COLLATE

Software Engineering Notes UNIT - 2

SOURCE : NOTESHUB

REQUIREMENT DOCUMENTATION:-

This is done to represent the requirements in a consistent format. Requirement documentation is called SRS [SOFTWARE REQUIREMENT SPECIFICATION].

It is a legal document and contains a set of predefined points. It acts as a contract between the user and the developer.

- CHARACTERISTICS OF GOOD SRS:-

* CORRECT :- If every requirement stated is fulfilled.
* UNAMBIGUOUS:- If every requirement stated has only one interpretation.
* COMPLETE:- All the predefined points are mentioned in the document
* CONSISTENT:- If it has no conflict.
  Ex: One requirement may state that A must always follow B while another requires that "A" and "B" occur simultaneously.
* VERIFIABLE:-
  Ex:- The output of the program shall be produced within 20 sec. of event. This statement can be verified because it uses a measurable quantity.

MODIFIABLE :- The structure of SRS is retained while the requirements can be changed easily.

- TRACEABLE :- If origin of each requirement is clear and it facilitates the referencing of each requirement.

* ORGANIZATION OF SRS :- [IEEE-std. 830-1993]

1. INTRODUCTION — gives basic layout & terms to be used in our software system
   - 1.1 Purpose
   - 1.2 Scope
   - 1.3 Definitions, Acronyms
   - 1.4 References
2. OVERALL DESCRIPTION
   - 2.1. Product Perspectives
     - 2.1.1 System Interfaces
     - 2.2.2 H/w Interfaces - what all devices are to be attached
     - 2.2.3 S/w Interfaces - version no., source, name, specification no.
   - 2.2 Product Functions
   - 2.3 Constraints - Any conditions in which our s/w won't respond
   - 2.4 Assumptions and Dependencies - things we have assumed and can affect the requirements of SRS
3. SPECIFIC REQUIREMENTS → which are valid to our particular s/w system.
   - 3.1 Performance Requirements
   - 3.2 Logical Database Requirements
   - 3.3 Design Constraints
   - 3.4 Software System Attributes
     - 3.4.1 Reliability
     - 3.4.2 Availability
     - 3.4.3 Security
     - 3.4.4 Maintainability
4. Change MANAGEMENT PROCESS — Identifies the changes, evaluates & updates the SRS.
5. DOCUMENT APPROVALS — Approvers' name, sign. & date should be done.
6. SUPPORTING INFORMATION → Table of content, Index

FEASIBILITY STUDY - Study made before committing to a project.

- technical
- economical
- personal

RISK MANAGEMENT-

Risk- Problem that could cause some loss or threaten the success of the project but which has not happened yet

