Software Metrics

Software Metrics

Software Metrics: What and Why ?

1. How to measure the size of a software?
2. How much will it cost to develop a software?
3. How many bugs can we expect?
4. When can we stop testing?
5. When can we release the software?

Software Metrics

6. What is the complexity of a module?
7. What is the module strength and coupling?
8. What is the reliability at the time of release?
9. Which test technique is more effective?
10. Are we testing hard or are we testing smart?
11. Do we have a strong program or a week test suite?

Software Metrics

- Pressman explained as “A measure provides a quantitative indication of the extent, amount, dimension, capacity, or size of some attribute of the product or process”.
- Measurement is the act of determine a measure
- The metric is a quantitative measure of the degree to which a system, component, or process possesses a given attribute.
- Fenton defined measurement as “ it is the process by which numbers or symbols are assigned to attributes of entities in the real world in such a way as to describe them according to clearly defined rules”.

Software Metrics

- Definition

Software metrics can be defined as "*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

- Areas of Applications

The most established area of software metrics is cost and size estimation techniques.

The prediction of quality levels for software, often in terms of reliability, is another area where software metrics have an important role to play.

The use of software metrics to provide quantitative checks on software design is also a well established area.

Software Metrics

- Problems During Implementation

➢ Statement	: Software development is to complex; it cannot be managed like other parts of the organization.
Management view	: Forget it, we will find developers and managers who will manage that development.
➢ Statement	: I am only six months late with project.
Management view	: Fine, you are only out of a job.

Software Metrics

➢ Statement	: I am only six months late with project.
Management view	: Fine, you are only out of a job.
➢ Statement	: But you cannot put reliability constraints in the contract.
Management view	: Then we may not get the contract.

Software Metrics

- Categories of Metrics

  i. **Product metrics:** describe the characteristics of the product such as size, complexity, design features, performance, efficiency, reliability, portability, etc.

  ii. **Process metrics:** describe the effectiveness and quality of the processes that produce the software product. Examples are:

     - effort required in the process
     - time to produce the product
     - effectiveness of defect removal during development
     - number of defects found during testing
     - maturity of the process

Software Metrics

ii. **Project metrics:** describe the project characteristics and execution. Examples are :

- number of software developers
- staffing pattern over the life cycle of the software
- cost and schedule
- productivity

Software Metrics

Token Count

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

Software Metrics

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

Software Metrics

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).

Software Metrics

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.

Software Metrics

- Estimated Program Length

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

$$\hat{N} = 14 \log_2 14 + 10 \log_2 10$$

$= 53.34 + 33.22 = 86.56$

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

$$N_J = Log_2(\eta_1!) + \log_2(\eta_2!)$$

Software Metrics

$$N_B = \eta_1 Log_2 \eta_2 + \eta_2 \log_2 \eta_1$$

$$N_c = \eta_1 \sqrt{\eta_1} + \eta_2 \sqrt{\eta_2}$$

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

The definitions of unique operators, unique operands, total operators and total operands are not specifically delineated.

Software Metrics

- 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.

Software Metrics

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.

Software Metrics

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.

Software Metrics

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. Some names of member elements in different structure variables are counted as unique operands.

15. All the hash directive are ignored.

Software Metrics

- Potential Volume

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

- Estimated Program Level / Difficulty

^

Halstead offered an alternate formula that estimate the program level.

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

where

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

Software Metrics

- Effort and Time

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

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

$$T = E / \beta$$

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.

Software Metrics

- Language Level

$$\lambda = L \times V^* = L^2 V$$

Using this formula, Halstead and other researchers determined the language level for various languages as shown in Table 1.

Software Metrics

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

Software Metrics

Example- 6.I

Consider the sorting program in Fig. 2 of chapter 4. List out the operators and operands and also calculate the values of software science measures like η, N, V, E, λ *etc.*

Software Metrics

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.)...

Software Metrics

;	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 of fig. 2 of chapter 4

Software Metrics

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

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

$= 14 * 3.81 + 10 * 3.32$

$= 53.34 + 33.2 = 86.45$

Software Metrics

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 $L = V^* / V$

Software Metrics

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

D = I / L

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

Estimated program level

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

Software Metrics

We may use another formula

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

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

=417 / 0.038 = 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

Software 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;
}
```

Table 3

(Contd.)...

Software Metrics

