Estd. 2000

Software Metric and Measurement

(Rank-band: 151-200)

Topics

- Measurement
- Definition :- Measure , Measurement , Metric
- Software Project Planning
- Line of Code (LOC)
- Function Point (FP)

Measurement

- Measurement is fundamental to any engineering discipline
- Software Metrics - Broad range of measurements for computer software
- Software Process - Measurement can be applied to improve it on a continuous basis
- Software Project - Measurement can be applied in estimation, quality control, productivity assessment & project control
- Measurement can be used by software engineers in decision making.

Definitions

- **Measure** - Quantitative indication of the extent, amount, dimension, capacity or size of some attribute of a product or process

- **Measurement** - The act of determining a measure

- **Metric** - A quantitative measure of the degree to which a system, component, or process possesses a given attribute (IEEE Standard Glossary of Software Engineering Terms)

Definitions

- **Indicator** – An indicator is a metric or combination of metrics that provide insight into the software process, a software project or the product itself.

Software Project Planning

After the finalization of SRS, we would like to estimate size, cost and development time of the project. Also, in many cases, customer may like to know the cost and development time even prior to finalization of the SRS.

Software Project Planning

In order to conduct a successful software project, we must understand:

- Scope of work to be done
- The risk to be incurred
- The resources required
- The task to be accomplished
- The cost to be expended
- The schedule to be followed

Software Project Planning

Software planning begins before technical work starts, continues as the software evolves from concept to reality, and culminates only when the software is retired.

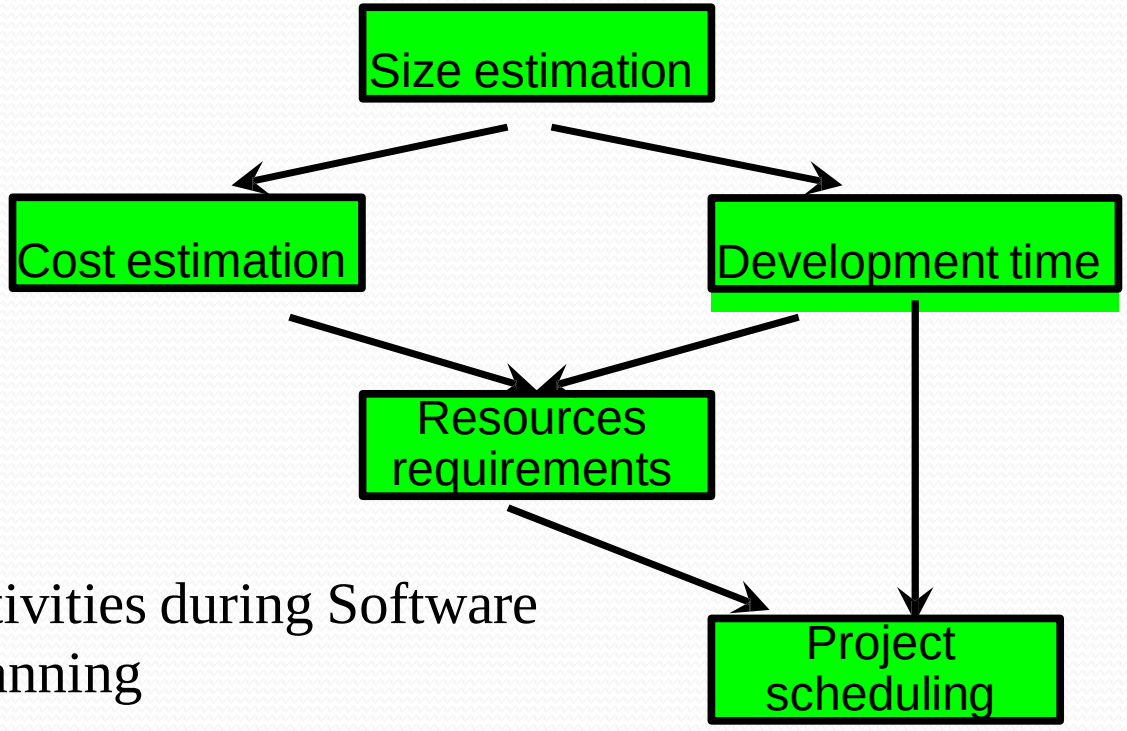


Fig. 1: Activities during Software Project Planning

Line of Code

- Size Estimation
- Lines of Code (LOC)

- If LOC is simply a count of the number of lines then figure shown below contains 18 LOC .