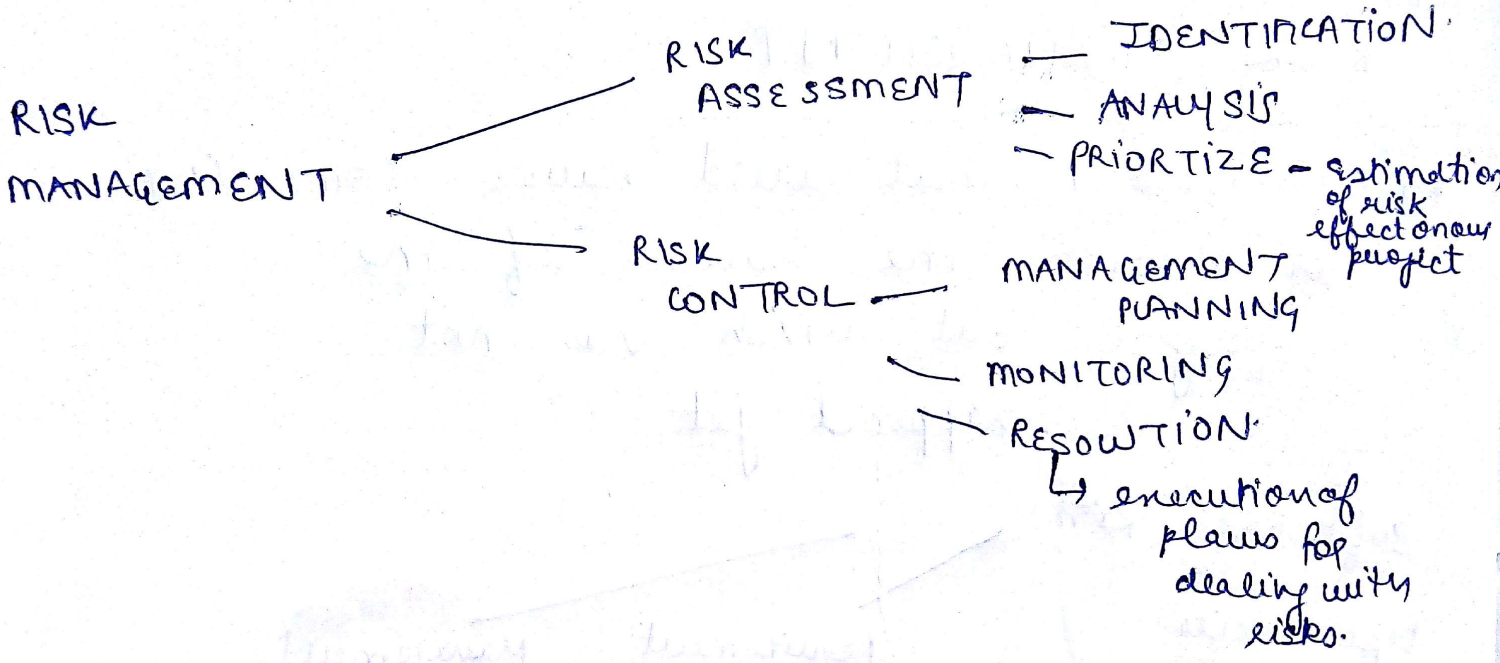
Software Risks

Dependencies
- Availability of trained, experienced people
- due to being dependent on outside agencies

Requirement Issue
- lack of clear product vision
- Rapidly changing requirements

Management Issue
- Inadequate Planning.
- Poor coordination among team staff

Lack of knowledge
- Poor understanding of methods, tools
- Inadequate training.



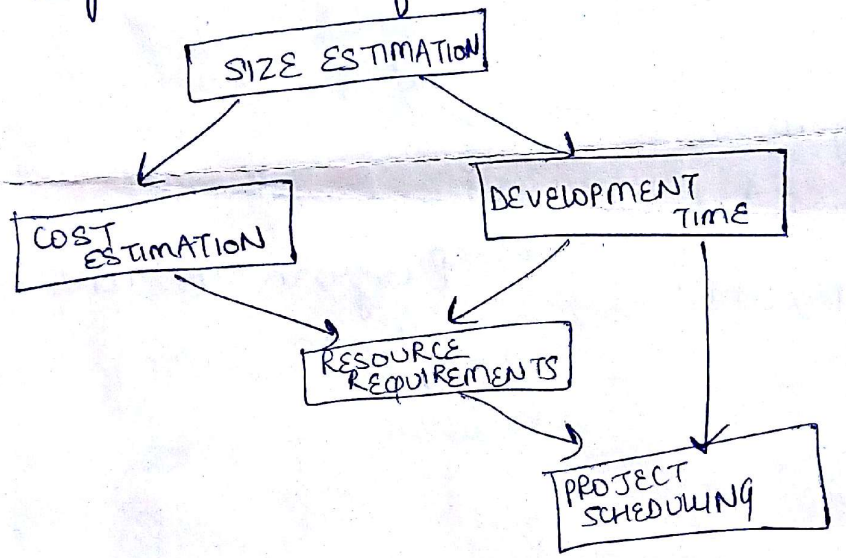
CH- SOFTWARE PROJECT PLANNING

After the SRS is prepared, we would like to estimate size, cost and development time of the project.

Software managers are responsible for planning and scheduling project development.

to ensure that work is done to meet the required standards.

So, Project planning is done to estimate the cost, risk, size and time to be taken to make the software project successful.



ACTIVITIES DURING S/W PROJECT PLANNING

- Here size is estimated, based on it cost & time depends. The cost estimation and development time are further used to know the resources to be required.

and last stage is Project scheduling so as to monitor the progress of project.