```
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;
}
```

Table 3

Software Metrics

Example- 6.2

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

Software Metrics

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.)...

Software Metrics

( )	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	—	—
$\eta_1 = 24$	$N_1 = 84$	$\eta_2 = 18$	$N_2 = 55$

Table 5

Software Metrics

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 $= \frac{2}{\eta_1} \times \frac{\eta_2}{N_2}$

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

Software Metrics

Minimal volume $V^* = 20.4417$

Effort $= V / \hat{L}$

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

= 27488.33 elementary mental discriminations.

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

= 1527.1295 seconds

= 25.452 minutes

Software Metrics

Data Structure Metrics

Program	***Data Input***	***Internal Data***	***Data Output***
Payroll	Name/ Social Security No./ Pay Rate/ Number of hours worked	Withholding rates Overtime factors Insurance premium Rates	Gross pay withholding Net pay Pay ledgers
Spreadsheet	Item Names/ Item amounts/ Relationships among items	Cell computations Sub-totals	Spreadsheet of items and totals
Software Planner	Program size/ No. of software developers on team	Model parameters Constants Coefficients	Est. project effort Est. project duration

Fig.1: Some examples of input, internal, and output data

Software Metrics

- The Amount of Data

One method for determining the amount of data is to count the number of entries in the cross-reference list.

A variable is a string of alphanumeric characters that is defined by a developer and that is used to represent some value during either compilation or execution.

Software Metrics

```
#include < stdio. h >
struct  check
   {
      float gross, tax, net;
   } pay;
   float hours, rate;
   void main ( )
      {
      while  (! feof  (stdin))
         {
            scanf("%f %f", & hours, & rate);
            pay. gross = hours * rate;
            pay. tax = 0.25 * pay. gross;
            pay. net = pay. gross - pay. tax;
            printf("%f %f %f/n", pay. gross, pay. tax, pay. net);
         }
      }
```

Fig.2: Payday program

Software Metrics

check	2				
gross	4	12	13	14	15
hours	6	11	12		
net	4	14	15		
pay	5	12	13	13	14
	14	14	15	15	15
rate	6	11	12		
tax	4	13	14	15	

Fig.3: A cross reference of program payday

Software Metrics

feof	9
stdin	10

Fig.4: Some items not counted as VARS

η_2 = VARS + unique constants + labels.

Halstead introduced a metric that he referred to as η_2 to be a count of the operands in a program – including all variables, constants, and labels. Thus,

$$\eta_2 = VARS + \text{unique constants} + \text{labels}$$

Software Metrics

```
# include < stdio. h >
struct  [check]
{
   float  [gross],  [tax],  [net];
}  [pay];
float  [hours],  [rate);
void main  ( )
{
while  (!  feof  (stdin))
   {
      scanf("% f % f", & [hour], & [rate]);
      [pay]  .  [gross]  =  [hours]  *  [rate];
      [pay]  .  [tax]  =  0.25  *  [pay]  .  [gross];
      [pay]  .  [net]  =  [pay]  .  [gross]  –  [pay]  .  [tax];
      printf("% f % f % f/n", [pay] . [gross] [pay] . [tax], [pay] . [net]);
   }
}
```

Fig.6: Program payday with operands in brackets

The Usage of Data within a Module

✓ Live Variables

Definitions :

1. A variable is live from the beginning of a procedure to the end of the procedure.
2. A variable is live at a particular statement only if it is referenced a certain number of statements before or after that statement.
3. A variable is live from its first to its last references within a procedure.

Software Metrics

```
#include < stdio. h >

void swap (int x [ ], int K)
  {
    int t;
    t = x[K];
    x[K] = x[K + 1];
    x[K + 1] = t;
  }

void main ( )
{
int i, j, last, size, continue, a[100], b[100];
scanf("% d", & size);
for (j = 1; j < = size; j + +)
scanf("%d %d", & a[j], & b[j];
last = size;
continue = 1;
```

cont.

Software Metrics

19	`while(continue)`
20	`{`
21	`  continue = 0;`
22	`  last = last-1;`
23	`  i = 1;`
24	`while (i < = last)`
25	`{`
26	`  if (a[i] > a[i + 1])`
27	`  {`
28	`    continue = 1;`
29	`    swap (a, i);`
30	`    swap (b, i);`
31	`  }`
32	`    i = i + 1;`
33	`}`
34	`}`
35	`for (j = 1; j < = size; j ++)`
36	`printf("%d %d\n", a[j], b[j]);`
37	`}`

Fig.6: Bubble sort program

Software Metrics

It is thus possible to define the average number of live variables, $\overline{(LV)}$ which is the sum of the count of live variables divided by the count of executable statements in a procedure. This is a complexity measure for data usage in a procedure or program. The live variables in the program in fig. 6 appear in fig. 7 the average live variables for this program is

$$\frac{124}{34} = 3.647$$

Software Metrics

Line	Live Variables	Count
4	----	0
5	----	0
6	t, x, k	3
7	t, x, k	3
8	t, x, k	3
9	----	0
10	----	0
11	----	0
12	----	0
13	----	0
14	size	1
15	size, j	2
16	Size, j, a, b	4

cont…

Software Metrics

Line	Live Variables	Count
17	size, j, a, b, last	5
18	size, j, a, b, last, continue	6
19	size, j, a, b, last, continue	6
20	size, j, a, b, last, continue	6
21	size, j, a, b, last, continue	6
22	size, j, a, b, last, continue	6
23	size, j, a, b, last, continue, i	7
24	size, j, a, b, last, continue, i	7
25	size, j, a, b, continue, i	6
26	size, j, a, b, continue, i	6
27	size, j, a, b, continue, i	6
28	size, j, a, b, continue, i	6
29	size, j, a, b, i	5

cont…

Software Metrics

Line	Live Variables	Count
30	size, j, a, b, i	5
31	size, j, a, b, i	5
32	size, j, a, b, i	5
33	size, j, a, b	4
34	size, j, a, b	4
35	size, j, a, b	4
36	j, a, b	3
37	--	0

Fig.7: Live variables for the program in fig.6

Software Metrics

✓ Variable spans

