Q. 12. **Suppose you are managing a software development project. The project is expected to be completed in 8 months at a cost of $10000 per month. After 2 months, you realize that the project is 30% completed at a cost of $40,000. You need to determine whether the project is on-time and on-budget after 2 months. Let's see how healthy the project is by calculating the cost variance and schedule variance.**

Ans. We can solve the problem in two steps as follows:

Step1 : Calculate the Planned Value and Earned Value

From the scenario :

- Budget at Completion (BAC) = $ 10,000 * 8 = $ 80,000
- Actual Cost (AC) = $ 40,000
- Planned Completion = 2/8 = 25 %
- Actual Completion = 30 %

Therefore,

- Planned Value = Planned Completion (%) * BAC = 25% * $ 80,000 = $ 20,000
- Earned Value = Actual Completion (%) * BAC = 30 % * $ 80,000 = $ 24,000

Step 2: Compute the earned value management cost and schedule variances:

- Cost Variance = EV – AC = $ 24,000 – $ 40,000 = – $ 16,000
- Schedule Variance = EV – PV = $ 24,000 – $ 20,000 = $ 4,000

Q. 41. For a given source code calculate the Halestead's software metrics

```
                    Z = 0;
While               X > 0
                    Z = Z + Y;
                    X = X – 1;

Endwhile;
print(Z);
```

Ans. The operator and operand count will be as fllows :

Operators	*Occurrences*	*Operands*	*Occurrences*
=	3	Z	4
;	5	0	2
While-End while	1	X	3
>	1	Y	1
+	1	1	1
–	1		
Print	1		
()	1		
$n_1 = 8$	$N_1 = 14$	$n_2 = 5$	$N_2 = 11$

Halestead's Software metrics :

Program Length	$N = N_1 + N_2$	$N = 11 + 14$	$N = 25$
Program Vocabulary	$n = n_1 + n_2$	$n = 8 + 5$	$n = 13$
Predicted Prog. Length	$N' = n_1 \log_2 n_1 + n_2 \log_2 n_2$	$N'\ 8\log_2 8 + 5 \log_2 5$	$N' = 35.5$
Program Volume	$V = N \log_2 (n_1 + n_2)$		$V = 92.510$
Potential Volume	$V^* = (2 + n_2)\log_2 (n_1 + n_2)$		$V^* = 8$
Program Level	$L = V^*/V$	$L = 8/92.510$	$L = 0.086$
Program Efforts	$E = V/L$		$E = 1075.69$
Time	$T = E/S$	$T = 1075.69/18$	$T = 59.76$ sec
Program Difficulty	$D = 1/L$	$D = 1/0.086$	$D = 11.62$

Q. 42. Calculate Halestead's measure on factorial code given below :

```
int fact (int n) {
if (n == 0)
{
(return 1;)
}
else
{return n * fact (n – 1); } }
```

Ans. The table factorial of operand and operators is as follows :

(*i*) Program length $N = N_1 + N_2 = 18 + 8 = 26$

(*ii*) Vocabulary of program $n = n_1 + n_2 = 10 + 3 = 13$

Operator	*Occurrence*	*Operand*	*Occurrence*
int	2	*n*	4
( )	4	fact	2
if	1	1	2
;	2		
= =	1		
–	1		
return	2		
else	1		
*	1		
{ }	3		
$n_1 = 10$	$N_1 = 18$	$n_2 = 3$	$N_2 = 8$

(*iii*) Volume $V = N \log_2 n = 26 \log_2 13 = 96.211$

(*iv*) Estimated length $N' = n_1 \log_2 n_1 + n_2 \log_2 n_2 = 10 \log_2 10 + 3 \log_2 3$
$= 33.219 + 4.754 = 37.973$

(*v*) Effort $E = \frac{n_1 N_2}{2n_2}(N \log_2 n) = \frac{10 \times 8}{2 \times 3}(26 \log_2 12) = \frac{10 \times 8 \times 93.209}{6}$
$= 1242.787$

Q. 43. For the following 'C' program estimate the Halstead's length and volume measures. Compare Halstead's length and volume measures of size with LOC measure.

```
/*Program to calculate GCD of two numbers*/
int compute-gcd (x, y)
   while (x != y)
   if (x > y) then x = x – y;
   else y = y – x;
   return x;
}
```

Ans.

Operator	*Occurrence*	*Operand*	*Occurrence*
Compute-gcd	1	x	7
=	2	y	6
!=	1		
while	1		
>	1		
–	2		
return	1		
if	1		
{}	1		
()	3		
;	3		
int	1		
else	1		
then	1		
.....	.....	.....	.....
$n_1 = 14$	$N_1 = 20$	$n_2 = 2$	$N_2 = 13$

Q. 45. **Consider a project with the following functional units :**

Number of user inputs = 50
Number of user outputs = 40
Number of user inquiries = 35
Number of user files = 6
Number of external interfaces = 4
Assume all complexity adjustment factors and weighting factors are average. Compute the function points for the project.