SIZE ESTIMATION :-

To establish a unit of measure for size, we have :-

(i) LINES OF CODE :-

A line of code is any line of program text that is not a comment or blank lines.
It includes all lines containing program header, declarations, and executable and non-executable statements.

```
Ex :- 1.  int  Sort (int x[], int n)
      2.  {
      3.    int i, j, save;
      4.    /* This function sorts array */
      5.    if (n<2) return 1;
      6.    for (i=2; i<=n; i++)
      7.    {
      8.      min = i-1;
      9.      if (x[i] < x[j])
      10.     {
      11.       save = x[i];
      12.     }
      13.   return 0;
      14. }
```

Program contains

LOC = 13

```
Ex-1. if (n<2)
      2.   Save=1;
      3.   else
      4.   save=2;
           LOC=4
```

```
if(n<2) ? save=1 : save=2;
LOC=1
```

DISADVANTAGES :-

* Is language dependent
* Not useful for large programs.

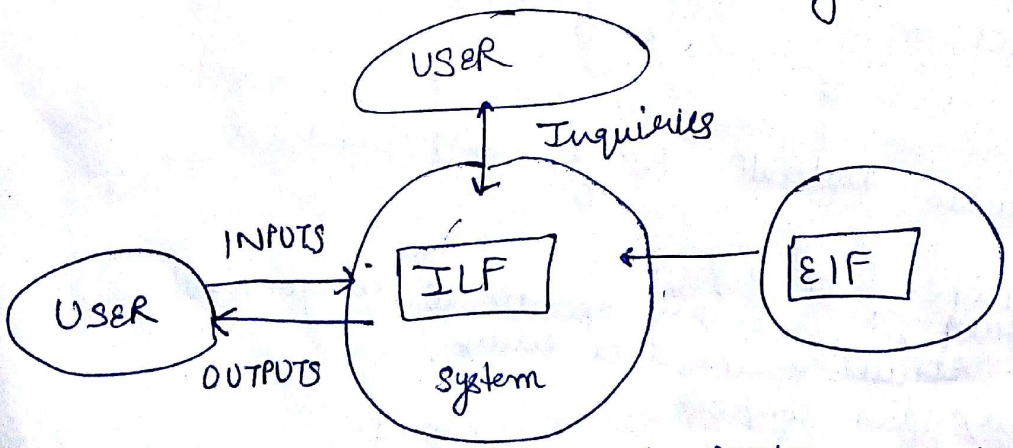
FUNCTION COUNT:-

To overcome the disadvantage of LOC, we have a technique called as FPA (FUNCTION POINT ANALYSIS) was developed. It measures functionality from the user's point of view. So, size is measured in terms of functionality being delivered and not the lines of code, source modules, files etc.

ADVANTAGES:-

- Not language dependent
- based on system users external view of system.
- FPA is a system decomposed into functional units :- DATA FUNCTION (ILF, EIF) → TRANSACTION FUNCTION

* INPUTS : information entered into the system
* OUTPUTS: information leaving the system
* ENQUIRIES: request for instant access to information (data retrieval)
* INTERNAL LOGICAL FILES: info held within the system.
* EXTERNAL INTERFACE FILES :- info held by other systems that is used by present system being analyzed.



FUNCTION POINT ANALYSIS

COUNTING FUNCTION POINTS:

The five functional units are ranked according to their complexity i.e low, AVG., HIGH.

FUNCTIONAL UNITS	WEIGHTING FACTORS		
	LOW	AVERAGE	HIGH
EI	3	4	6
EO	3	5	7
EQ	3	4	6
ILF	7	10	15
EIF	5	7	10

(1) UNADJUSTED FUNCTION POINT COUNT (UFP) = $\sum_{i=1}^{5} \sum_{j=1}^{3} Z_{ij} \cdot W_{ij}$

To compute complexity

(2) FP (FUNCTION POINT) = UFP * CAF

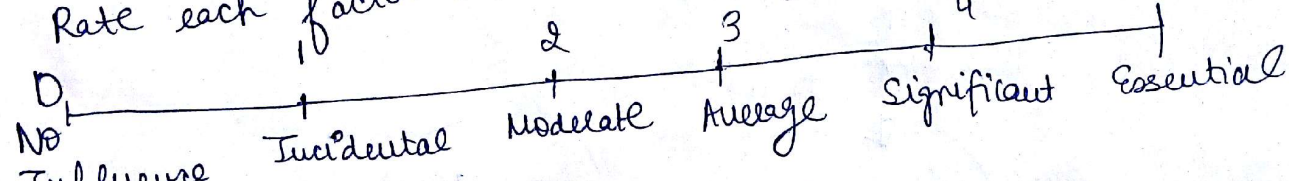
(CAF: Cumulative Adjustment factor)

To analyze the system.