```
…
21          scanf (“%d %d, &a, &b)
…
32          x =a;
…
45          y = a – b;
…
53          z = a;
…
60          printf (“%d %d, a, b);
…
```

Fig.: Statements in ac program referring to variables a and b.

The size of a span indicates the number of statements that pass between successive uses of a variables

Software Metrics

- Making program-wide metrics from intra-module metrics

For example if we want to characterize the average number of live variables for a program having modules, we can use this equation.

$$\overline{LV}\, program = \frac{\sum_{i=1}^{m} \overline{LV_i}}{m}$$

where $\overline{(LV)}_i$ is the average live variable metric computed from the *i*th module

The average span size $\overline{(SP)}$ for a program of n spans could be computed by using the equation.

$$\overline{SP}\, program = \frac{\sum_{i=1}^{n} \overline{SP_i}}{n}$$

Software Metrics

- Program Weakness

A program consists of modules. Using the average number of live variables $(\overline{LV})$ and average life variables (γ), the module weakness has been defined as

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

Software Metrics

A program is normally a combination of various modules, hence program weakness can be a useful measure and is defined as:

$$WP = \frac{\left(\sum_{i=1}^{m} WM_i \right)}{m}$$

where,	WM_i	: weakness of *i*th module
	WP	: weakness of the program
	m	: number of modules in the program

Software Metrics

Example- 6.3

Consider a program for sorting and searching. The program sorts an array using selection sort and than search for an element in the sorted array. The program is given in fig. 8. Generate cross reference list for the program and also calculate $\overline{LV}$, γ and WM for the program.

Software Metrics

Solution

The given program is of 66 lines and has 11 variables. The variables are a, I, j, item, min, temp, low, high, mid, loc and option.

```
 1   /**********************************************************************/
 2   /****  PROGRAM TO SORT AN ARRAY USING SELECTION SORT & THEN SEARCH
     *******/
 3   /****  FOR AN ELEMENT IN THE SORTED ARRAY ********/
 4   /**********************************************************************/
 5
 6   #include <stdio.h>
 7   #define MAX 10
 8
 9   main ()
10   {
```

(Contd.)...

Software Metrics

```
int  a[MAX];
int  i,j,item,min,temp;
int  low=0,high,mid,loc;
char  option;

for  (i=0;i<MAX;i++)
  {
  printf("Enter  a[%d]:",i);
  scanf("%d",&a[i]);
  }
/*  selection  sort  */
  for  (i=0;i<(MAX-1);i++)
  {
  min=i;
  for  (j=i+1;  j<MAX;j++)
    {
    if  (a[min]>a[j])
```

(Contd.)...

Software Metrics

```
                {
                temp=a[min];
                a[min]=a[j];
                a[j]=temp;
               }
            }
        }
printf("\n The Sorted Array:\n");
for (i=0; i<MAX;i++)
  printf("\n a[%d]=%d",i,a[i]);
printf("\n Do you want to search any element in the array
                                                    (Y/N):");
fflush(stdin);
scanf("%c",&option);
if (toupper(option)=='Y')
    {
    printf("\n Enter the item to be searched :");
```

(Contd.)...

Software Metrics

```
scanf("%d", &item);
high=MAX;
mid=(int)(low+high)/2;
while ((low<=high)&&(item!=a[mid]))
{
  if (item>a[mid])
  low=mid+1;
  else high=mid-1;
  mid=(int) (low+high)/2;
}
if(low>high)
{
  loc=0;
  printf("\n No such item is present in the array\n");
}
```

(Contd.)...

Software Metrics

