text that is not a comment or blank line, regardless of the number of statements or fragments of statements on the line. Thus specifically includes all lines containing program header declarations and executable and non executable statements"

Function count →

Measuring software size in terms of lines of code is analogous to measuring a car stereo by number of resistors, capacitors and integrated circuits involved in its production. The number of components is useful in predicting the number of assembly line staff needed, but it does not say anything about of functions available in the finished stereo. Function point measures functionality from the user's point of view, that is, on the basis of what the user requests and receives in return from the system. The principle of albrecht's function point analysis (FPA) is that a system is decomposed into functional units.

Inputs : information entering the system
Outputs : Information leaving the system
Enquiries : requests for instant access to information
Internal logical files : information held within the system
External interface files : information held by other systems that is used by the system being analyzed.



ILF : Internal logical files
EIF : External interfaces

Five functional units are divided into two categories
i) Data function types
Internal logical files (ILF) : A user identifiable group of logically related data or control information referenced by the system, but maintained within another system. This means that EIF counted for one system, may be an ILF in another system.
ii) Transactional function types:-
external input (EI) : An EI processes data or control information that comes from outside the system. The EI is an elementary process, which is the smallest unit of activity that is meaningful to the end user in the business.
External output (EO) : An EO is an elementary process that generates data or control information to be sent outside the system
External inquiry (EQ) : An EQ is an elementary process that is made up of an input - output combination that results in data retrieval.
The five functional units are ranked according to their complexity, low, average, or high, using a set of prescriptive standards. Organisations that use FP methods develop criteria for determining whether a particular entry is low, average or high. after classifying each of the five function types, the unadjusted function points (UFP) are calculated using predefined weights for each function type as given in table. The weighted factors are identified for all functional units and multiplied with the functional units accordingly. The procedure for the calculation of UFP in mathematical form
[illegible] 5 3 [illegible]

Where i indicates the row and j indicates the column w_{ij} = it is the entry of ith row and jth column of the table 4.1

Z_{ij} ; It is the count of the number of functional units of type i that have been classified as having the complexity corresponding to column j

Functional units with weighting factors - I

figure 1

Functional Units	Weighting factors		
	Low	Average	high
External inputs (EI)	3	4	6
External outputs (EO)	4	5	7
External Inquiries (EQ)	3	4	6
Internal logical files (ILF)	7	10	15
External Interface files (EIF)	5	7	10

Functional Units	Count Complexity		Complexity Totals	Functional Unit Totals
External Inputs (EI)	Low ×3	=	□	
	average ×4	=	□	
	high ×6	=	□	□
External outputs (EO)	Low ×4	=	□	
	Average ×5	=	□	
	high ×7	=	□	□
External Inquiries (EQ)	Low ×3	=	□	
	Average ×4	=	□	
	high ×6	=	□	□
Internal logical files (ILF)	Low ×7	=	□	
	Average ×10	=	□	
	high ×15	=	□	□
External Interface Files (EIF)	Low ×5	=	□	
	average ×7	=	□	
	high ×10	=	□	□
Total of function point count Unadjusted			□	□

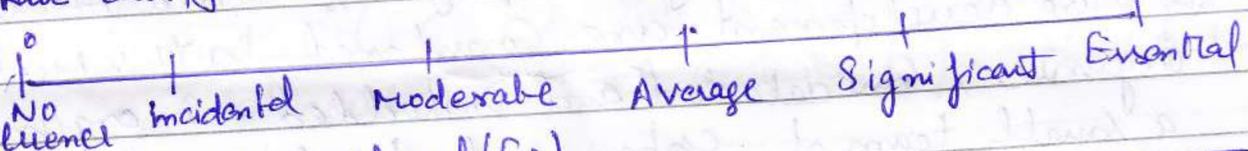
figure 2 UFP calculation table

The final number of function points is arrived at by multiplying the UFP by an adjustment factor that is determined by considering 14 aspects of processing Complexity which are given in figure 3. The adjustment factor allows the UFP Count to be modified by at most ± 35%. The final adjusted FP Count is obtained by using the following relationship

$FP = UFP * CAF$ Where CAF is Complexity adjustment factor and is equal to $[0.65 + 0.01 \times \sum F_i]$ The F_i $(i = 1$ to $14)$ are the degree's of influence and are based on responses to questions noted in figure 3

Computing function points (figure 3)

Rate each factor on a scale of 0 to 5.