$= 0.65 + 0.01 * \sum F_i \quad (i = 1 \text{ to } 14)$

No. of factors = F_i $(i = 1 \text{ to } 14)$

Rate each factor on a scale of 0 to 5



1. Does system require backup?
2. Will system run in heavily operational environment
3. Does system require online data entry?
4. Are input, output, files complex?
5. Can design code be reusable? - - - upto 14.

Q: Consider a project with foll. functional units:

Number of user inputs = 50
" " user outputs = 40
" " " enquiries = 35
" " user files = 06
" " external interfaces = 04

Assume all complexity adjustment factors and weighting factors are average.

$$UFP = \sum_{i=1}^{5} \sum_{j=1}^{3} z_{ij} \cdot w_{ij}$$

$$= 50 \times 4 + 40 \times 5 + 35 \times 4 + 6 \times 10 + 4 \times 7$$

$$= 200 + 200 + 140 + 60 + 28 = 628$$

$$CAF = 0.65 + 0.01 \times \sum F_i$$

$$= 0.65 + 0.01 \times 14 \times 3 = 0.65 + 0.42 = 1.07$$

$$FP = UFP * CAF$$

$$= 628 \times 1.07 = 672$$

Q. An application has foll:-
10 low external inputs, 12 high external outputs, 20 low internal logical files, 15 high external interface, 12 average external inquiries & a value of complexity adjustment factor of 1.10.
What are unadjusted & adjusted function point counts?

INFORMATION FLOW METRICS :-

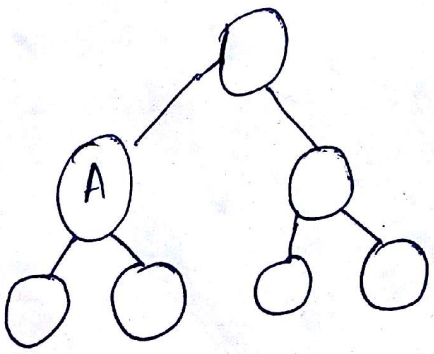
- **Component :-** Any element identified by decomposing a system into its constituent parts.
- **Cohesion :-** The degree to / measure of how well the lines reside a source code belong / work together in a module
- **Coupling :-** The degree of linkage between one module to another.

So high cohesion & low coupling

→ BASIC INFORMATION FLOW MODEL :-

FAN IN = Count of the number of components that ~~are called~~ can call to component A

Ex- main() - function
other modules are dependent on it



FAN OUT = Count of number of component that are called by component A

main() function variables are used by other modules

$$IF = [FAN(IN)(A) \times FAN(OUT)]^2$$

processed in and output from software is called Data structure metrics.

Ex:-

Program	Data input	Internal data	Data output
PAYROLL	Name / Employee Id / Pay Rate / Number of hours worked	Overtime details Insurance of employees	Gross / Net Pay

- **AMOUNT OF DATA:-**

To determine the amount of data, there is a need to maintain a CROSS-REFERENCE LIST. It indicates the line where a certain variable is declared and the line where it is referenced. It is useful in debugging and maintenance.

Ex:-

```
1. #include <stdio.h>
2. struct check
3. {
4. float gross, tax, net;
5. } pay;
6. float hours, rate;
7. void main()
8. {
9. while(!feof(stdin))
10. {
11. scanf("%f %f", &hours, &rate);
12. pay.gross = hours * rate;
13. pay.tax = 0.25 * pay.gross;
14. pay.net = pay.gross - pay.tax;
15. printf("%f %f %f\n", pay.gross, pay.tax, pay.net, pay.hours);
```

16. 3
17.3

variables							lines in which they occur
check	2			13	14	15	
gross	4		12				
hours	6		11	12			
net	4		14	15			
tax	4		13	14	15		
pay	5	12	13		13	14	In line 15, pay is written three times so mention it three times
	14	14	15		15	15	
rate	6	11	12				

CROSS- REFERENCE LIST.