```
      if  (item==a[mid])
      {
        loc=mid;
        printf("\n The item %d is present at location %d in the sorted
        array\n",item,loc);
      }
    }
  printf("\n Sorting & Searching done");
}
```

Fig.8: Sorting & searching program

Software Metrics

Cross-Reference list of the program is given below:

a	11	18	19	27	27	29	30	30	31	37	47	49	59		
i	12	16	16	16	18	19	22	22	22	24	36	36	36	37	37
j	12	25	25	25	27	30	31								
item	12	44	47	49	59	62									
min	12	24	27	29	30										
temp	12	29	31												
low	13	46	47	50	52	54									
high	13	45	46	47	51	52	54								
mid	13	46	47	49	50	51	52	59	61						
loc	13	56	61	62											
option	14	40	41												

Live Variables per line are calculated as:

Line	Live Variables	Count
13	low	1
14	low	1
15	low	1
16	low, i	2
17	low, i	2
18	low, i, a	3
19	low, i, a	3
20	low, i, a	3
22	low, i, a	3
23	low, i, a	3
24	low, i, a, min	4
25	low, i, a, min, j	5
26	low, i, a, min, j	5

cont…

Software Metrics

Line	Live Variables	Count
27	low, i, a, min, j	5
28	low, i, a, min, j	5
29	low, i, a, min, j, temp	6
30	low, i, a, min, j, temp	6
31	low, i, a, j, temp	5
32	low, i, a	3
33	low, i, a	3
34	low, i, a	3
35	low, i, a	3
36	low, i, a	3
37	low, i, a	3
38	low, a	2
39	low, a	2

cont…

Software Metrics

Line	Live Variables	Count
40	low, a, option	3
41	low, a, option	3
42	low, a	2
43	low, a	2
44	low, a, item	3
45	low, a, item, high	4
46	low, a, item, high, mid	5
47	low, a, item, high, mid	5
48	low, a, item, high, mid	5
49	low, a, item, high, mid	5
50	low, a, item, high, mid	5
51	low, a, item, high, mid	5
52	low, a, item, high, mid	5

cont…

Software Metrics

Line	Live Variables	Count
53	low, a, item, high, mid	5
54	low, a, item, high, mid	5
55	a, item, mid	3
56	a, item, mid, loc	4
57	a, item, mid, loc	4
58	a, item, mid, loc	4
59	a, item, mid, loc	4
60	item, mid, loc	3
61	item, mid, loc	3
62	item, loc	2

cont…

Software Metrics

Line	Live Variables	Count
63		0
64		0
65		0
66		0
	Total	174

Software Metrics

Average number of live variables $(\overline{LV}) = \dfrac{\text{Sum of count of live variables}}{\text{Count of executable statements}}$

$$LV = \frac{174}{53} = 3.28$$

$$\gamma = \frac{\text{Sum of count of live variables}}{\text{Total number of variables}}$$

$$\gamma = \frac{174}{11} = 15.8$$

$$\text{Module Weakness (WM)} = \overline{\text{LV}} \times \gamma$$

$$WM = 3.28 \times 15.8 = 51.8$$

Software Metrics

- The Sharing of Data Among Modules

A program normally contains several modules and share coupling among modules. However, it may be desirable to know the amount of data being shared among the modules.

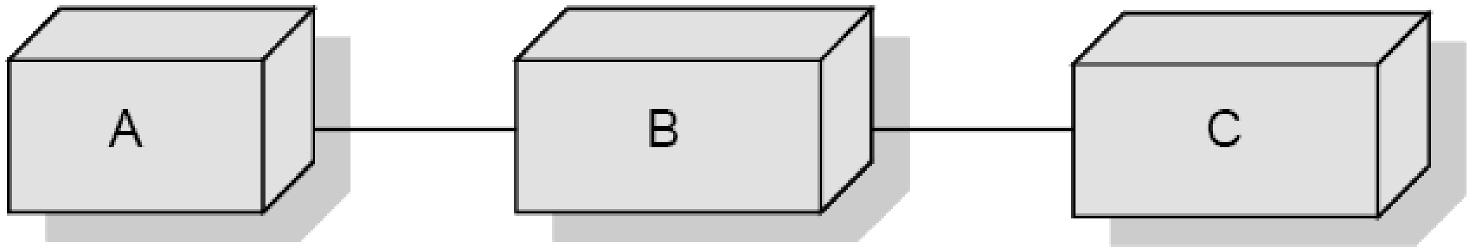

Fig.10: Three modules from an imaginary program

Software Metrics

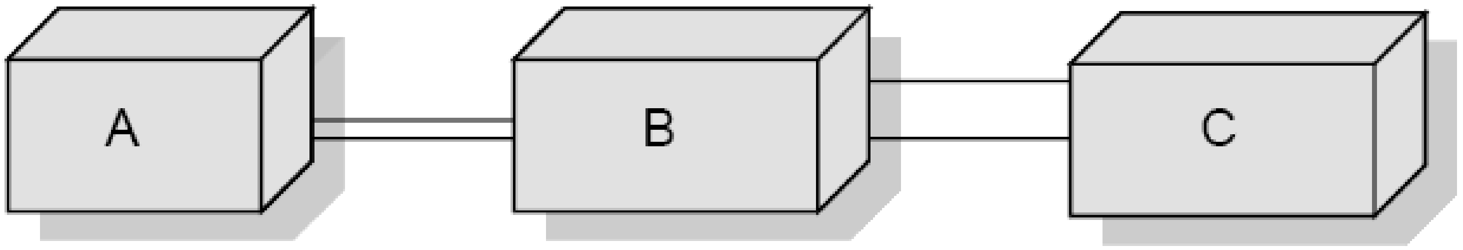

Fig.11: ”Pipes” of data shared among the modules

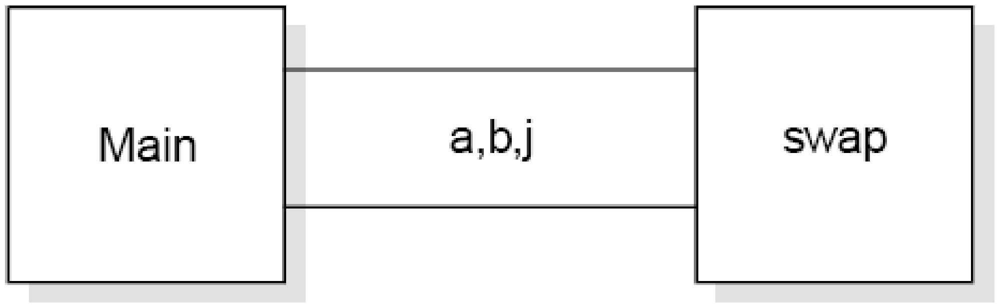

Fig.12: The data shared in program bubble

Software Metrics

Information Flow Metrics

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	: The term used to describe the degree of linkage between one component to others in the same system.

Software Metrics

- ## The Basic Information Flow Model

Information Flow metrics are applied to the Components of a system design. Fig. 13 shows a fragment of such a design, and for component 'A' we can define three measures, but remember that these are the simplest models of IF.

1. 'FAN IN' is simply a count of the number of other Components that can call, or pass control, to Component A.

2. 'FANOUT' is the number of Components that are called by Component A.

3. This is derived from the first two by using the following formula. We will call this measure the INFORMATION FLOW index of Component A, abbreviated as IF(A).

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

Software Metrics

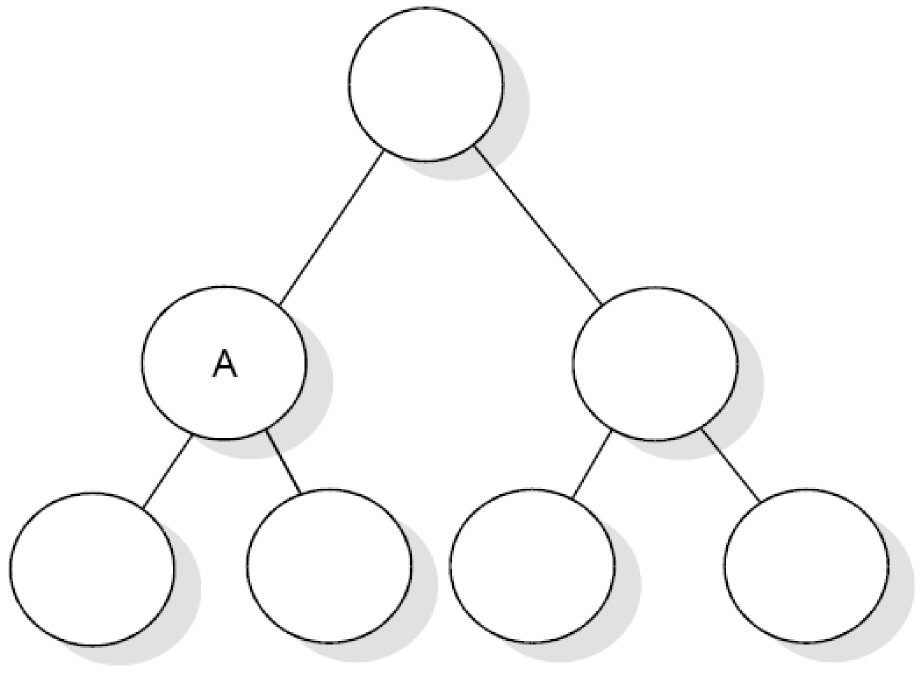

Fig.13: Aspects of complexity

Software Metrics