No of factors considered (F_i)

1. Does the system require reliable backup and recovery?
2. Is data Communication required?
3. Are there distributed processing functions?
4. ~~Are~~ Are ~~dist~~ performance Critical?
5. Will the system run in an existing heavily utilized operational environment
6. Does the System require on line data entry?
7. Does the on line data entry require the input transaction to be built over multiple screens or operations?
8. Are the master files updated on line?
9. Is the inputs, outputs files or inquiries Complex?
10. Is the internal processing Complex?
11. Is the code designed to be reusable?
12. Are conversion and installation included in the design
13. Is the system designed for multiple installations in different organizations?
14. Is the application designed to facilitate change and ease of

FP's may compute the following important metrics.
productivity = FP/ persons-months
Quality = defects / FP
Cost = Rupees / FP
Documentation = pages of documentation per FP

Constructive Cost Model (COCOMO)
COCOMO is a hierarchy of software cost estimation models, which include basic, intermediate and detailed sub models.
Basic Model - The basic model aims at estimating in the quick and rough fashion, most of the small to medium sized software projects. Three mode of software development are considered in this model; organic, semi detached and embedded. In organic mod. a small team of experienced developers develops software in a very familiar environment. The size of the software development in this model ranges from small (a few KLOC) to medium (a few tens of KLOC), while in other two modes the size ranges from small to very large (a few hundreds of KLOC).
In embedded mode of software development, the project has tight constraints, which might be related to the target processor and it's interface with the associated hardware. The problem to be solved in unique and so it is often hard to find experienced persons, as the same does not usually exist.
The semi detached mode is an intermediate mode between the organic mode and embedded mode.
The comparison of all three modes. Depending on the problem at hand, the team might include a

Mixture of experienced and less experienced people with only a recent history of working together. The basic Cocomo equations take the form

$$E = a_b (KLOC)^{b_b}$$

$$D = C_b (E)^{d_b}$$ where E is effort applied in person-months and D is the development time in months. The coefficients a_b, b_b, C_b and d_b are given table figure (5)

fig 4 Comparison of three COCOMO Modes →

Mode	project size	Nature of project	innovation	Deadline of the project	Development Environment
Organic	Typically 2-50 KLOC	Small size project Experienced developer eg- payroll, inventory projects	Little	Not tight	familiar & In house
Semi detached	typically 50-300 KLOC	Medium size project Medium size team eg utility system database systems editors etc	Medium	Medium	Medium
Embedded	Typically over 300 KLOC	large project real time systems complex interface Very little previous experience eg → ATMs Air traffic Control	Significant	Tight	Complex Hardware/ Customer interface required.

fig-5 Basic COCOMO Coefficients →

project	a_b	b_b	C_b	d_b
Organic	2.4	1.05	2.5	0.38
Semidetached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

when effort and development time are known, the average staff size to complete the project may be calculated as:

Average staff size (SS) = $\frac{E}{D}$ persons

when the project size is known, the productivity level may be calculated as:

productivity (P) = $\frac{KLOC}{E}$ KLOC/PM

② Intermediate Model →

The basic Model allowed for a quick and rough estimate but it resulted in a lack of accuracy. Bohem introduced an additional set of 15 predictors called cost drivers in the intermediate Model to take account of software development environment. Cost drivers are used to adjust the nominal cost of a project to the actual environment, hence increasing the accuracy of the estimate.

