UNIT – 4
Software Metrics

Software Metrics

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

- Software metrics is all about measurements which in turn, involve numbers, the use of numbers to make things better, to improve the process of developing software and to improve all aspects of management of that process.
- It covers the techniques for monitoring and controlling the process of software development, such that the fact it is going to be late for six months can be recognized as early as possible rather than the day before delivery is due.

Area of Applications

- Most established area of software metrics is Cost and Size estimation techniques.
- Controlling software development projects through measurement is an area of great deal of interest.
- The prediction of quality level for software, often in terms of reliability, is another area where Software metrics have an important role to play.
- Software metrics is also used to provide management information. This includes information about productivity, quality and process effectiveness.

Problem during Implementation

- Implementing Software metrics in an organization of any size is difficult.
- Another problem during implementation arises if we start measuring the performance of the individuals. “this may appear attractive to some managers”.
- Software metrics cant solve all the problems, and they can enable managers to improve their processes, to improve productivity and quality.

Categories of Metrics

- Product Metrics:- Describes the characteristics of the product such as size, complexity, design features, performance, efficiency, reliability, portability etc.
- Process Metrics :- describe the effectiveness and quality of the processes that produce the software product. Example :-

  - effort required in the process
  - time to produce the product
  - effectiveness of defect removal during development.
  - number of defects found during testing
- Project Metrics:- describe the project chracteristics and execution. Example
  - number of software developers
  - staffing pattern over the life cycle
  - cost and schedule
  - productivity.

Information Flow (IF) Metrics

- IF metrics model the degree of cohesion and coupling for the particular system component.
- Cohesion :- if the component has to do numerous tasks, it is said to have lack of “cohesion”.
- Coupling :- if it passes to and/or accepts information from many other components with in the system, it is said to highly coupled.

- Component :- Any element identified by decomposing a software system into its constituent parts.
- Cohesion :- the degree to which a component performs a single function.
- Coupling :- degree of linkage between one component to others in the same system.

Basic IF Model

- FAN IN , is simply a count of the numbers of other components that can call, or pass control.
- FAN OUT, is the number of components that are called by a component.
- IF(component) = [FAN IN * FAN OUT]^2

- High FAN IN indicate lack of cohesion.
- High FAN OUT also indicate a lack of cohesion.
- High IF indicate highly coupled components.

Token Count

Halstead developed software science family of measures based on tokens known as Halstead Analysis (Halstead Metrics)

Token = Operand or operator

Vocabulary

The size of the vocabulary of a program, which consists of the number of unique tokens used to build a program is defined as:

$$\eta = \eta_1 + \eta_2$$

Where

η : vocabulary of a program

η_1 : number of unique operators

η_2 : number of unique operands

Token Count

Program Length

The length of the program in the terms of the total number of tokens used is

$$N = N_1+N_2$$

Where

N : program length

N_1 : total occurrences of operators

N_2 : total occurrences of operands

Token Count

Volume

$$V = N * \log_2 \eta$$

The unit of measurement of volume is the common unit for size "bits". It is the actual size of a program if a uniform binary encoding for the vocabulary is used.

Program Level

$$L = V^* / V$$

The value of L ranges between zero and one, with L=1 representing a program written at the highest possible level (i.e., with minimum size).

V*=minimum volume of the algorithm implemented with a program as same algo can be implemented by many programs

Token Count

Program Difficulty

$D = 1 / L$

As the volume of an implementation of a program increases, the program level decreases and the difficulty increases. Thus, programming practices such as redundant usage of operands, or the failure to use higher-level control constructs will tend to increase the volume as well as the difficulty.

Effort

$E = V / L = D * V$

The unit of measurement of E is elementary mental discriminations.

For a particular programming language , if V* increases, L decreases in such a way that V*×L remains constant, this constant is called language level (λ)

Language	*Language Level* λ	*Variance* σ
PL/1	1.53	0.92
ALGOL	1.21	0.74
FORTRAN	1.14	0.81
CDC Assembly	0.88	0.42
PASCAL	2.54	–
APL	2.42	–
C	0.857	0.445

Table 1: Language levels

Token Count

- Counting rules for C language
  1. Comments are not considered.
  2. The identifier and function declarations are not considered.
  3. All the variables and constants are considered operands.
  4. Global variables used in different modules of the same program are counted as multiple occurrences of the same variable.
  5. Local variables with the same name in different functions are counted as unique operands.
  6. Functions calls are considered as operators.
  7. All looping statements e.g., do {…} while ( ), while ( ) {…}, for ( ) {…}, all control statements e.g., if ( ) {…}, if ( ) {…} else {…}, etc. are considered as operators.
  8. In control construct switch ( ) {case:…}, switch as well as all the case statements are considered as operators.

