SOFTWARE ENGINEERING

LECTURE:
HALSTEAD’S SOFTWARE METRICS

Halstead's Software Science

- Halstead's software science is an analytical technique to measure size, development effort, and development cost of software products.
- Halstead used a few primitive program parameters to develop the expressions for over all program length, potential minimum volume, actual volume, language level, effort, and development time.

Halstead's 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

Halstead’s 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

Halstead's Metrics

Counting rules for C language

1. Comments are not considered.
2. 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.

Halstead’s Metrics

Counting rules for C language

6. Functions calls are considered as operators. Function name in a function call statement is considered as an operator, and the arguments of the function call are considered as operands.
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.

Halstead’s Metrics

Counting rules for C language

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. A pair of parentheses, as well as a block begin block end pair, are considered as single operators. A sequence (statement termination) operator ’;’ is considered as a single operator.
11. GOTO is counted as an operator and the label is counted as an operand.
12. Assignment, arithmetic, and logical operators are usually counted as operators. The unary and binary occurrence of “+” and “-” are dealt separately.

Halstead's Metrics

Counting rules for C language

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.

Operators and Operands for the ANSI C language

- The following is a suggested list of operators for the ANSI C language:

  ([. , -> * + - ~ ! ++ -- * / % + - << >> < > <= >= != == & ^ | && || = *= /= %= += -= <<= >>= &= ^= |= : ? { ;

  CASE DEFAULT IF ELSE SWITCH WHILE DO FOR GOTO

  CONTINUE BREAK RETURN and a function name in a function call

- Operands are those variables and constants which are being used with operators in expressions. Note that variable names appearing in declarations are not considered as operands.

Examples

Example 1: Consider the expression a = &b; a, b are the operands and =, & are the operators.

Example 2: Consider the function call statement: func (a, b);
In this, func ' ,', ( ) a n d ; are considered as operators and variables a, b are treated as operands.

Halstead's 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).

Halstead's 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.

$$D = (n1/2) * (N2/n2)$$

- **Effort**

$$E = V / L = D * V$$

The unit of measurement of E is elementary mental discriminations.

Halstead's Metrics

- **Estimated Program Length**

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

- **Potential Minimum Volume**

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

η_2^* unique input and output parameters

Halstead's Metrics

- **Estimated Program Level / Difficulty**

Halstead offered an alternate formula that estimate the program level.

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

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

Halstead’s Metrics

- **Effort and Time**

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

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

$$T = E / \beta$$

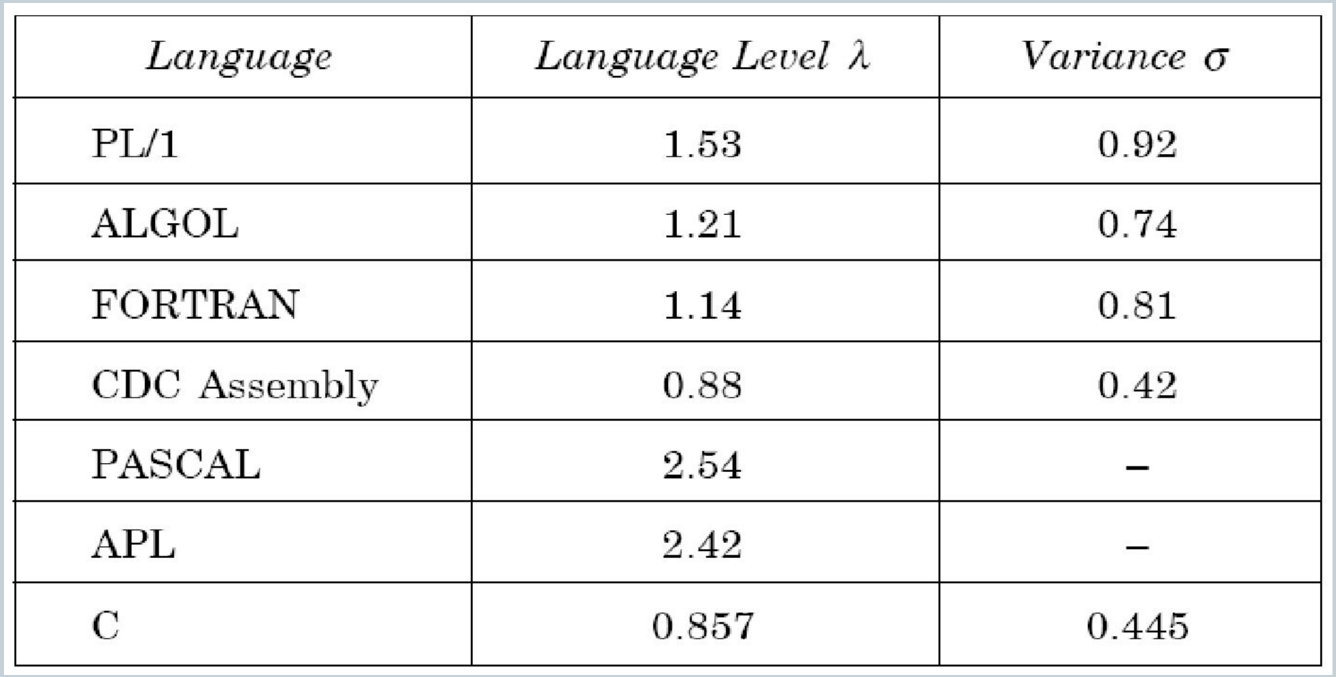
β is normally set to 18 since this seemed to give best results in Halstead’s earliest experiments, which compared the predicted times with observed programming times, including the time for design, coding, and testing.