- When comments and blank lines are ignored, the program in figure 2 shown below contains 17 LOC.

Fig. 2: Function for sorting an array

```
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. }
```

Line of Code

Furthermore, if the main interest is the size of the program for specific functionality, it may be reasonable to include executable statements. The only executable statements in figure shown above are in lines 5-17 leading to a count of 13. The differences in the counts are 18 to 17 to 13. One can easily see the potential for major discrepancies for large programs with many comments or programs written in language that allow a large number of descriptive but non-executable statement. Conte has defined lines of code as:

Line of Code

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

This is the predominant definition for lines of code used by researchers. By this definition, figure shown above has 17 LOC.

Function Count

Function Count

Alan Albrecht while working for IBM, recognized the problem in size measurement in the 1970s, and developed a technique (which he called Function Point Analysis), which appeared to be a solution to the size measurement problem.

Function Point

The principle of Albrecht's function point analysis (FPA) is that a system is decomposed into functional units.

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

Function Points

The FPA functional units are shown in figure given below:

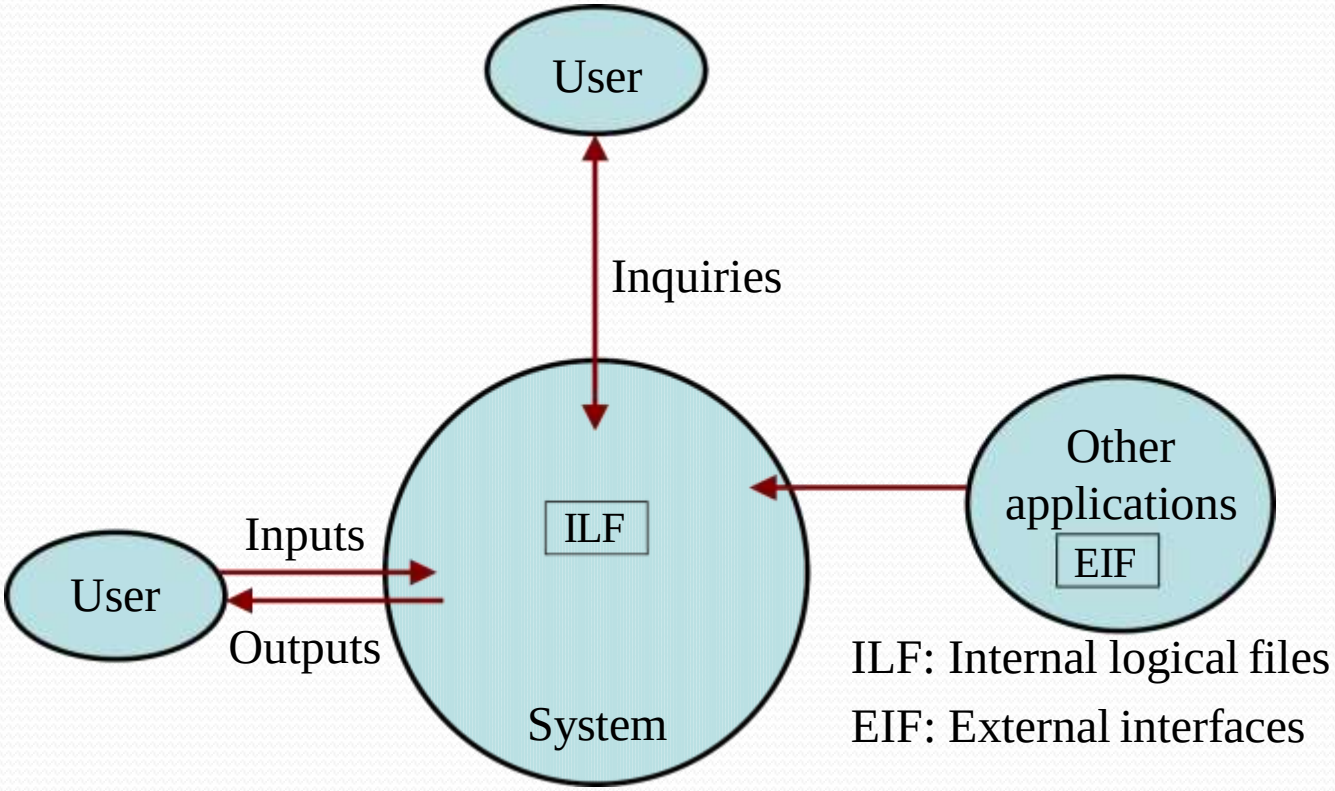


Fig. 3: FPAs functional units System

Function Points

Special features

- Function point approach is independent of the language, tools, or methodologies used for implementation; i.e. they do not take into consideration programming languages, data base management systems, processing hardware or any other data base technology.
- Function points can be estimated from requirement specification or design specification, thus making it possible to estimate development efforts in early phases of development.

Function Points

- Function points are directly linked to the statement of requirements; any change of requirements can easily be followed by a re-estimate.
- Function points are based on the system user's external view of the system, non-technical users of the software system have a better understanding of what function points are measuring.

Counting Function Points

Counting function points

Functional Units	Weighting factors		
	Low	Average	High
External Inputs (EI)	3	4	6
External Output (EO)	4	5	7
External Inquiries (EQ)	3	4	6
External logical files (ILF)	7	10	15
External Interface files (EIF)	5	7	10

Table 1 : Functional units with weighting factors