Ans.

$$UFP = \sum_{i=3}^{5}\sum_{j=1}^{3} Z_{ij} * W_{ij}$$

$$UFP = 50 \times 4 + 40 \times 5 + 35 \times 4 + 6 \times 10 + 4 \times 7$$
$$= 200 + 200 + 140 + 60 + 28$$
$$= 628$$
$$CAF = (0.65 + 0.01\ \Sigma F_i)$$
$$= (0.65 + 0.01\ (14 \times 3))$$
$$= 0.65 + 0.42 = 1.07$$
$$FP = UFP \times CAF$$
$$= 628 \times 1.07$$
$$= 672$$

Q. 46. **Consider a project with the following parameters :**

(*i*) **External Inputs :**
- (*a*) **10 with low complexity**
- (*b*) **15 with average complexity**
- (*c*) **17 with high complexity**

(*ii*) **External Outputs :**
- (*a*) **6 with low complexity**
- (*b*) **13 with high complexity**

(*iii*) **External Inquiries :**
- (*a*) **3 with low complexity**
- (*b*) **4 with average complexity**
- (*c*) **2 with high complexity**

(*iv*) **Internal Logical Files :**
- (*a*) **2 with average complexity**
- (*b*) **1 with high complexity**

(v) External Interface Files :

(a) 9 with low complexity

In addition to above, system requires

(i) Significant data, communication

(ii) Performance is very critical

(iii) Designed code may be moderately reusable

(iv) System is not designed for multiple installations in different organizations.

Other complexity adjustment factors are treated as average. Compute the function points for the project.

Ans. To calculate the function point firstly, we have to calculate the un-adjusted functional point counts as follows :

Functional Units	*Count*	*Complexity*		*Complexity Total*	*Functional Unit totals*
External Inputs (ELs)	10 15 17	Low × 3 Average × 4 High × 6	= = =	30 60 102	192
External Outputs (EOs)	6 0 13	Low × 4 Average × 5 High × 7	= = =	24 0 91	115
External Inquiries (EQs)	3 4 2	Low × 3 Average × 4 High × 6	= = =	9 16 12	37
Internal Logical Files (ILFs)	0 2 1	Low × 7 Average × 10 High × 15	= = =	0 20 15	35
External Interface Files (EIFs)	9 0 0	Low × 5 Average × 7 High × 10	= = =	45 0 0	45
		Total unadjusted function point count = 424			

The factors given previously can be calculated as :

$$\sum_{i=1}^{14} F_i = 3+4+3+5+3+3+3+3+3+3+2+3+0+3 = 41$$

$$\begin{aligned} CAF &= (0.65 + 0.01 * \Sigma F_i) \\ &= (0.65 + 0.01 * 41) \\ &= 1.06 \\ FP &= UFP * CAF \\ &= 424 * 1.06 \\ &= 449.44 \end{aligned}$$

Q. 48. Consider a flow graph given, calculate its cyclomatic complexity.

Ans. Cyclomatic complexity can be calculated by any of the three methods :

1. $V(G) = e - n + 2P$
   $= 13 - 10 + 2 = 5$
2. $V(G) = n + 1$
   $= 4 + 1 = 5$
3. $V(G) =$ number of regions
   $= 5$

Therefore, complexity value of a flow graph is 5.