Halstead's 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 below.



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

Exercise

```
void sort( int *a, int n )
{
int i, j, t;
if ( n <2 )
return;
for( i=0 ; i < n-1; i++)
{
for( j=i+ 1 ; j < n ; j++)
 {
if ( a[i] > a[ j] )
{
t = a[ i] ;
a[ i] =a[ j] ;
a[ j] = t;
}}}}
```

Sol

- Ignore the function definition
- Count operators and operands

	Total	Unique
Operators	N1 = 43	n1 = 15
Operands	N2 = 31	n2 = 8

McCabe’s Complexity Measures

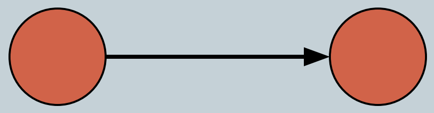
McCabe's Complexity Measures

- A software metric used to measure the complexity of software
- Developed by Thomas McCabe
- Described (informally) as the number of simple decision points + 1

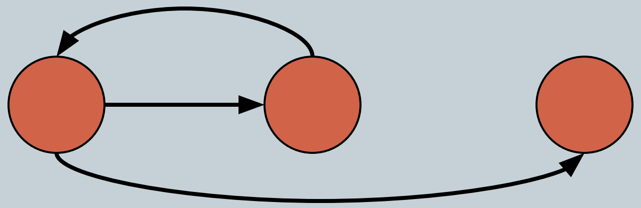
- McCabe's metrics are based on a control flow representation of the program.
- A program graph is used to depict control flow.
- Nodes represent processing tasks (one or more code statements)
- Edges represent control flow between nodes