Function Counts

Table 2: UFP calculation table

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

Formula for UFP

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 indicate the row and j indicates the column of Table 1

W_{ij} : It is the entry of the i^{th} row and j^{th} column of the table 1

Zij : 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*.

Function Points

Organizations 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.

$$FP = UFP * CAF$$

Where CAF is complexity adjustment factor and is equal to [0.65 + 0.01 x ΣF_i]. The F_i (i=1 to 14) are the degree of influence and are based on responses to questions noted in table 3.

Function Points

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 Functions 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 method is also being developed.

Function Points

Table 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 ?

Example: Function Points

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.

Software ftroject ftlanning

Solution

We know

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

UFP = 50 x 4 + 40 x 5 + 35 x 4 + 6 x 10 + 4 x 7

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

CAF = $(0.65 + 0.01 \Sigma F_i)$

= (0.65 + 0.01 (14 x 3)) = 0.65 + 0.42 = 1.07

FP = UFP x CAF

= 628 x 1.07 = 672

Example

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 inquiries, and a value of complexity adjustment factor of 1.10.

What are the unadjusted and adjusted function point counts ?

Example

Solution

Unadjusted function point counts may be calculated using as:

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

= 10 x 3 + 12 x 7 + 20 x 7 + 15 + 10 + 12 x 4

= 30 + 84 +140 + 150 + 48

FP = 452

= UFP x CAF

= 452 x 1.10 = 497.2.

Example

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 high complexity

Example

(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 installation in different organizations.

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

Example…

Solution: Unadjusted function points may be counted using table 2

Functional Units	Count	Complexity		Complexity Totals	Functional Unit Totals
External Inputs (EIs)	10	Low x 3	=	30	
	15	Average x 4	=	60	
	17	High x 6	=	102	192
External Outputs (EOs)	6	Low x 4	=	24	
	0	Average x 5	=	0	
	13	High x 7	=	91	115
External Inquiries (EQs)	3	Low x 3	=	9	
	4	Average x 4	=	16	
	2	High x 6	=	12	37
External logical Files (ILFs)	0	Low x 7	=	0	
	2	Average x 10	=	20	
	1	High x 15	=	15	35
External Interface Files (EIFs)	9	Low x 5	=	45	
	0	Average x 7	=	0	
	0	High x 10	=	0	45
Total Unadjusted Function Point Count					424

Example ..