The following is a step-by-step guide to deriving these most simple of IF metrics.

1. Note the level of each Component in the system design.

2. For each Component, count the number of calls so that Component – this is the FAN IN of that Component. Some organizations allow more than one Component at the highest level in the design, so for Components at the highest level which should have a FAN IN of zero, assign a FAN IN of one. Also note that a simple model of FAN IN can penalize reused Components.

3. For each Component, count the number of calls from the Component. For Component that call no other, assign a FAN OUT value of one.

cont…

Software Metrics

4. Calculate the IF value for each Component using the above formula.

5. Sum the IF value for all Components within each level which is called as the LEVEL SUM.

6. Sum the IF values for the total system design which is called the SYSTEM SUM.

7. For each level, rank the Component in that level according to FAN IN, FAN OUT and IF values. Three histograms or line plots should be prepared for each level.

8. Plot the LEVEL SUM values for each level using a histogram or line plot.

Software Metrics

- A More Sophisticated Information Flow Model

a = the number of components that call A.

b = the number of parameters passed to A from components higher in the hierarchy.

c = the number of parameters passed to A from components lower in the hierarchy.

d = the number of data elements read by component A.

Then:

$$\text{FAN IN}(A) = a + b + c + d$$

Software Metrics

Also let:

e = the number of components called by A;

f = the number of parameters passed from A to components higher in the hierarchy;

g = the number of parameters passed from A to components lower in the hierarchy;

h = the number of data elements written to by A.

Then:

$$\text{FAN OUT}(A) = e + f + g + h$$

Object Oriented Metrics

Terminologies

S.No	Term	Meaning/purpose
1	Object	Object is an entity able to save a state (information) and offers a number of operations (behavior) to either examine or affect this state.
2	Message	A request that an object makes of another object to perform an operation.
3	Class	A set of objects that share a common structure and common behavior manifested by a set of methods; the set serves as a template from which object can be created.
4	Method	an operation upon an object, defined as part of the declaration of a class.
5	Attribute	Defines the structural properties of a class and unique within a class.
6	Operation	An action performed by or on an object, available to all instances of class, need not be unique.

Object Oriented Metrics

Terminologies

S.No	Term	Meaning/purpose
7	Instantiation	The process of creating an instance of the object and binding or adding the specific data.
8	Inheritance	A relationship among classes, where in an object in a class acquires characteristics from one or more other classes.
9	Cohesion	The degree to which the methods within a class are related to one another.
10	Coupling	Object A is coupled to object B, if and only if A sends a message to B.

Object Oriented Metrics

- Measuring on class level
  - coupling
  - inheritance
  - methods
  - attributes
  - cohesion
- Measuring on system level

Object Oriented Metrics

Size Metrics:

- Number of Methods per Class (NOM)

- Number of Attributes per Class (NOA)

- Weighted Number Methods in a Class (WMC)
  - Methods implemented within a class or the sum of the complexities of all methods

Object Oriented Metrics

Coupling Metrics:

- Response for a Class (RFC)
  – Number of methods (internal and external) in a class.

- Data Abstraction Coupling(DAC)
  - Number of Abstract Data Types in a class.

- Coupling between Objects (CBO)
  – Number of other classes to which it is coupled.

Object Oriented Metrics

- Message Passing Coupling (MPC)
  - Number of send statements defined in a class.

- Coupling Factor (CF)