- USAGE OF DATA WITHIN A MODULE:-

LIVE VARIABLE:-

A variable is live from the beginning of a procedure to the end of procedure.
It is live at a particular statement only if it is referenced a certain number of statements before or after that statement.

LINE NO.	LIVE VARIABLES	COUNT (5)
2.	-	0
3.	-	0
4	-	0
5	-	1
6	-	1
7	-	1
8.	-	1
9.	-	
10.	- hour, rate	2
11.		4
12.	hour, rate, pay, gross	4
13.	pay, tax, gross, hours	5
14.	pay, net, gross, tax, hours	5
15.	pay, gross, tax, net, hours	20

(1) $\overline{LV} = \frac{20}{14}$

(2) $\gamma = \frac{20}{7}$

(3) $\omega_m = \frac{20}{14} \times \frac{20}{7}$

Q. Draw the cross-reference list and find the average number of line variable and module weakness & average life of live variable.

```
1. # include < stdio.h>
2.
3. void swap (int x[ ], int k)
4. {
5.  int t;
6.  t= x[k];
7.  x[k]= x[k+1];
8.  x[k+1]= t;
9.  }
10.
11.  void main ( )
12. {
13. int i, j, last, size, continue, a[100], b[100];
14. scanf ("%d", & size);
    for (j=1; j<= size ; j++)
    scanf ("%d %d", &a[j], &b[j];
     last = size;
      continue =1;
     while (continue)
    {
     continue = 0;
      last = last-1;
       i=1;
    while (i<= last)
    { if (a[i]> a[i+1])
    { continue =1;
      swap (a, i);
       swap (b, i);
    }
     i= i+1;
  }
```

```
for (j=1; j<= size ; j++)
printf ("%d %d\n", a[j], t[j]);
}
```

LINE	LIVE VARIABLES	COUNT
4. 5. 6.	t, n, k	3
7.	t, n, k	3
8.	t, n, k	3
9. 10. 11. 12. 13. 14.	size	1
15.	size, j	2
16.	size, a, b, j	4
17.	size, last	2
18.	continue	1

(upto line 18)

Total = 19

$$\overline{LV} = \frac{\text{Sum of count of live variables}}{\text{Count of executable statements}}$$

$$= \frac{19}{15}$$

$$=$$

$$\gamma = \frac{\text{Sum of count of live variables}}{\text{Total no of variables}} = \frac{19}{9}$$

$$WM = \overline{LV} \times \gamma$$

$$= \frac{19}{15} \times \frac{19}{9}$$

$$=$$

NOTES:-
Ignore # directives, comments & function declarations statements

t	5	6	8				
n	6	7	7	8			
k	6	7	7	8			
size	13	14	15	17			
i	13						
j	13	15	15	15	16	16	
a	13	16					
b	13	16					
last	13	17					
continue	13	19					

- CROSS-REFERENCE LIST

COST ESTIMATION :-

- STATIC SINGLE VARIABLE MODELS
- STATIC MULTIVARIABLE MODELS
- COCOMO-I
- COCOMO-II

* STATIC SINGLE VARIABLE MODELS :- → valid for one organization

This method is used to estimate cost, time, effort etc. They all depend on same variable / predictors. Here a unique variable (say size) is taken for calculating all others (say cost, time)

$C = aL^b$

a, b - constants

Cost → (Person-month)

Size → (number of lines of code)

* STATIC MULTIVARIABLE MODELS :- → valid for all organization, is a standard model since we have inclusion of a large number of cost participations, memory constraints etc

These models depend on several variables and All variables are interdependent. Here several variables are needed to describe the software process and further an estimate of time and cost is provided.

Ex- SEL (Software Engineering Laboratory) model

$E = 1.4L^{0.93}$ → Effort (Person-months)

$DOC = 30.4L^{0.90}$ → (Documentation - no. of pages)

$D = 4.6L^{0.26}$ → Duration in months

Ex:-

WALSTON AND FELIX AT IBM.