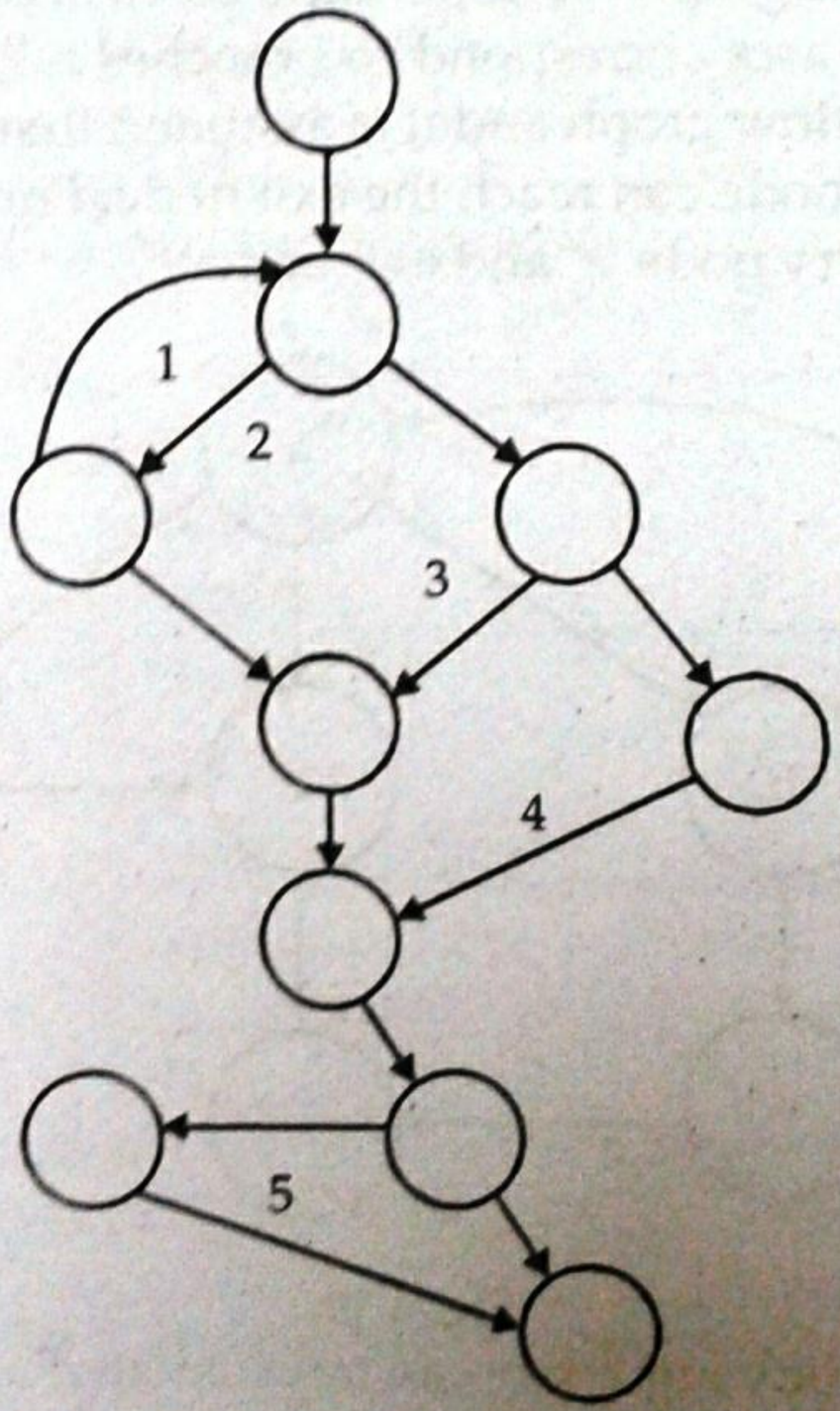


Fig. 4.20.

Table 4.2: UFP calculation table

Functional Units	*Count Complexity*		*Complexity Totals*	*Functional Unit Totals*
External Inputs (EIs)	☐	Low × 3	= ☐	☐
	☐	Average × 4	= ☐	
	☐	High × 6	= ☐	
External Outputs (EOs)	☐	Low × 4	= ☐	☐
	☐	Averge × 5	= ☐	
	☐	High × 7	= ☐	
External Inquiries (EQs)	☐	Low × 3	= ☐	☐
	☐	Average × 4	= ☐	
	☐	High × 6	= ☐	
Internal logical files (ILFs)	☐	Low × 7	= ☐	☐
	☐	Average × 10	= ☐	
	☐	High × 15	= ☐	
External Interface files (EIFs)	☐	Low × 5	= ☐	☐
	☐	Average × 7	= ☐	
	☐	High × 10	= ☐	
Total Unadjusted Function Point Count				☐

The weighting factors are identified (as per Table 4.1) for all functional units and multiplied with the functional units accordingly. The procedure for the calculation of Unadjusted Function Point (UFP) is given in Table 4.2.

The procedure for the calculation of UFP in mathematical form is given below:

$$UFP = \sum_{i=1}^{5} \sum_{J=1}^{3} Z_{ij} w_{ij}$$

where i indicates the row and j indicates the column of Table 4.1.

w_{ij} : It is the entry of the i^{th} row and j^{th} 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.

Organisations that use function point methods develop a criterion for determining whether a particular entry is Low, Average or High. Nonetheless, the determination of complexity is somewhat subjective.

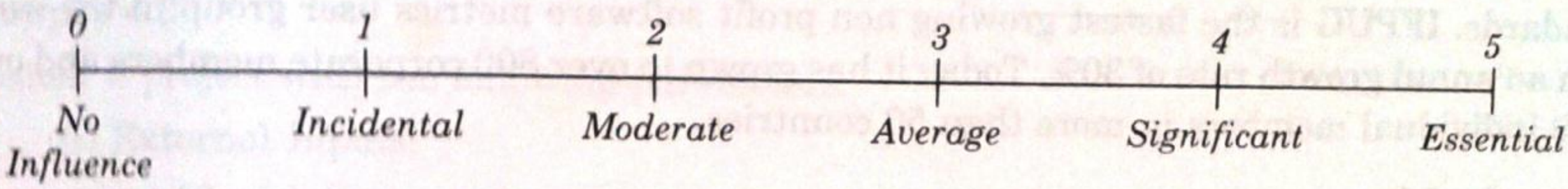
The final number of function points is arrived at by multiplying the UFP by an adjustment factor that is determined by considering 14 aspects of processing complexity which are given in Table 4.3. This adjustment factor allows the UFP count to be modified by at most ± 35%. The final adjusted FP count is obtained by using the following relationship

$$FP = UFP * CAF$$

Where CAF is complexity adjustment factor and is equal to $[0.65 + 0.01 \times \Sigma F_i]$. The F_i (i = 1 to 14) are the degrees of influence and are based on responses to questions noted in Table 4.3.