  - Ratio of actual number of coupling in the system to the max. possible coupling.

Object Oriented Metrics

Cohesion Metrics:

- LCOM: Lack of cohesion in methods
  - Consider a class C_1 with n methods M_1, M_2…., M_n. Let (I_j) = set of all instance variables used by method M_i. There are n such sets $\{I_1\}$,…….$\{I_n\}$. Let

$$P = \{(I_i, I_j) \mid I_i \cap I_j = 0\} \text{ and } Q = \{((I_i, I_j) \mid I_i \cap I_j \neq 0\}$$

If all n $\{(I_1\},\ldots\ldots\ .(I_n)\}$ sets are 0 then P=0

$$\text{LCOM} = |P| - |Q|, \text{ if } |P| > |Q|$$

$$= 0 \text{ otherwise}$$

Object Oriented Metrics

- Tight Class Cohesion (TCC)

  _ Percentage of pairs of public methods of the class with common attribute usage.

- Loose Class Cohesion (LCC)
  - Same as TCC except that this metric also consider indirectly connected methods.
- Information based Cohesion (ICH)
  - Number of invocations of other methods of the same class, weighted by the number of parameters of the invoked method.

Object Oriented Metrics

Inheritance Metrics:

- DIT - Depth of inheritance tree
- NOC - Number of children
  - only immediate subclasses are counted.

Object Oriented Metrics

Inheritance Metrics:

- AIF- Attribute Inheritance Factor
  - Ratio of the sum of inherited attributes in all classes of the system to the total number of attributes for all classes.

$$AIF = \frac{\sum_{i=1}^{TC} A_d(C_i)}{\sum_{i=1}^{TC} A_a(C_i)}$$

$$A_a(C_i) = A_i(C_i) + A_d(C_i)$$

TC= total number of classes

Ad (Ci) = number of attribute declared in a class

Ai (Ci) = number of attribute inherited in a class

Object Oriented Metrics

Inheritance Metrics:

- MIF- Method Inheritance Factor
  - Ratio of the sum of inherited methods in all classes of the system to the total number of methods for all classes.

$$\text{MIF} = \frac{\sum_{i=1}^{TC} M_i(C_i)}{\sum_{i=1}^{TC} M_a(C_i)}$$

$$M_a(C_i) = M_i(C_i) + M_d(C_i)$$

TC= total number of classes

Md(Ci)= the number of methods declared in a class

Mi(Ci)= the number of methods inherited in a class

Use-Case Oriented Metrics

- Counting actors

Type	Description	Factor
Simple	Program interface	1
Average	Interactive or protocol driven interface	2
Complex	Graphical interface	3

Actor weighting factors

- Simple actor: represents another system with a defined interface.
- Average actor: another system that interacts through a text based interface through a protocol such as TCP/IP.
- Complex actor: person interacting through a GUI interface.

The actors weight can be calculated by adding these values together.

Use-Case Oriented Metrics

- Counting use cases

Type	Description	Factor
Simple	3 or fewer transactions	5
Average	4 to 7 transactions	10
Complex	More than 7 transactions	15

Transaction-based weighting factors

The number of each use case type is counted in the software and then each number is multiplied by a weighting factor as shown in table above.

Web Engineering Project Metrics

- ⇨ Number of static web pages
- ⇨ Number of dynamic web pages
- ⇨ Number of internal page links
- ⇨ Word count
- ⇨ Web page similarity
- ⇨ Web page search and retrieval
- ⇨ Number of static content objects
- ⇨ Number of dynamic content objects

Metrics Analysis

Statistical Techniques

- Summary statistics such as mean, median, max. and min.
- Graphical representations such as histograms, pie charts and box plots.
- Principal component analysis
- Regression and correlation analysis
- Reliability models for predicting future reliability.

Metrics Analysis

Problems with metric data:

- Normal Distribution
- Outliers
- Measurement Scale
- Multicollinearity

Metrics Analysis

Common pool of data:

- The selection of projects should be representative and not all come from a single application domain or development styles.
- No single very large project should be allowed to dominate the pool.
- For some projects, certain metrics may not have been collected.

Metrics Analysis

Pattern of Successful Applications:

- Any metric is better then none.
- Automation is essential.
- Empiricism is better then theory.
- Use multifactor rather then single metrics.
- Don’t confuse productivity metrics with complexity metrics.
- Let them mature.
- Maintain them.
- Let them die.

Multiple Choice Questions

Note: Choose most appropriate answer of the following questions:

6.1 Which one is not a category of software metrics ?
(a) Product metrics (b) Process metrics
(c) Project metrics (d) People metrics

6.2 Software science measures are developed by
(a) M.Halstead (b) B.Littlewood
(c) T.J.McCabe (d) G.Rothermal

6.3 Vocabulary of a program is defined as:
$(a)\eta = \eta_1 + \eta_2$ $(b)\eta = \eta_1 - \eta_2$
$(c)\eta = \eta_1 \times \eta_2$ $(d)\eta = \eta_1 / \eta_2$

6.4 In halstead theory of software science, volume is measured in bits. The bits are
(a) Number of bits required to store the program
(b) Actual size of a program if a uniform binary encoding scheme for vocabulary is used
(c) Number of bits required to execute the program
(d) None of the above

Multiple Choice Questions

6.5 In Halstead theory, effort is measured in
   (a) Person-months (b) Hours
   (c) Elementary mental discriminations (d) None of the above

6.6 Language level is defined as
   (a) $\lambda = L^3V$ (b) $\lambda = LV$
   (c) $\lambda = LV^*$ (d) $\lambda = L^2V$

6.7 Program weakness is
   (a) $WM = \overline{LV} \times \gamma$ (b) $WM = \overline{LV} / \gamma$
   (a) $WM = \overline{LV} + \gamma$ (d) None of the above

6.8 'FAN IN' of a component A is defined as
   (a) Count of the number of components that can call, or pass control, to component A
   (b) Number of components related to component A
   (c) Number of components dependent on component A
   (d) None of the above

Multiple Choice Questions

6.9 'FAN OUT' of a component A is defined as
- (a) number of components related to component A
- (b) number of components dependent on component A
- (c) number of components that are called by component A
- (d) none of the above

6.10 Which is not a size metric?

(a) LOC	(b) Function count
(c) Program length	(d) Cyclomatic complexity

6.11 Which one is not a measure of software science theory?

(a) Vocabulary	(b) Volume
(c) Level	(d) Logic

6.12 A human mind is capable of making how many number of elementary mental discriminations per second (i.e., stroud number)?

(a) 5 to 20	(b) 20 to 40
(c) 1 to 10	(d) 40 to 80

Multiple Choice Questions

6.13 Minimal implementation of any algorithm was given the following name by Halstead:

(a) Volume (b) Potential volume
(c) Effective volume (d) None of the above

6.14 Program volume of a software product is

(a) $V=N \log_2 n$ (b) $V=(N/2) \log_2 n$
(c) $V=2N \log_2 n$ (d) $V=N \log_2 n+1$

6.15 Which one is the international standard for size measure?

(a) LOC (b) Function count
(c) Program length (d) None of the above

6.16 Which one is not an object oriented metric?

(a) RFC (b) CBO
(c)DAC (d) OBC

Multiple Choice Questions

6.17 Which metric also consider indirect connected methods?

(a) TCC	(b) LCC
(c) Both of the above	(d) None of the above

6.18 depth of inheritance tree (DIT) can be measured by:

(a) Number of ancestors classes	(b) Number of successor classes
(c) Number of failure classes	(d) Number of root classes

6.19 A dynamic page is:

(a) where contents are not dependent on the actions of the user
(b) where contents are dependent on the actions of the user
(c) where contents cannot be displayed
(d) None of the above

6.20 Which of the following is not a size metric?

(a) LOC	(b) FP
(c) Cyclomatic Complexity	(d) program length

Exercises

6.1 Define software metrics. Why do we really need metrics in software?

6.2 Discus the areas of applications of software metrics? What are the problems during implementation of metrics in any organizations?

6.3 What are the various categories of software metrics? Discuss with the help of suitable example.