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

CAF = $(0.65 + 0.01 \times \Sigma F_i)$

= $(0.65 + 0.01 \times 41)$

= 1.06

FP = UFP x CAF

= 424 x 1.06

= 449.44

Hence FP = 449

Complexity Metrics

- Not a measure of computational complexity
- Measures psychological complexity, specifically structural complexity; that is, the complexity that arises from the structure of the software itself, independent of any cognitive issues.
- Many complexity metrics exist: H. Zuse lists over 200 in his 1990 taxonomy.
- Complexity metrics can be broadly categorized according to the fundamental software attribute measures on which they are based:
  - software science parameters
  - control-flow
  - data-flow
  - information-flow
  - hybrid

Halstead Metrics

- Software Science is generally agreed to be the beginning of systematic research on metrics as predictors for qualitative attributes of software.
- Proposed by Maurice Halstead in 1972 as a mixture of information theory, psychology, and common sense.
- These are ***linguistic metrics***.
- Based on four measures of two fundamental software attributes, operators and operands:
  - n_1 - number of unique operators
  - n_2 - number of unique operands
  - N_1 - total number of operators
  - N_2 - total number of operands

Halstead Metrics

- Halstead conjectures relationships between these fundamental measures and a variety of qualitative attributes:
  - **Length: $N = N_1 + N_2$**
  - **Estimate of N=n1 log n1 + n2 log n2**
  - **Vocabulary: $n = n_1 + n_2$**
  - **Volume: $V = N*lg(n)$**
  - **Level: $L = (2*n_2)/(n_1N_2)$**
  - **Difficulty: $D = 1/L$**
  - **Effort: $E = V * D$**
- Halstead also defines a number of other attributes:
  - potential volume, intelligence content, program purity, language level, predicted number of bugs, predicted number of seconds required for implementation

Question : Find the value of(n1,n2, N1, N2, h, *N,V, E,)*

```
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

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

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

Exercise