Table 4.3: Computing function points.

Rate each factor on a scale of 0 to 5.



Number 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. Is 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 use by the user?

Uses of function points

The collection of function point data has two primary motivations. One is the desire by managers to monitor levels of productivity, for example, number of function points achieved per work hour expended. From this perspective, the manager is not concerned with when the function point counts are made, but only that the function points accurately describe the size of the final software project. In this instance, function points have an advantage over LOC in that they provide a more objective measure of software size by which to assess productivity.

Another use of function points is in the estimation of software development cost. There are only a few studies that address this issue, though it is arguably the most important potential use of function point data.

Functions points may compute the following important metrics:

Productivity = FP/persons-months
Quality = Defects/FP
Cost = Rupees /FP
Documentation = Pages of documentation per FP

These metrics are controversial and are not universally acceptable. There are standards issued by the International Function Point User Group (IFPUG, covering the Albrecht method) and the United Kingdom Function Point User Group (UFPGU, covering the MK11 method). An ISO standard for function point methods is also being developed.

The function point method continues to be refined. So if we intend to use it we should obtain copies of the latest international function point user group (IFPUG) guidelines and standards. IFPUG is the fastest growing non profit software metrics user group in the world with an annul growth rate of 30%. Today it has grown to over 800 corporate members and over 1500 individual members in more than 50 countries.

Example 4.1

Consider a project with the following functional units:

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

Assume all complexity adjustment factors and weighting factors are average.

Compute the function points for the project.

Solution

We know $$UFP = \sum_{i=1}^{5} \sum_{J=1}^{3} Z_{ij} w_{ij}$$

$$UFP = 50 \times 4 + 40 \times 5 + 35 \times 4 + 6 \times 10 + 4 \times 7$$
$$= 200 + 200 + 140 + 60 + 28 = 628$$
$$CAF = (0.65 + 0.01 \Sigma F_i)$$
$$= (0.65 + 0.01(14 \times 3)) = 0.65 + 0.42 = 1.07$$
$$FP = UFP \times CAF$$
$$= 628 \times 1.07 = 672.$$

Example 4.2

An application has the following:

10 low external inputs, 12 high external outputs, 20 low internal logical files, 15 high external interface files, 12 average external inequiries, and a value of complexity adjustment factor of 1.10.

What are the unadjusted and adjusted function point counts ?

Solution

Unadjusted function point counts may be calculated using as:

$$\text{UFP} = \sum_{i=1}^{5} \sum_{\text{J}=1}^{3} \text{Z}_{ij}\, w_{ij}$$

$$= 10 \times 3 + 12 \times 7 + 20 \times 7 + 15 + 10 + 12 \times 4$$

$$= 30 + 84 + 140 + 150 + 48$$

$$= 452$$

$$\text{FP} = \text{UFP} \times \text{CAF}$$

$$= 452 \times 1.10 = 497.2.$$

Example 4.3

Consider a project with the following parameters.

(*i*) External Inputs:

(*a*) 10 with low complexity

(*b*) 15 with average complexity

(*c*) 17 with high complexity.

(*ii*) External Outputs:

(*a*) 6 with low complexity

(*b*) 13 with high complexity.

(*iii*) External Inquiries:

(*a*) 3 with low complexity

(*b*) 4 with average complexity

(*c*) 2 with high complexity.

(*iv*) Internal logical files:

(*a*) 2 with average complexity

(*b*) 1 with high complexity.

(*v*) External Interface files:

(*a*) 9 with low complexity.

In addition to above, system requires

(*i*) Significant data communication

(*ii*) Performance is very critical

(*iii*) Designed code may be moderately reusable

(*iv*) System is not designed for multiple installations in different organisations.

Other complexity adjustment factors are treated as average. Compute the function points for the project.

Solution

Unadjusted function points may be counted using Table 4.2.

Functional Units	*Count*	*Complexity*			*Complexity Totals*	*Functional Unit totals*
External Inputs (EIs)	10	Low	× 3	=	30	
	15	Average	× 4	=	60	
	17	High	× 6	=	102	192
External Outputs (EOs)	6	Low	× 4	=	24	
	0	Average	× 5	=	0	
	13	High	× 7	=	91	115
External Inquiries (EIs)	3	Low	× 3	=	9	
	4	Average	× 4	=	16	
	2	High	× 6	=	12	37
Internal Logical Files (ILFs)	0	Low	× 7	=	0	
	2	Average	× 10	=	20	
	1	High	× 15	=	15	35
External Interface Files (EIFs)	9	Low	× 5	=	45	
	0	Avergae	× 7	=	0	
	0	High	× 10	=	0	45
		Total unadjusted function point count =				424

The factors given in Table 4.3 may be calculated as:

$$\sum_{i=1}^{14} F_i = 3 + 4 + 3 + 5 + 3 + 3 + 3 + 3 + 3 + 3 + 2 + 3 + 0 + 3 = 41$$

$$CAF = (0.65 + 0.01 \times \Sigma F_i)$$
$$= (0.65 \times 0.01 \times 41)$$
$$= 1.06$$
$$FP = UFP \times CAF$$
$$= 424 \times 1.06$$
$$= 449.44$$

Hence $\boxed{FP = 449}$

Example 4.5

Suppose that a project was estimated to be 400 KLOC. Calculate the effort and development time for each of the three modes *i.e.*, organic, semidetached and embedded.