- $E = 5.2L^{0.91}$

$D = 4.1L^{0.36}$

Productivity Index $I = \sum_{i=1}^{29} w_i x_i$ → weight factor

29 variables

$x_i = -1, 0, +1$ → variable increases productivity; 0 → has no effect; -1 → variable decreases productivity

Q. Compare the Walston-Felix Model and SEL model equation on a software development expected to involve 8 person years of effort

a) Find no. of LOC

Effort = 8PY = 96 person-months

$\frac{E_1}{E_2} = \frac{1.4 L_1^{b_1}}{5.2 L_2^{b_2}}$ $L = \left(\frac{E}{a}\right)^{1/b}$

$\Rightarrow E = 1.4 L^{0.93}$

$\Rightarrow \frac{96}{1.4} = L^{0.93}$

$\Rightarrow L = \left(\frac{96}{1.4}\right)^{\frac{1}{0.93}} = 94264$ LOC

L(SEL)

$L(W\text{-}F) = \left(\frac{96}{5.2}\right)^{1/0.91} = 24632$

b) Duration in months-

$D(SEL) = 4.6^{0.26L} = 4.6(94264)^{0.26}$

$= 15$ months

$D(W\text{-}F) = 13$ months

c) Productivity - no. of lines of code per person/month

μ LOC/PY

$P(SEL) = \frac{94264}{8} = 11783$ LOC/Person-years

$P(W\text{-}F) = 24632/8 = 3079$ LOC/Person-years

d) Average manning = Average number of Persons req./month

$M(SEL) = 96\,PM/15M = 6.4$ Persons

$M(W\text{-}F) = 96/13 = 7.4$ Persons

CONSTRUCTIVE COST MODEL (COCOMO)

Is a hierarchy of software cost estimation models which include

BASIC — INTERMEDIATE — DETAILED

(1) BASIC MODEL :-

- Aims at estimating in a quick and rough fashion, for small to medium sized projects.

	ORGANIC	SEMI-DETACHED	EMBEDDED
PROJECT SIZE	2-50 KLOC	50-300 KLOC	>300 KLOC
	Small size projects	medium size project	Real time systems
	Ex:- Pay roll projects	Ex:- Database systems	Ex:- ATMS
	Experienced developers	Any experience on similar projects	very little previous experience
Deadline:-	Not tight	~~Little~~ Medium	Tight
Innovation:-	Little	medium	Significant
Development environment	Familiar	Medium	Complex interfaces are required

The basic COCOMO equations take the form:-

$$\begin{cases} E = a(KLOC)^b \\ D = c(E)^d \end{cases}$$

E - Effort - Person-Months

D - Development time - months

Average Staff Size $= \frac{E}{D}$ Persons

Productivity $= \frac{KLOC}{E}$ KLOC/PM

PROJECT	a_b	b_b	c_b	d_b
ORGANIC	2.4	1.05	2.5	0.38
SEMI DETACHED	3.0	1.12	2.5	0.35
EMBEDDED	3.6	1.20	2.5	0.32

Q. Suppose that a project is estimated at 400 KLOC. Calculate the effort and development for each of the three modes.

ORGANIC MODE :- $E = 2.4(400)^{1.05} = 1295.31$ PM

$D = 2.5(1295.31)^{0.38} = 38.07$ M

SEMI-DETACHED :- $E = 3.0(400)^{1.12} = 2462.79$ PM

$D = 2.5(2462.79)^{0.35} = 38.45$ M

EMBEDDED :- $E = 3.6(400)^{1.20} = 4772.81$ PM

$D = 2.5(4772.8)^{0.32} = 38$ M

Q. A project size of 200 KLOC is to be developed. S/w development team has average experience on similar type of projects, project schedule is not tight.

→ SEMI DETACHED MODE :-

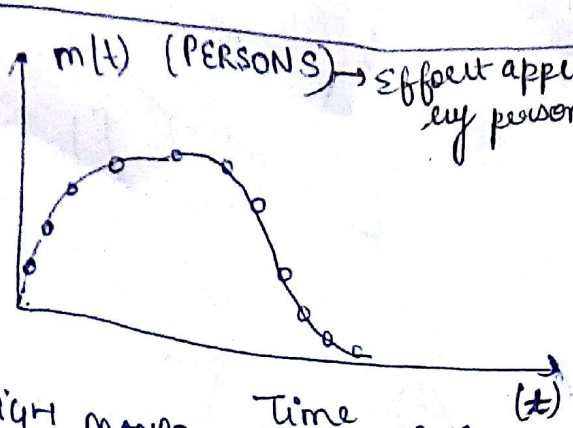
COCOMO - II

- It is the revised version of original COCOMO model.
- This model provides a set of techniques for evaluating the effects of software technology improvements on software life cycle costs and schedules.