The cost drivers are grouped into four categories

1) product attributes
   a) required software reliability (RELY)
   b) Database size (DATA)
   c) product complexity (CPLX)
2) Computer attributes
   a) Execution time constraint (Time)
   b) Main storage constraint (STOR)
   c) Virtual Machine Volatility (VIRT)
   d) Computer turnaround time (TURN)
3) personnel attributes
   a) Analyst capability (ACAP)
   b) Application experience (AEXP)
   c) programmer capability (PCAP)
   d) Virtual Machine experience (VEXP)
   e) programming language experience (LEXP)

a) Modern programming practices (MODP)
b) Use of Software tools (TOOL)
c) Required development Schedule (SCED)
each Cost driver is rated for a given project environment
The rating uses a scale Very low, low, nominal high
Very high, extra high Which describes to what extent
the Cost driver applies to the project being estimated.

(Multiplier Values for effort calculations - fig)

Cost drivers	RATINGS					
	Very Low	Low	Nominal	High	Very high	Extra high
product Attributes						
RELY	0.75	0.88	1.00	1.15	1.40	—
DATA	—	0.94	1.00	1.08	1.16	—
CPLX	0.70	0.85	1.00	1.15	1.30	1.65
Computer Attributes						
Time TIME	—	—	1.00	1.11	1.30	1.66
STOR	—	—	1.00	1.06	1.21	1.56
VIRT	—	0.87	1.00	1.15	1.30	—
TURN	—	0.87	1.00	1.07	1.15	—
personnel Attributes						
ACAP	1.46	1.19	1.00	0.86	0.71	—
AEXP	1.29	1.13	1.00	0.91	0.82	—
PCAP	1.42	1.17	1.00	0.86	0.70	—
VEXP	1.21	1.10	1.00	0.90	—	—
LEXP	1.41	1.07	1.00	0.95	—	—
project Attributes						
MODP	1.24	1.10	1.00	0.91	0.82	—
TOOL	1.25	1.10	1.00	0.91	0.83	—
SCED	1.23	1.08	1.00	1.04	1.10	—

→ Coefficients for intermediate COCOMO

Project	a_i	b_i	c_i	d_i
Organic	3.2	1.05	2.5	0.38
Semidetached	3.0	1.12	2.5	0.35
Embedded	2.8	1.20	2.5	0.32

The multiplying factors for all 15 cost drivers are multiplied to get the effort adjustment factor (EAF). Typical values for EAF range from 0.9 to 1.4
The intermediate COCOMO equations take the form:

$$E = a_i (KLOC)^{b_i}$$

$$D = c_i (E)^{d_i}$$

The coefficients a_i, b_i, c_i and d_i are given in fig

Detailed COCOMO model

A large amount of work has been done by Bohem to capture all significant aspect of a software development. It offers a means for processing all the project characteristics to construct a software estimate. detailed model introduces two more capabilities:

1) Phase sensitive effort multipliers:
Some phases (design, programming, integration/test) are more affected than others by factors defined by the cost drivers. The detailed model provides a set of phase sensitive effort multipliers for each cost drivers. The helps in determining the manpower allocation for each phase of the project.

2) three level product hierarchy :-
Three product levels are defined. These are module, subsystem and system levels. The rating's of the cost drivers are done at appropriate level; that is the level at which it is most susceptible to variation.

3 Quality plan → Quality plans are important for ensuring that the final product is of high Quality. The project Quality plan identifies all the V & V activities that have to be performed at different stages in the development and how they are to be performed. Much of the Quality plan revolves around testing and reviews. effectiveness of reviews depends on how they are conducted. One particular process of conducting reviews called inspections.

4) Risk management → A Software project is a complex undertaking. Unforseen events may have an adverse impact on a projects cost, schedule, or quality. Risk management is an attempt to minimize the chances of failure caused by unplanned events. The aim of Risk management is not to avoid getting into projects that have risks but to minimize the impact of risks in the projects that are Undertaken. Risk is a probabilistic event - it may or may not occur, but if they do occur, they have a negative impact on the project. To meet project goals even Under the presence of Risks requires proper risk management. Risk management requires that risks be identified, analyzed and prioritized. Then risk mitigation plans are made and performed to minimize the effect of highest priority Risks.

5) project Monitoring plan →
for the plan to be successfully implemented it is essential that the project be monitored carefully. Activity level monitoring, status reports and Milestone analysis are the mechanisms that are often used. for analysis and reports, the actual effort, schedule, defects and size should be measured. With these measurements