Solution

The basic COCOMO equations take the form:

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

$$D = c_b(KLOC)^{d_b}$$

Estimated size of the project = 400 KLOC

(*i*) **Organic mode**

$$E = 2.4(400)^{1.05} = 1295.31 \text{ PM}$$

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

(*ii*) **Semidetached mode**

$$E = 3.0(400)^{1.12} = 2462.79 \text{ PM}$$

$$D = 2.5(2462.79)^{.35} = 38.45 \text{ M}$$

(*iii*) **Embedded mode**

$$E = 3.6(400)^{1.20} = 4772.81 \text{ PM}$$

$$D = 2.5(4772.8)^{.32} = 38 \text{ M}$$

As we have seen, effort calculated for embedded mode is approximately 4 times, the effort for organic mode. However, the effort calculated for semidetached mode is 2 times the effort of organic mode. There is a large difference in these values. But, surprisingly, the development time is approximately the same for all three modes. It is clear from here that the

Example 4.6

A project size of 200 KLOC is to be developed. Software development team has average experience on similar type of projects. The project schedule is not very tight. Calculate the effort, development time, average staff size and productivity of the project.

Solution

The semi-detached mode is the most appropriate mode; keeping in view the size, schedules and experience of the development team.

Hence $E = 3.0(200)^{1.12} = 1133.12$ PM

$D = 2.5(1133.12)^{.35} = 29.3$ M

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

$= \frac{1133.12}{29.3} = 38.67$ Persons.

Productivity $= \frac{KLOC}{E} = \frac{200}{1133.12} = 0.1765$ KLOC/PM

$P = 176$ LOC/PM.

Example 6.1

Consider the sorting program given in Fig. 4.2 of chapter 4 (software project planning). List out the operators and operands and also calculate the values of software science measures like η, N, V, E, λ etc.

Solution

The list of operators and operands is given in Table 6.2.

Table 6.2: Operators and operands of sorting program of Fig. 4.2 of chapter 4.

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	—	—
$\eta_1 = 14$	$N_1 = 53$	$\eta_2 = 10$	$N_2 = 38$

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.

If a binary encoded scheme is used to represent each of the 24 items in the vocabulary, it would take 5 bits per item, since a 4 bit scheme leads to 16 unique codes (which is not enough), and a 5-bit scheme leads to 32 unique codes (which is more than sufficient). Each of the 91 tokens used in the program could be represented in order by a 5-bit code, leading to a string of $5 \times 91 = 455$ bits that would allow us to store the entire program in memory. Notice that size analysis is based on storing not a compiled version of the subroutine, but a binary translation of the original program. We could then say that this program occupies 455 bits of storage in its encoded form. However, also notice that a 5-bit scheme allows for 32 tokens instead of just 24 tokens, so instead of using the integer 5, volume uses the non-integer $\log_2 \eta = 4.58$ to arrive at a slightly smaller volume of 417.

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. This is used both as input and output}.

{N : the size of the array to be sorted}.

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

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

which is not very close to the 0.027 level, we determined earlier using the conceptually unique operands.

We may use another formula

$$\hat{V}^* = V \times \hat{L} = 417 \times 0.038 = 15.67$$

The discrepancy between V^* and $\hat{V}^*$ does not inspire confidence in the application of this portion of software science theory to more complicated programs.

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

$$= 417 / 0.038 = 10973.68$$

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

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

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

Table 6.3

```
#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;
}
```

Example 6.2

Consider the program shown in Table 6.3. Calculate the various software science metrics.

Solution

List of operators and operands are given in Table 6.4.

Table 6.4

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

(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 $= V/\hat{L}$

$= \frac{749.605}{.02727}$

$= 27488.33$ elementary mental discriminations

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

$$= 1527.1295 \text{ seconds}$$

$$= 25.452 \text{ minutes}$$

Example 8.1

Consider a program for the determination of the nature of roots of a quadratic equation. Its input is a triple of positive integers (say a, b, c) and values may be from interval [0, 100]. The program output may have one of the following words:

[Not a quadratic equation; Real roots; Imaginary roots; Equal roots]

Design the boundary value test cases.