```
PROGRAM
Int sum,i;
Sum = 0;
For (i=1; i<=20; i++)
      sum = sum  + i;
Printf(sum);
```

Solution

List of Operators :

Operator	frequency
Int	1
,	1
;	6
=	3
()	2
<=	1
++	1
For	1
+	1
Printf	1

List of Operands :

Operands	frequency
Sum	5
i	5
0	1
1	1
20	1

Cyclomatic Complexity

- Cyclomatic Complexity, v(G), is a measure of the amount of control structure or decision logic in a program.
- Studies have shown a high correlation between v(G) and the occurrence of errors and it has become a widely accepted indicator of software quality.
- Based on the flowgraph representation of code:
  - Nodes - representing one or more procedural statements
  - Edges - the arrows represent flow of control
  - Regions - areas bounded by edges and nodes; includes the area outside the graph
- Cyclomatic Complexity is generally computed as:
  - v(G) = number of regions in the flowgraph
  - v(G) = number of conditions in the flowgraph + 1
  - v(G) = number of edges - number of nodes + 2
  - **v(G)= e-n+2**

Cyclomatic Complexity

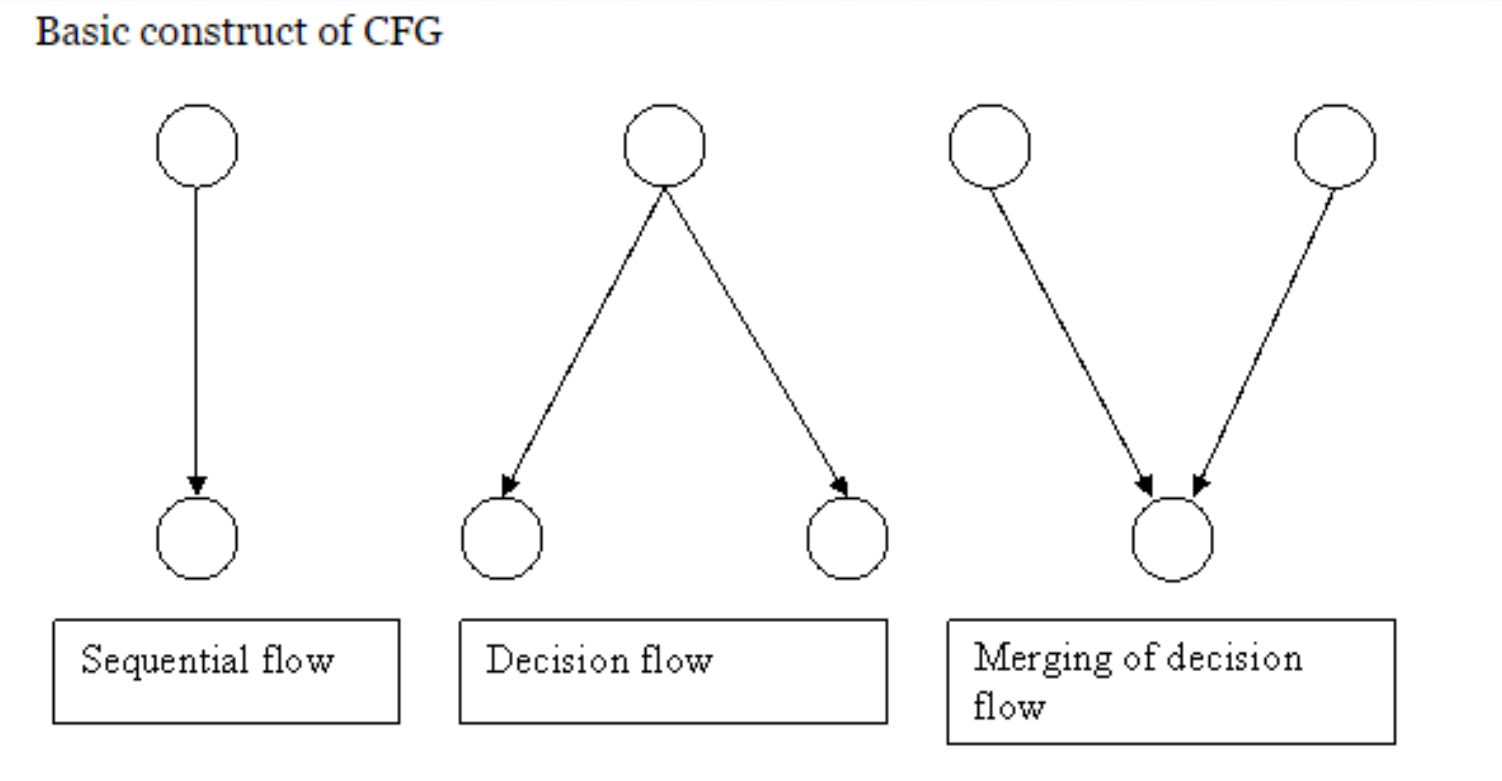
- Cyclomatic complexity can be used to
  - Determine the maximum number of test cases to ensure that all independent paths through the code have been tested.