Here the projects are categorized into three types :-

* END USER PROGRAMMING :- Applicable to small system.
  Ex- Spreadsheets, query systems.
  Tools - MS-Excel, MS-Access
* ~~INFRASTRUCTURE~~ SECTOR :- It includes s/w that provides infrastructure like operating system, database management system. Ex- Oracle, Linux
* INTERMEDIATE SECTOR :- These applications are complex and require specialised developers with good knowledge of development and engineering practise
  Ex:- 4GL programming like Power Build

PUTNAM RESOURCE ALLOCATION MODEL :-



RAYLEIGH MANPOWER Time CURVE

Two scientists NORDEN PUTNAM who observed that a Rayleight curve can be used as an approximate model for a range of software development projects.

Rayleight curve - drawn b/w manpower involved and the time taken by them.

$$m(t) = \frac{dy}{dt} = 2Kat e^{-at^2}$$ → Rayleigh curve → (1) is given by differential eqn.

STANDARD EQN.

- $m(t)$ → manpower utilization rate per unit time (i.e. effort put in by persons)
- K → area under curve
- t → time taken
- a → parameter that affects shape of curve

Integrating → (1), we get cumulative manpower cost

$$y(t) = K\left[1 - e^{-at^2}\right] \rightarrow (2)$$

$y(0) = 0$

$y(\infty) = K$

Derivative taken on (1) → Acc. to Product Rule

$$\frac{d^2y}{dt^2} = 2Ka e^{-at^2}\left[1 - 2at^2\right] = 2Ka\left[e^{-at^2} + t\left(e^{-at^2}\right)(-2at)\right]$$

and evaluate to zero

On evaluating diff. eqn. derivate to zero, we get maxima

$$\Rightarrow t_d^2 = \frac{1}{2a}$$ → t_d - time when maximum effort rate occurs

peak manning time / peak time

$$\Rightarrow a = \frac{1}{t_d^2}$$

Now, substitute t_d in place of t in (2), we get

$$\varepsilon = y(t) = K\left[1 - e^{-\frac{t_d^2}{2t_d^2}}\right] = K(1 - e^{-0.5})$$

$$\left[1 - e^{-a t_d^2}\right] \rightarrow \left[1 - e^{-\frac{a}{2a}}\right]$$

$$\Rightarrow \boxed{\varepsilon = y(t) = 0.3935K}$$

Now putting t_d in place of t in (1), we get

$$m(t) = 2kate^{-at^2} = \frac{2K t_d}{2t_d^2} e^{-\frac{1}{2}}$$

$$\boxed{m_0 = \frac{K}{t_d\sqrt{e}}}$$

→ PEAK MANNING since we have used peak manning time, t_d

K → total project ~~cost~~ effort in person years

t_d = delivery time

m_0 - no. of persons employed at peak

Q. A software project is planned to cost 95PY in a period of 1 year and 9 months. Calculate peak manning & average rate of s/w team build up

$K = 95PY$

$t_d = 1.75$ years

Peak manning, $m_0 = \frac{K}{t_d\sqrt{e}} = \frac{95}{1.75 \times 1.648} = 32.9 \simeq 33$ persons

Q. Average rate of software team build up:-

$= m_0 / t_d = 33/1.75 = 18.8$ person/year

Q. Consider a large scale project for which the manpower requirement is $K = 600$ PY and development time is 3 years 6 months $= 3.5$ years

a) Calculate peak manning & peak time.