Solution

Quadratic equation will be of type:

$$ax^2 + bx + c = 0$$

Roots are real if $(b^2 - 4ac) > 0$

Roots are imaginary if $(b^2 - 4ac) < 0$

Roots are equal if $(b^2 - 4ac) = 0$

Equation is not quadratic if $a = 0$

The boundary value test cases are:

Test Case	*a*	*b*	*c*	*Expected output*
1	0	50	50	Not Quadratic
2	1	50	50	Real Roots
3	50	50	50	Imaginary Roots
4	99	50	50	Imaginary Roots
5	100	50	50	Imaginary Roots
6	50	0	50	Imaginary Roots
7	50	1	50	Imaginary Roots
8	50	99	50	Imaginary Roots
9	50	100	50	Equal Roots
10	50	50	0	Real Roots
11	50	50	1	Real Roots
12	50	50	99	Imaginary Roots
13	50	50	100	Imaginary Roots

Example 8.2

Consider a program for determining the Previous date. Its input is a triple of day, month and year with the values in the range

$$1 \leq \text{month} \leq 12$$

$$1 \leq \text{day} \leq 31$$

$$1900 \leq \text{year} \leq 2025$$

The possible outputs would be Previous date or invalid input date. Design the boundary value test cases.

Solution

The Previous date program takes a date as input and checks it for validity. If valid, it returns the previous date as its output.

As we know, with single fault assumption theory, $4n + 1$ test cases can be designed and which are equal to 13. The boundary value test cases are:

Test case	*Month*	*Day*	*Year*	*Expected output*
1	6	15	1900	14 June, 1900
2	6	15	1901	14 June, 1901
3	6	15	1962	14 June, 1962
4	6	15	2024	14 June, 2024
5	6	15	2025	14 June, 2025
6	6	1	1962	31 May, 1962
7	6	2	1962	1 June, 1962
8	6	30	1962	29 June, 1962
9	6	31	1962	Invalid date
10	1	15	1962	14 January, 1962
11	2	15	1962	14 February, 1962
12	11	15	1962	14 November, 1962
13	12	15	1962	14 December, 1962

Example 8.3

Consider a simple program to classify a triangle. Its input is a triple of positive integers (say x, y, z) and the data type for input parameters ensures that these will be integers greater than 0 and less than or equal to 100. The program output may be one of the following words:

[Scalene; Isosceles; Equilateral; Not a triangle]

Design the boundary value test cases.

Solution

The boundary value test cases are shown below:

Test case	x	y	z	*Expected output*
1	50	50	1	Isosceles
2	50	50	2	Isosceles
3	50	50	50	Equilateral
4	50	50	99	Isosceles
5	50	50	100	Not a triangle
6	50	1	50	Isosceles
7	50	2	50	Isosceles
8	50	99	50	Isosceles

(Contd.)...

Test case	*x*	*y*	*z*	*Expected output*
9	50	100	50	Not a triangle
10	1	50	50	Isosceles
11	2	50	50	Isosceles
12	99	50	50	Isosceles
13	100	50	50	Not a triangle

classes based on input and output domains.

Example 8.7

Consider the program for the determination of nature of roots of a quadratic equation as explained in Example 8.1. Identify the equivalence class test cases for output and input domains.

Solution

Output domain equivalence class test cases can be identified as follows:

$O_1 = \{< a, b, c >$: Not a quadratic equation if $a = 0\}$

$O_2 = \{< a, b, c >$: Real roots if $(b^2 - 4ac) > 0\}$

$O_3 = \{< a, b, c >$: Imaginary roots if $(b^2 - 4ac) < 0\}$

$O_4 = \{< a, b, c >$: Equal roots if $(b^2 - 4ac) = 0\}$

The number of test cases can be derived from above relations and shown below:

Test case	*a*	*b*	*c*	*Expected output*
1	0	50	50	Not a quadratic equation
2	1	50	50	Real roots
3	50	50	50	Imaginary roots
4	50	100	50	Equal roots

We may have another set of test cases based on input domain.

$I_1 = \{a: a = 0\}$

$I_2 = \{a: a < 0\}$

$I_3 = \{a: 1 \le a \le 100\}$

$I_4 = \{a: a > 100\}$

$I_5 = \{b: 0 \le b \le 100\}$

$I_6 = \{b: b < 0\}$

$I_7 = \{b: b > 100\}$

$I_8 = \{c: 0 \le c \le 100\}$

$I_9 = \{c: c < 0\}$

$I_{10} = \{c: c > 100\}$

In these classes, our basic assumption is single fault theory. Hence one value is at an extreme and other values are nominal values. Input domain test cases are:

Test case	*a*	*b*	*c*	*Expected output*
1	0	50	50	Not a quadratic equation
2	– 1	50	50	Invalid input
3	50	50	50	Imaginary roots
4	101	50	50	Invalid input
5	50	50	50	Imaginary roots
6	50	– 1	50	Invalid input
7	50	101	50	Invalid input
8	50	50	50	Imaginary roots
9	50	50	– 1	Invalid input
10	50	50	101	Invalid input

Here test cases 5 and 8 are redundant test cases. If we choose any value other than nominal, we may not have redundant test cases. Hence total test cases are 10 + 4 = 14 for this problem.

Example 8.8

Consider the program for determining the Previous date in a calendar as explained in Example 8.3. Identify the equivalence class test cases for output & input domains.