  - Ensure the code covers all the decisions and control points in the design.
  - Determine when modularization can decrease overall complexity.
  - Determine when modules are likely to be too buggy.

Cyclomatic Complexity Thresholds

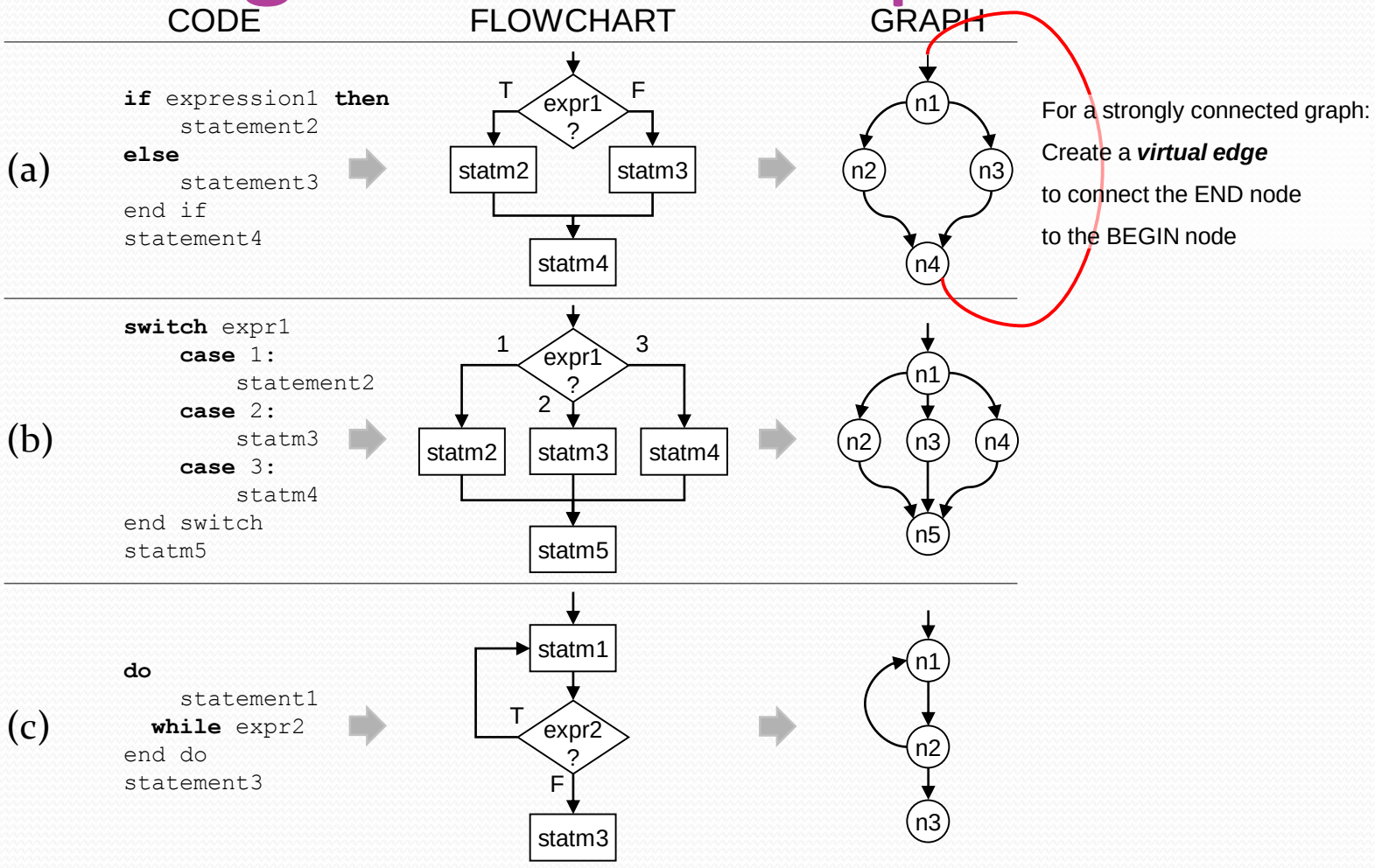
- 1-10
  - A simple module, without much risk
- 11-20
  - More complex, moderate risk
- 21-50
  - Complex, high risk
- Greater than 50
  - Untestable, very high risk

[From SEI reports]

Flow graph notation for a program



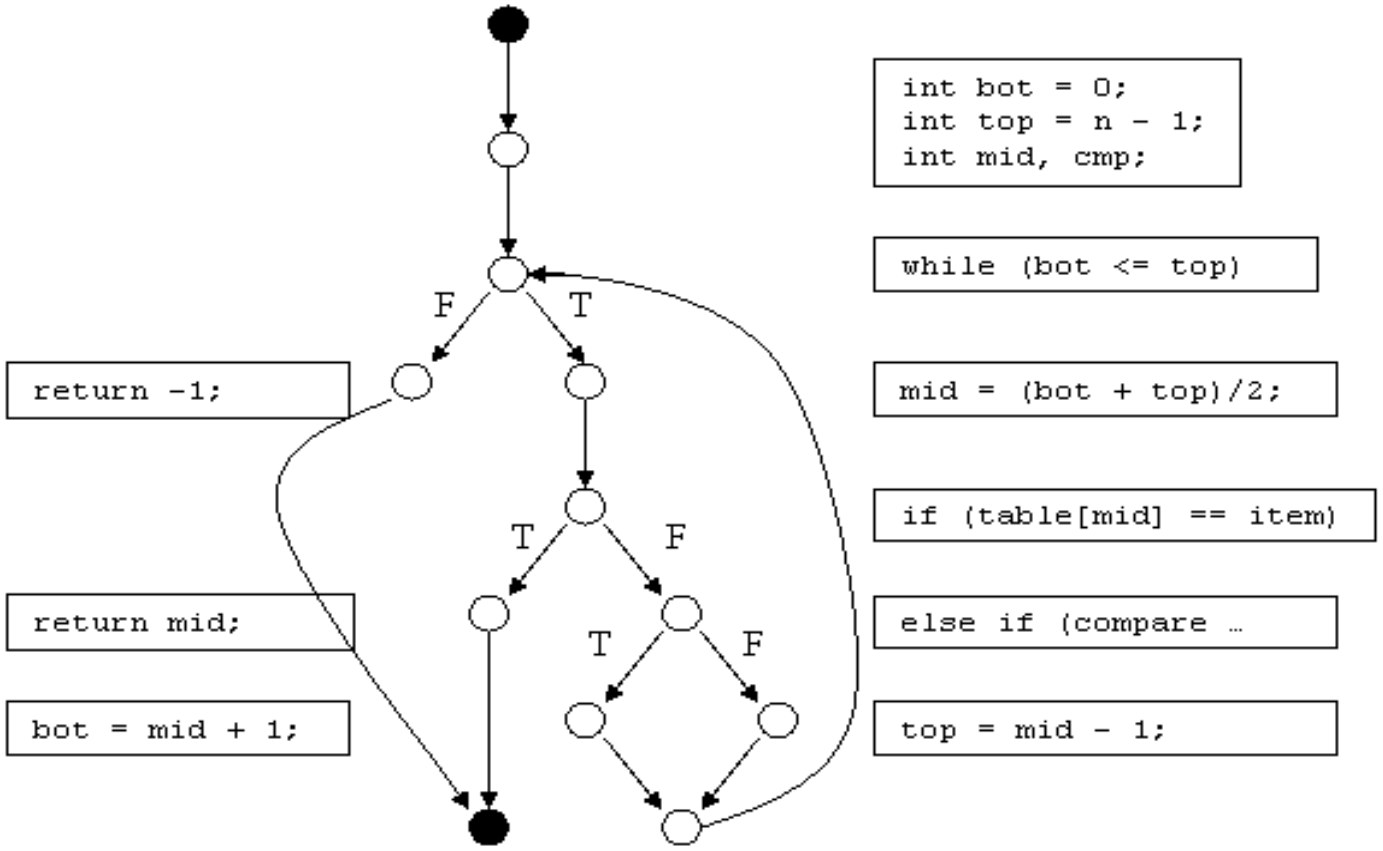
Converting Code to Graph



EXAMPLE