Flow Graph Notation

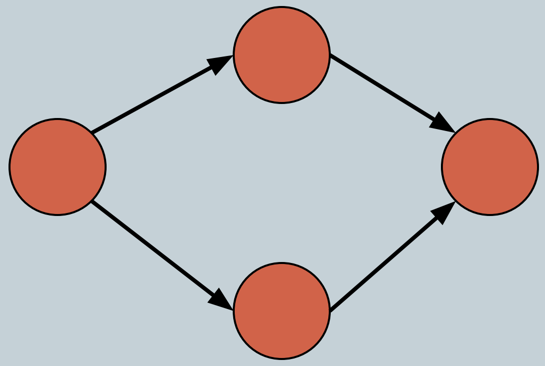
Sequence



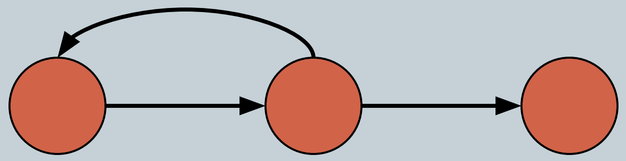
While



If-then-else



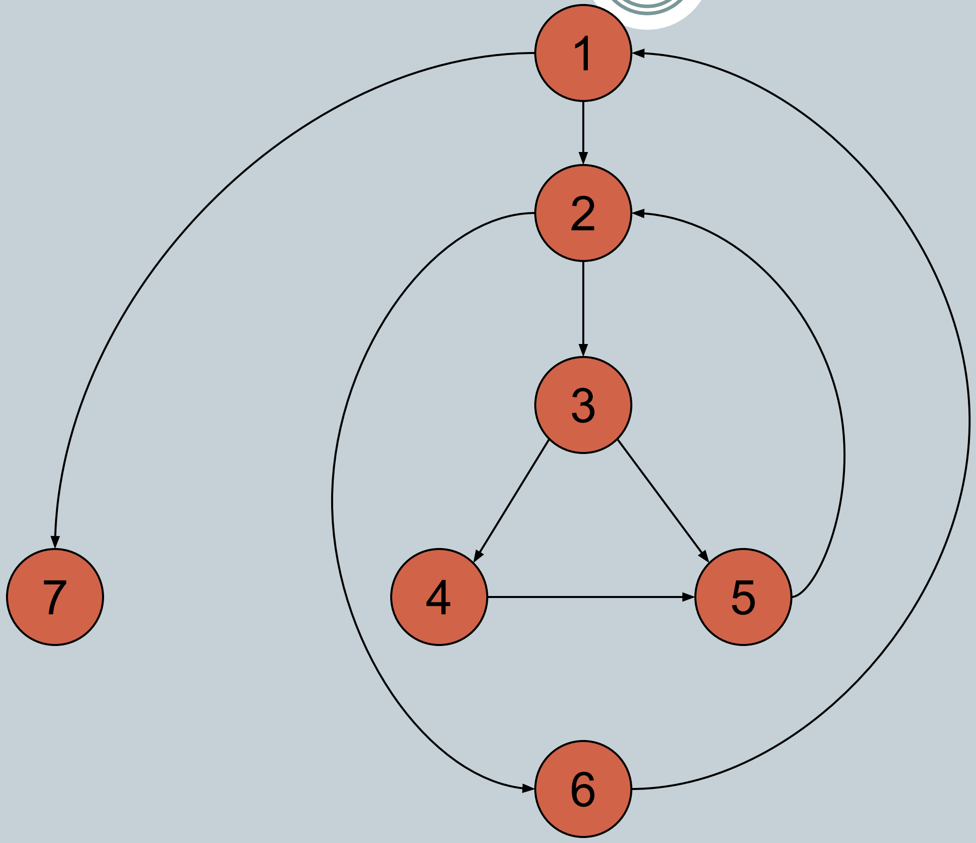
Until



Steps to calculate complexity

- Draw the control flow graph of the code
- Count the number of edges = E
- Count the number of nodes = N
- Count the number of connected components = P
- **Complexity = E – N + 2P**