Solution

Output domain equivalence classes are:

$O_1 = \{< D, M, Y >:$ Previous date if all are valid inputs$\}$

$O_2 = \{< D, M, Y >:$ Invalid date if any input makes the date invalid$\}$

Test case	*M*	*D*	*Y*	*Expected output*
1	6	15	1962	14 June, 1962
2	6	31	1962	Invalid date

We may have another set of test cases which are based on input domain.

I_1 = {month: $1 \le m \le 12$}

I_2 = {month: $m < 1$}

I_3 = {month: $m > 12$}

I_4 = {day: $1 \le D \le 31$}

I_5 = {day: $D < 1$}

I_6 = {day: $D > 31$}

I_7 = {year: $1900 \le Y \le 2025$}

I_8 = {year: $Y < 1900$}

I_9 = {year: $Y > 2025$}

Input domain test cases are :

Test case	*M*	*D*	*Y*	*Expected output*
1	6	15	1962	14 June, 1962
2	– 1	15	1962	Invalid input
3	13	15	1962	Invalid input
4	6	15	1962	14 June, 1962
5	6	– 1	1962	Invalid input
6	6	32	1962	Invalid input
7	6	15	1962	14 June, 1962
8	6	15	1899	Invalid input (value out of range)
9	6	15	2026	Invalid input (value out of range)

Example 8.9

Consider the triangle problem specified in example 8.3. Identify the equivalence class test cases for output and input domain.

Solution

Output domain equivalence classes are:

O_1 = {$< x, y, z >$: Equilateral triangle with sides x, y, z}

O_2 = {$< x, y, z >$: Isosceles trinagle with sides x, y, z}

O_3 = {$< x, y, z >$: Scalene triangle with sides x, y, z}

O_4 = {$< x, y, z >$: Not a triangle with sides x, y, z}

The test cases are:

Test case	*x*	*y*	*z*	*Expected output*
1	50	50	50	Equilateral
2	50	50	99	Isosceles
3	100	99	50	Scalene
4	50	100	50	Not a triangle

Input domain based classes are:

$I_1 = \{x: x < 1\}$

$I_2 = \{x: x > 100\}$

$I_3 = \{x: 1 \leq x \leq 100\}$

$I_4 = \{y: y < 1\}$

$I_5 = \{y: y > 100\}$

$I_6 = \{y: 1 \leq y \leq 100\}$

$I_7 = \{z: z < 1\}$

$I_8 = \{z: z > 100\}$

$I_9 = \{z: 1 \leq z \leq 100\}$

Some input domain test cases can be obtained using the relationship amongst x, y and z.

$I_{10} = \{< x, y, z >: x = y = z\}$

$I_{11} = \{< x, y, z >: x = y, x \neq z\}$

$I_{12} = \{< x, y, z >: x = z, x \neq y\}$

$I_{13} = \{< x, y, z >; y = z, x \neq y\}$

$I_{14} = \{< x, y, z >: x \neq y, x \neq z, y \neq z\}$

$I_{15} = \{< x, y, z >: x = y + z\}$

$I_{16} = \{< x, y, z >: x > y + z\}$

$I_{17} = \{< x, y, z >: y = x + z\}$

$I_{18} = \{< x, y, z >: y > x + z\}$

$I_{19} = \{< x, y, z >: z = x + y\}$

$I_{20} = \{< x, y, z >; z > x + y\}$

Test cases derived from input domain are:

Test case	*x*	*y*	*z*	*Expected output*
1	0	50	50	Invalid input
2	101	50	50	Invalid input
3	50	50	50	Equilateral
4	50	0	50	Invalid input
5	50	101	50	Invalid input
6	50	50	50	Equilateral
7	50	50	0	Invalid input
8	50	50	101	Invalid input
9	50	50	50	Equilateral
10	60	60	60	Equilateral
11	50	50	60	Isosceles
12	50	60	50	Isosceles
13	60	50	50	Isosceles
14	100	99	50	Scalene
15	100	50	50	Not a triangle
16	100	50	25	Not a triangle
17	50	100	50	Not a triangle
18	50	100	25	Not a triangle
19	50	50	100	Not a triangle
20	25	50	100	Not a triangle

Example 8.15

Consider a flow graph given in the Fig. 8.23 and calculate the cyclomatic complexity by all three methods.

Solution

Cyclomatic complexity can be calculated by any of the three methods.

1. V(G) = $e - n + 2P$
   = 13 – 10 + 2 = 5
2. V(G) = $\Pi + 1$
   = 4 + 1 = 5
3. V(G) = number of regions
   = 5

Therefore, complexity value of a flow graph in Fig. 8.23 is 5.