Token Count

9. The reserve words like return, default, continue, break, sizeof, etc., are considered as operators.
10. All the brackets, commas, and terminators are considered as operators.
11. GOTO is counted as an operator and the label is counted as an operand.
12. The unary and binary occurrence of “+” and “-” are dealt separately. Similarly “*” (multiplication operator) are dealt with separately.

Token Count

13. In the array variables such as “array-name [index]” “array- name” and “index” are considered as operands and [ ] is considered as operator.
14. In the structure variables such as “struct-name. member-name” or “struct-name -> member-name”, struct-name, member-name are taken as operands and ‘.’, ‘->’ are taken as operators. Same names of member elements in different structure variables are counted as unique operands.
15. All the hash directive are ignored.

Example: Count the number of operators and operands

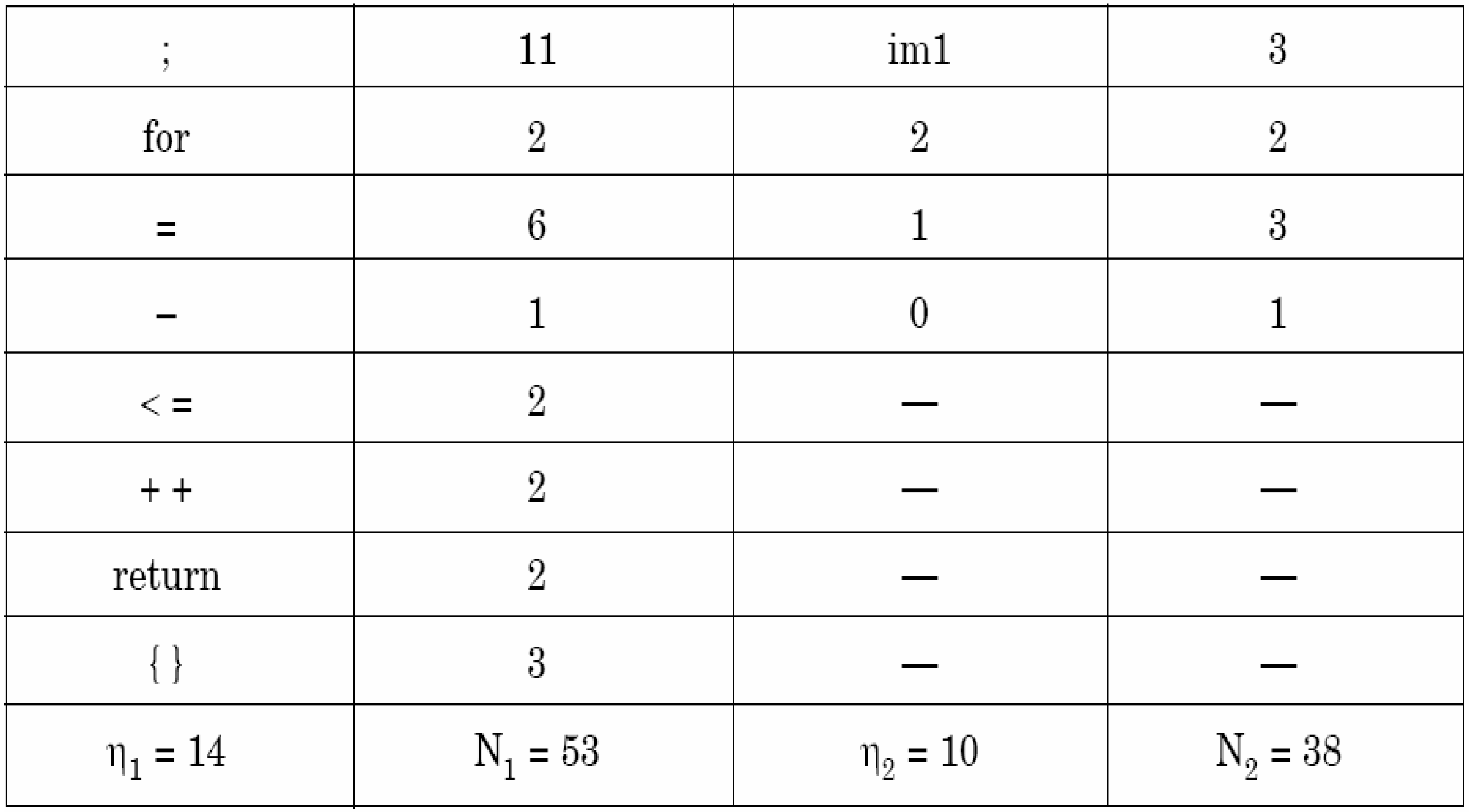
1.	int. sort (int x[ ], int n)
2.	{
3.	int i, j, save, im1;
4.	/*This function sorts array x in ascending order */
5.	If (n<2) return 1;
6.	for (i=2; i<=n; i++)
7.	{
8.	im1=i-1;
9.	for (j=1; j<=im; j++)
10.	if (x[i] < x[j])
11.	{
12.	save = x[i];
13.	x[i] = x[j];
14.	x[j] = save;
15.	}
16.	}
17.	return 0;
18.	}