$t_d = 3$ years & 6 months $= 3.5$ years

$m_0 = \frac{K}{t_d \sqrt{e}} = \frac{600}{3.5 \times 1.648} \cong 104$ persons

b) $y(t) = K[1 - e^{-at^2}]$

$y(1.7) = 600[1 - e^{-0.041(1.7)^2}]$

$= 32.6$ PY

$t = 1.17$ years

$a = \frac{1}{2t_d^2} = \frac{1}{2 \times (3.5)^2} = 0.041$

DIFFICULTY METRIC:-

$m'(t) = \frac{d^2y}{dt^2} = 2Kae^{-at^2}(1 - 2at^2)$

for $t = 0$

$\Rightarrow m'(0) = 2Ka = \frac{2K}{2t_d^2} = \frac{K}{t_d^2}$

DIFFICULTY measured in person/year

→ It tells that a project is more difficult to develop when manpower demand is high & when time schedule is short.

CH- SOFTWARE METRICS

The continous application of measurement based techniques to the software development process and products to its supply of meaningful and timelly management information, together with the use of those techniques to improve the process and its products.

Metrics is all about measurement.
It includes estimation of software system size, cost to develop a system and duration of the development of the project, prediction of quality levels.

CATEGORIES OF METRICS :-

(i) PRODUCT METRICS :- to describe the characterics of product such as size, complexity, performance.

(ii) PROCESS METRICS :-
to describe the efficiency and quality of processes.
Ex- effort, time, number of defects

(iii) PROJECT METRICS :- to describe the project characteristics and execution.
Ex- cost, schedule, number of software developers

SIZE METRICS :- LOC (LINES OF CODE), FUNCTION COUNT

TOKEN COUNT :-

A program is a series of tokens.
Developed by Professor Halstead.

Tokens

Operator

- Ex:- Symbol/keyword that specifies an algorithmic action.
- Punctuation marks
- Arithmetic symbols such as +, -, /, * and command names like "while", "for", "printf"
- Special symbols such as =, braces, parentheses, function calls/ names like "eof" brackets, commas, terminators

NOTE:-
- GOTO- is operator
- ~~array name & index~~ [ ]
- ~~Struct name, member name~~ →

Operands

- Symbol used to represent data
- variables, constants, labels.
- array name & index
- → struct name, member name

NOTE:-
- All hash directives are ignored
- Comments are not considered
- Identifier and function declarations are not considered

a) $\eta = \eta_1 + \eta_2$ → number of unique operands

SIZE ↓ VOCABULARY OF PROGRAM

η_1 → number of unique operators

b) $N = N_1 + N_2$ → Total occurrences of operands

N → Program length

N_1 → Total occurrences of operators

$= 2 * LOC$

c) $V = N * \log_2 \eta$ UNIT- bits

V → Volume

d) $E = V/L = D * V$ UNIT- Elementary mental discriminations

V → Volume

e) $L = V^*/V$

L → Program level

V^* → Potential volume

Range

$L = (0-1)$

f) $D = 1/L$

Difficulty

g) $T = E/\beta$ $(\beta = 18)$ UNIT- Seconds

T → Time

β → Stroud number (5-20)

h) Language level

$\lambda = L^2 V$

Q. 1. int sort (int x[ ], int n)
2. {
3. int i, j, save, iml;
4. /* This function sorts array */
5. if (n<2) return 1;
6. for (i=2; i<=n; i++)
7. {
8. iml = i-1;
9. for (j=1; j<= iml; j++)
10. if (x[i] < x[j])
11. {
12. save = x[i];
13. x[i] = x[j];
14. x[j] = save;
15. }
16. }
17. return 0;
18. }

OPERATOR	OCCURENCES	OPERANDS	OCCURRENCES
int	4	sort	1
( )	5	x	7
,	4	n	3
[ ]	7	i	8
{ }	3	j	7
if	2	save	3
<	2	iml	3
return	2	2	2
;	11	1	3
for	2	0	1
=	6		

++	2		
-	1		
<=	2		
$n_1 = 14$ (1 unique operator)	$N_1 = 53$ (Total occurrences)	$n_2 = 10$	$N_2 = 38$

Find η, N, V, E, λ

$\eta = n_1 + n_2 = 14 + 10 = 24$

$N = N_1 + N_2 = 53 + 38 = 91$

$V = N * \log_2 \eta$

$= 417$ bits

$L = V'/V$ (given)

$= 11.6 / 417$

$= 0.027$

$E = V/L$

$= 417 / 0.027$

$= 10973.63$

$\simeq 10974$ elementary mental discriminations

$\lambda = L^2 V$

$\lambda = (0.027)^2 \times 417$