Example Flow Graph

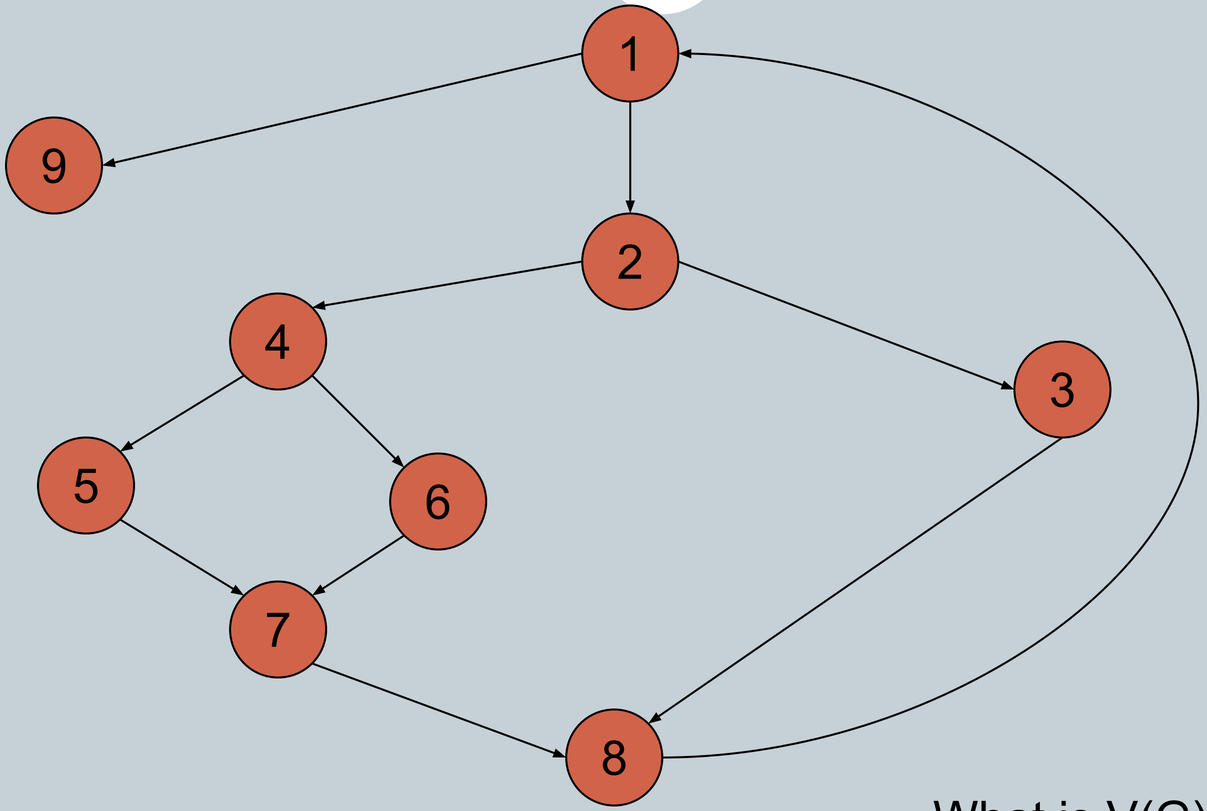


Computing V(G)

Complexity = E – N + 2P

- V(G) = 9 – 7 + 2 = 4
- V(G) = 3 + 1 = 4
- Basis Set
  - 1, 7
  - 1, 2, 6, 1, 7
  - 1, 2, 3, 4, 5, 2, 6, 1, 7
  - 1, 2, 3, 5, 2, 6, 1, 7