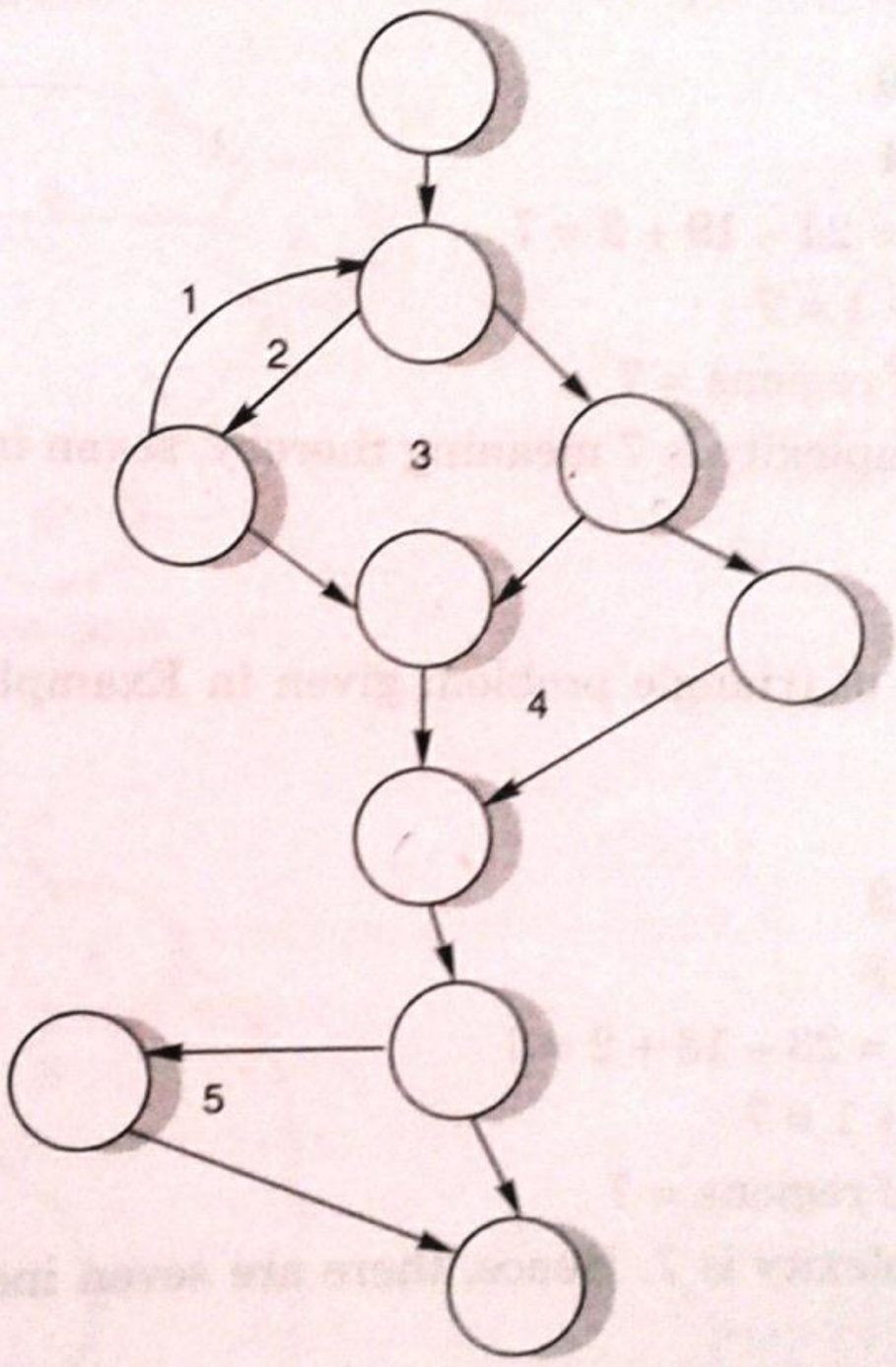


Fig. 8.23: [MCCA76]

Example 8.16

Consider the previous date program with DD path graph given in Fig. 8.17. Find cyclomatic complexity.

Solution

Number of edges (e) = 65

Number of nodes (n) = 49

(i) V(G) = $e - n + 2P$ = 65 – 49 + 2 = 18

(ii) V(G) = $\Pi + 1$ = 17 + 1 = 18

(iii) V(G) = Number of regions = 18.

Example 8.17

Consider the quadratic equation problem given in Example 8.13 with its DD Path graph. Find the cyclomatic complexity:

Solution

Number of nodes = 19

Number of edges = 24

(*i*) $V(G) = e - n + 2P = 24 - 19 + 2 = 7$

(*ii*) $V(G) = \Pi + 1 = 6 + 1 = 7$

(*iii*) V(G) = Number of regions = 7

Hence cyclomatic complexity is 7 meaning thereby, seven independent paths in the DD Path graph.

Example 8.18

Consider the classification of triangle problem given in Example 8.14. Find the cyclomatic complexity.

Solution

Number of edges = 23

Number of nodes = 18

(*i*) $V(G) = e - n + 2P = 23 - 18 + 2 = 7$

(*ii*) $V(G) = \Pi + 1 = 6 + 1 = 7$

(*iii*) V(G) = Number of regions = 7

The cyclomatic complexity is 7. Hence, there are seven independent paths as given in Example 8.14.