Solution

The list of operators and operands is given in table 2.

Operators	*Occurrences*	*Operands*	*Occurrences*
int	4	SORT	1
( )	5	x	7
,	4	n	3
[ ]	7	i	8
if	2	j	7
<	2	save	3

(Contd.)...



;	11	im1	3
for	2	2	2
=	6	1	3
–	1	0	1
< =	2	—	—
+ +	2	—	—
return	2	—	—
{ }	3	—	—
$\eta_1 = 14$	$N_1 = 53$	$\eta_2 = 10$	$N_2 = 38$

Table 2: Operators and operands of sorting program

Token Count

- Estimated Program Length (Halstead Length estimate)

$$\hat{N} = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2$$

The following alternate expressions have been published to estimate program length.

$$N_J = log_2(\eta_1!) + \log_2(\eta_2!)$$

$$N_B = \eta_1 log_2 \eta_2 + \eta_2 \log_2 \eta_1$$

$$N_c = \eta_1(\eta_1)^{1/2} + \eta_2 (\eta_2)^{1/2}$$

$$N_s = (\eta \log_2 \eta) / 2$$

- Potential Volume

$$V^* = (2+\eta_2^*)\log_2(2+\eta_2^*)$$

Where η_2^* is the number of conceptually unique input and output parameters

- Estimated Program Level $\hat{}$/ Difficulty

Halstead offered an alternate formula that estimate the program level.

$$\hat{L} = 2\eta_2/(\eta_1 N_2)$$

where

$$\hat{D} = \frac{1}{\hat{L}} = \frac{\eta_1 N_2}{2\eta_2}$$

- Effort and Time

$$\mathrm{E} = V / \hat{L} = V * \hat{D}$$

$$= (n_1 N_2 N \log_2 \eta) / 2\eta_2$$

$$\textit{Estimated time } T = E / \beta$$

β is normally set to 18 since this seemed to give best results in Halstead’s earliest experiments, which compared the predicted times with observed programming times, including the time for design, coding, and testing.

Example- I

Consider the sorting program given. List out the operators and operands and also calculate the values of software science measures like $\eta, N, V, E, \lambda\ etc.$

1.	int. sort (int x[ ], int n)
2.	{
3.	int i, j, save, im1;
4.	/*This function sorts array x in ascending order */
5.	If (n<2) return 1;
6.	for (i=2; i<=n; i++)
7.	{
8.	im1=i-1;
9.	for (j=1; j<=im; 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.	}

Here N_1=53 and N_2=38. The program length $N=N_1+N_2$=91

Vocabulary of the program $\eta = \eta_1 + \eta_2 = 14 + 10 = 24$

Volume $V = N \times \log_2 \eta$

$= 91 \times \log_2 24 = 417$ bits

The estimated program length $\hat{N}$ of the program

$= 14 \log_2 14 + 10 \log_2 10$

$= 14 * 3.81 + 10 * 3.32$

$= 53.34 + 33.2 = 86.45$

Conceptually unique input and output parameters are represented by η_2^*

$\eta_2^* = 3$ {x: array holding the integer to be sorted}
{N: the size of the array to be sorted}
{The return value}