Another Example

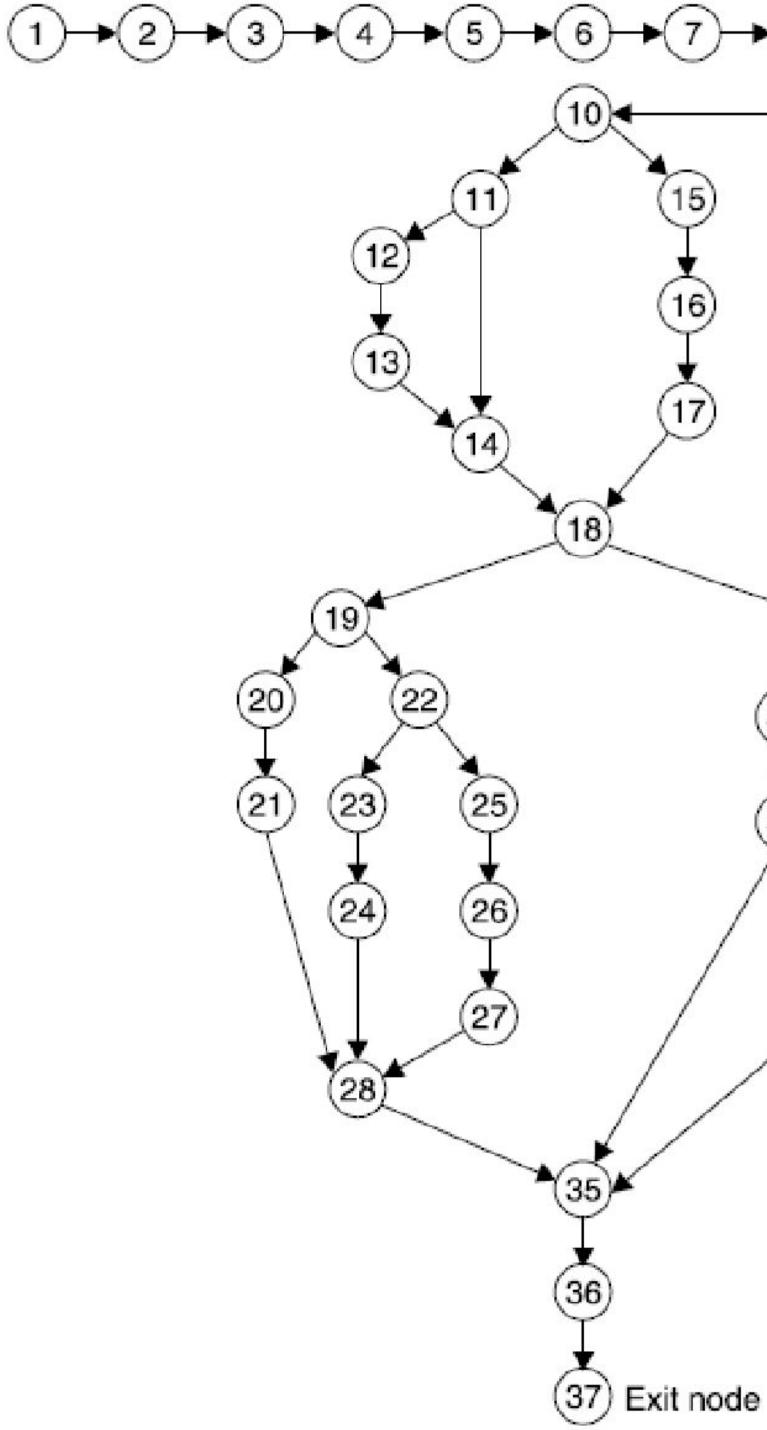


What is V(G)?

```
#include <stdio.h>
#include <conio.h>
1    int main()
2    {
3       int a,b,c,validInput=0;
4       printf(“Enter the side ‘a’ value: ”);
5       scanf(“%d”,&a);
6       printf(“Enter the side ‘b’ value: ”)
7       scanf(“%d”,&b);
8       printf(“Enter the side ‘c’ value:”);
9       scanf(“%d”,&c);
10      if ((a > 0) && (a <= 100) && (b > 0) && (b <= 100) && (c > 0)
            && (c <= 100)) {
11        if ( (a + b) > c) && ((c + a) > b) && ((b + c) > a)) {
12          validInput = 1;
13        }
14      }
15      else {
16        validInput = -1;
17      }
18      If (validInput==1) {
19        If ((a==b) && (b==c)) {
20          printf(“The trinagle is equilateral”);
21        }
22        else if ( (a == b) || (b == c) || (c == a) ) {
23          printf(“The triangle is isosceles”);
24        }
25        else {
26          printf(“The trinagle is scalene”);
27        }
28      }
29      else if (validInput == 0) {
30        printf(“The values do not constitute a Triangle”)
31      }
32      else {
33        printf(“The inputs belong to invalid range”);
34      }
35      getche();
36      return 1;
37    }
```

Draw the flow graph

Compute complexity


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
Exit node

Meaning

- V(G) is the number of (enclosed) regions/areas of the planar graph
- Number of regions increases with the number of decision paths and loops
- A quantitative measure of testing difficulty and an indication of ultimate reliability
- Experimental data shows value of V(G) should be no more then 10 - testing is very difficult above this value