```
int BinSearch (char *item, char *table[], int n)
{
   int bot = 0;
   int top = n - 1;
   int mid, cmp;
   while (bot <= top) {
      mid = (bot + top) / 2;
      if (table[mid] == item)
         return mid;
      else if (compare(table[mid], item) < 0)
         top = mid - 1;
      else
         bot = mid + 1;
   }
   return -1; // not found
}
```

Solution


int bot = 0;
int top = n - 1;
int mid, cmp;
while (bot <= top)
F
T
return -1;
mid = (bot + top)/2;
T
F
if (table[mid] == item)
return mid;
else if (compare …
T
F
bot = mid + 1;
top = mid - 1;

Solution contd..

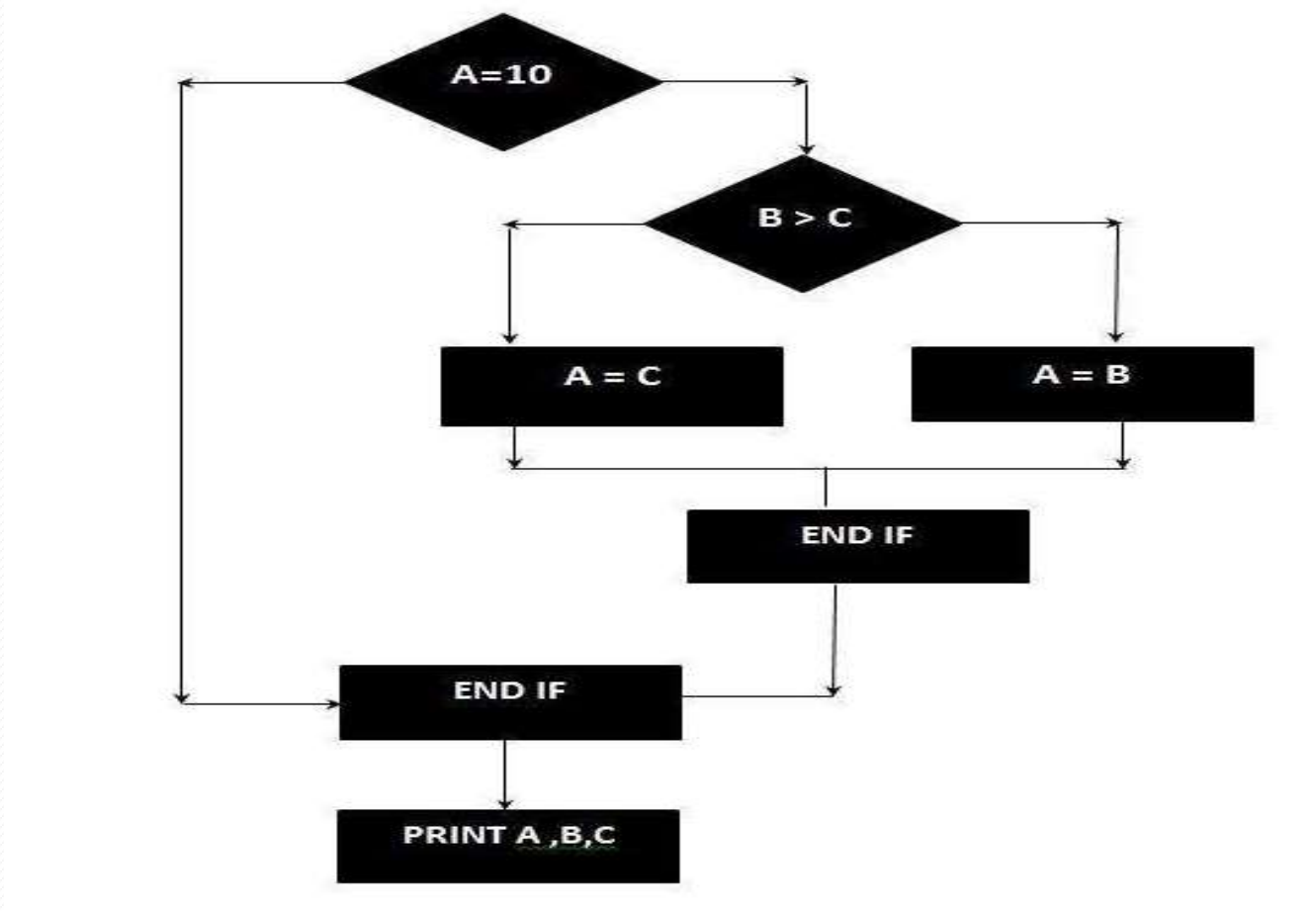
- In this example
- CC = number of edges – number of nodes + 2
- CC = 14 – 12 + 2

  = 4

Question

```
IF A = 10 THEN
 IF B > C THEN
   A = B
 ELSE
   A = C
 ENDIF
ENDIF
Print A
Print B
Print C
```

FLOWGRAPH



Question