6.4 Explain the Halstead theory of software science. Is it significant in today's scenario of component based software development?

6.5 What is the importance of language level in Halstead theory of software science?

6.6 Give Halstead's software science measure for:

(i) Program Length (ii) Program volume
(iii) Program level (iv) Effort
(v) Language level

Exercises

6.7 For a program with number of unique operators $\eta_1 = 20$ and number of unique operands $\eta_2 = 40$, Compute the following:

(i) Program volume (ii) Effort and time

(iii) Program length (iv) Program level

6.8 Develop a small software tool that will perform a Halstead analysis on a programming language source code of your choice.

6.9 Write a program in C and also PASCAL for the calculation of the roots of a quadratic equation, Find out all software science metrics for both the programs. Compare the outcomes and comment on the efficiency and size of both the source codes.

6.10 How should a procedure identifier be considered, both when declared and when called/ What about the identifier of a procedure that is passed as a parameter to another procedure?

Exercises

6.11 Assume that the previous payroll program is expected to read a file containing information about all the cheques that have been printed. The file is supposed to be printed and also used by the program next time it is run, to produce a report that compares payroll expenses of the current month with those of the previous month. Compute functions points for this program. Justify the difference between the function points of this program and previous one by considering how the complexity of the program is affected by adding the requirement of interfacing with another application (in this case, itself).

6.12 Define data structure metrics. How can we calculate amount of data in a program?

6.13 Describe the concept of module weakness. Is it applicable to programs also.

6.14 Write a program for the calculation of roots of a quadratic equation. Generate cross reference list for the program and also calculate for this program.

Exercises

6.15 Show that the value of SP at a particular statement is also the value of LV at that point.

6.16 Discuss the significance of data structure metrics during testing.

6.17 What are information flow metrics? Explain the basic information flow model.

6.18 Discuss the problems with metrics data. Explain two methods for the analysis of such data.

6.19 Show why and how software metrics can improve the software process. Enumerate the effect of metrics on software productivity.

6.20 Why does lines of code (LOC) not measure software nesting and control structures?

6.21 Several researchers in software metrics concentrate on data structure to measure complexity. Is data structure a complexity or quality issue, or both?

Exercises

6.22 List the benefits and disadvantages of using Library routines rather than writing own code.

6.23 Compare software science measure and function points as measure of complexity. Which do you think more useful as a predictor of how much particular software's development will cost?

6.24 Some experimental evidence suggests that the initial size estimate for a project affects the nature and results of the project. Consider two different managers charged with developing the same application. One estimates that the size of the application will be 50,000 lines, while the other estimates that it will be 100,000 lines. Discuss how these estimates affect the project throughout its life cycle.

6.25 Which one is the most appropriate size estimation technique and why?

6.26 Discuss the object oriented metrics. What is the importance of metrics in object oriented software development ?

Exercises

6.27 Define the following: RFC, CBO, DAC, TCC, LCC & DIT.

6.28 What is the significance of use case metrics? Is it really important to design such metrics?