The potential volume $V^* = 5 \log_2 5 = 11.6$

Since $L = V^* / V$

$$= \frac{11.6}{417} = 0.027$$

D = I / L

$$= \frac{1}{0.027} = 37.03$$

Estimated program level

$$\hat{L} = \frac{2}{\eta_1} \times \frac{\eta_2}{N_2} = \frac{2}{14} \times \frac{10}{38} = 0.038$$

We may use another formula

$$\hat{E} = V / \hat{L}$$

$$E = 417 / 0.038$$

$$E = 10973.68$$

Therefore, 10974 elementary mental discrimination are required to construct the program.

$$T = E / \beta = \frac{10974}{18} = 610 \text{ seconds} = 10 \text{ minutes}$$

This is probably a reasonable time to produce the program, which is very simple

Example- 2

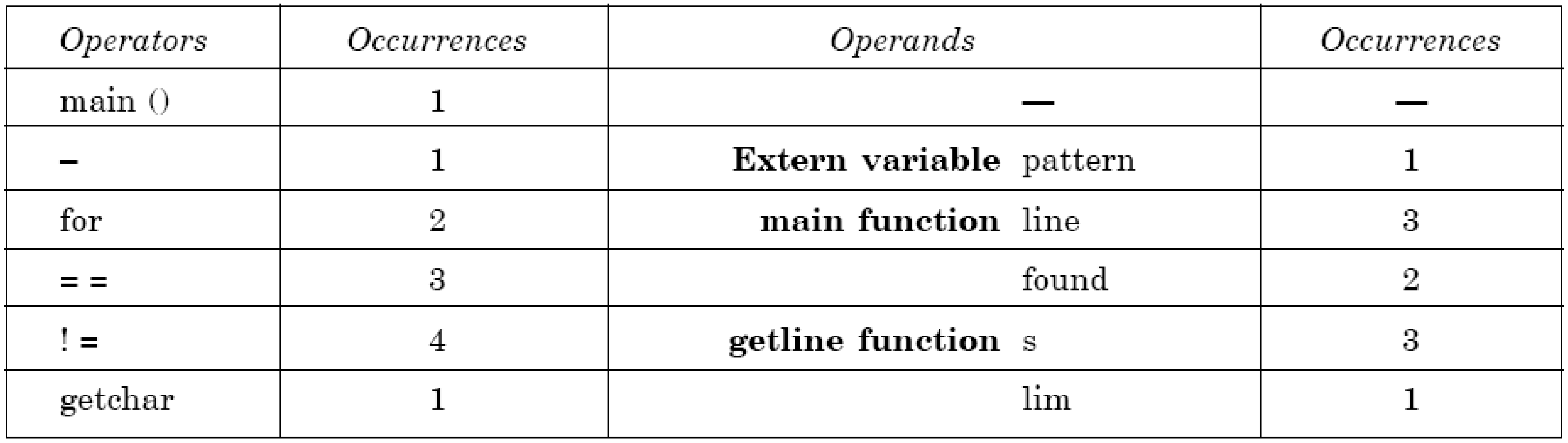
Consider the program shown in Table 3. Calculate the various software science metrics.

```
#include < stdio.h >
#define MAXLINE 100
int getline(char line[],int max);
int strindex(char source[],char search for[]);
char pattern[ ]="ould";
int main()
{
    char line[MAXLINE];
    int found = 0;
    while(getline(line,MAXLINE)>0)
        if(strindex(line, pattern)>=0)
        {
            printf("%s",line);
            found++;
        }
    return found;
}
```

```
int getline(char s[],int lim)
{
    int c,i=0;
    while(--lim > 0 && (c=getchar())!= EOF && c!='\n')
        s[i++]=c;
    if(c=='\n')
        s[i++] = c;
    s[i] = '\0';
    return i;
}
int strindex(char s[],char t[])
{
    int i,j,k;
    for(i=0;s[i] !='\0';i++)
    {
        for(j=i,k=0;t[k] != '\0',s[j] ==t[k];j++,k++);
        if(k>0 && t[k] =='\0')
            return i;
    }
    return -1;
}
```

Solution

List of operators and operands are given in Table 4.



Operators	*Occurrences*	*Operands*		*Occurrences*
main ()	1		—	—
–	1	**Extern variable**	pattern	1
for	2	**main function**	line	3
= =	3		found	2
! =	4	**getline function**	s	3
getchar	1		lim	1

Table 4

(Contd.)...

( )	1		c	5
&&	3		i	4
– –	1	**Strindex function**	s	2
return	4		t	3
+ +	6		i	5
printf	1		j	3
> =	1		k	6
strindex	1	**Numerical Operands**	1	1
If	3		MAXLINE	1
>	3		0	8
getline	1		'\0'	4
while	2		'\\n'	2
{ }	5	**strings "ould"**		1
=	10		—	—
[ ]	9		—	—
,	6		—	—
;	14		—	—
EOF	1		—	—
$\eta_1 = 24$	$N_1 = 84$		$\eta_2 = 18$	$N_2 = 55$

Program vocabulary $\eta = 42$

Program length $N = N_1 + N_2$

$= 84 + 55 = 139$

Estimated length $\hat{N} = 24\log_2 24 + 18\log_2 18 = 185.115$

% error $= 24.91$

Program volume $V = 749.605$ bits

Estimated program level $= \frac{2}{\eta_1} \times \frac{\eta_2}{N_2}$

$$= \frac{2}{24} \times \frac{18}{55} = 0.02727$$

Minimal volume V*=20.4417

Effort $= V / \hat{L}$

$$= \frac{748.605}{.02727}$$

= 27488.33 elementary mental discriminations.

Time T = $E / \beta = \dfrac{27488.33}{18}$

= 1527.1295 seconds

= 25.452 minutes

Maintenance Metrics

- There are two categories of maintenance key performance indicators which include the **leading and lagging indicators**. The leading indicators signal future events and the lagging indicators follow the past events.
- Using these maintenance metrics and turning the data into actionable information, organizations can acquire both qualitative and quantitative insights.

Maintenance Metrics

1. Planned maintenance percentage (PMP)

- This metrics represents the percentage of time spent on planned maintenance activities against the unplanned

PMP= (scheduled maintenance time/total maintenance time) x 100

- In a good system, 90% of the maintenance should be planned.

Maintenance Metrics

2. Overall Equipment Effectiveness (OEE)

- OEE is the measure of the productivity of a piece of equipment. It gives informed data on how effective organization’s maintenance processes is running based on factors like equipment quality, performance, and availability
- A 100% OEE means that your system is producing no defects, as fast as possible, and with no stops in the production.

OEE = availability x performance x quality

Maintenance Metrics

- **Availability**
  - Availability takes into account Availability Loss, which includes any events that stop planned production for an appreciable length of time (usually several minutes; long enough for an operator to log a reason)
  - The remaining time after Availability Loss is subtracted is called Run Time.
  - Runtime = Total time – Available Loss

Maintenance Metrics

- PERFORMANCE
  - Performance takes into account Performance Loss, which accounts for anything that causes the manufacturing process to run at less than the maximum possible speed when it is running (including both Slow Cycles and Small Stops).
  - The remaining time after Performance Loss is subtracted is called Net Run Time

Maintenance Metrics

- QUALITY
  - Quality takes into account Quality Loss, which accounts for outputs that do not meet quality standards.
  - The remaining time after Quality Loss is subtracted is called Fully Productive Time.
  - Our goal is not simply to measure OEE – it is to maximize Fully Productive Time.

Maintenance Metrics

3. Mean time to repair (MTTR)

- The MTTR clock starts ticking when the repairs start and it goes on until operations are restored. This includes repair time, testing period, and return to the normal operating condition.

MTTR= (SUM of downtime periods/ total number of repairs)

Maintenance Metrics

4. Mean time between failure (MTBF)

- MTBF is the measure of the predicted time between one breakdown to the next during normal operation.
- Higher MTBF means that the software (or product) will work longer before it experiences failure

MTBF= (SUM of operational time/total number of failures)

Maintenance Metrics

5. Preventive maintenance compliance (PMC)

- PM compliance is defined as the percentage of the preventive work scheduled and completed in a set time.
- For example, we might have 60 operations (that are a part of the PM plan) scheduled but 51 completed at the end of the month.

  In this case:

  $PMC = (51/60) \times 100 = 85\%$