Software Metrics

Lecture 7

Structure metrics (1)

Yuming Zhou

Contents

- **Control flow complexity**
- **Data flow complexity**
- **Data structure complexity**

Structural Measurement: Types

- **Control-flow structure:** Sequence of execution of instructions of the program
- **Data flow structure:** Keeping track of data as it is created or handled by the program
- **Data structure:** The organization of data itself independent of the program

Questions & Answer…

- Q1: How to represent “structure” of a program?
- A1: Control-flow diagram

- Q2: How to define “complexity” in terms of the structure?
- A2: Cyclomatic complexity

Section 1

Control flow complexity

- **Control-flow structure**
- **Cyclomatic complexity**
- **Essential complexity**

Basic Control Structure /1

- **Basic Control Structures (BCSs)** are set of essential control-flow mechanisms used for building the logical structure of the program.

- BCS types:
  - **Sequence:** e.g., a list of instructions with no other BCSs involved.
  - **Selection:** e.g., *if … then … else*.
  - **Iteration:** e.g., *do … while* ; *for … to … do*.

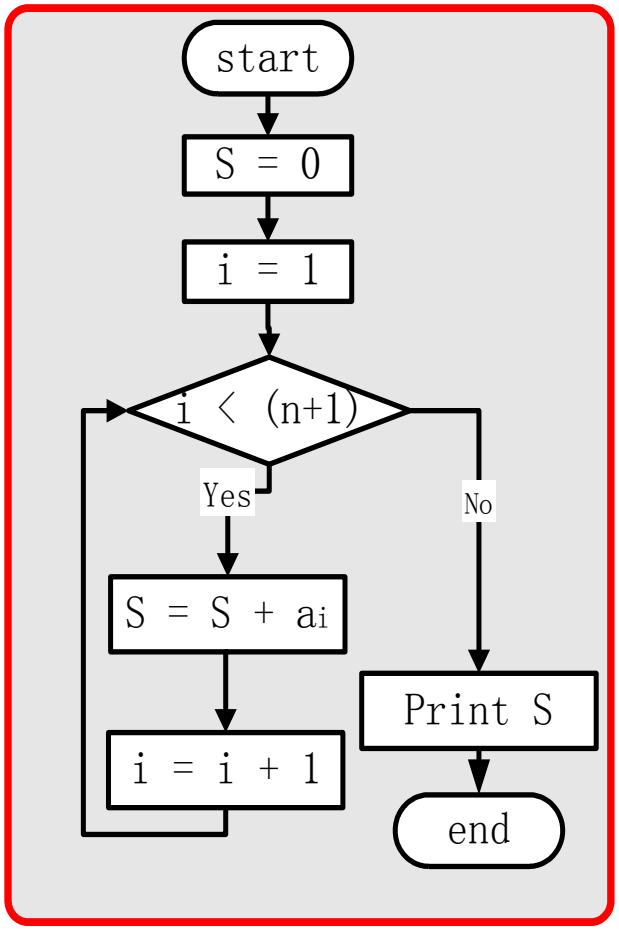
Basic Control Structure /2

- There are other types of BCSs, (may be called advanced BCSs), such as:
  - **Procedure/function/agent call**
  - **Recursion (self-call)**
  - **Interrupt**

Control Flow Graph /1(P281)

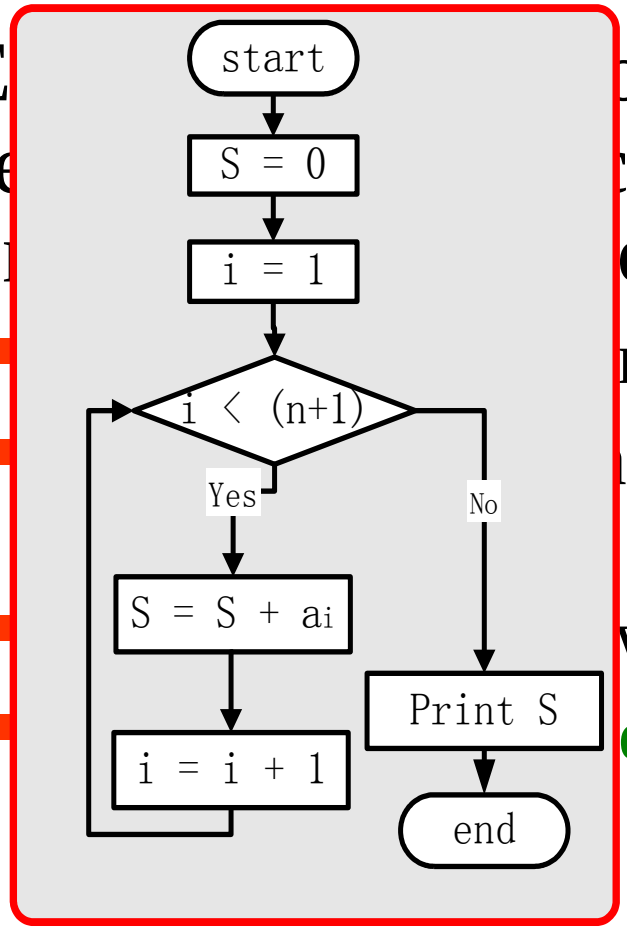
```
void sum()
{
    S=0;
    i=1;
    while (i<n+1) {
       S=S+a[i];
       i=i+1;
    }
  printf("S is %d", s);
}
```

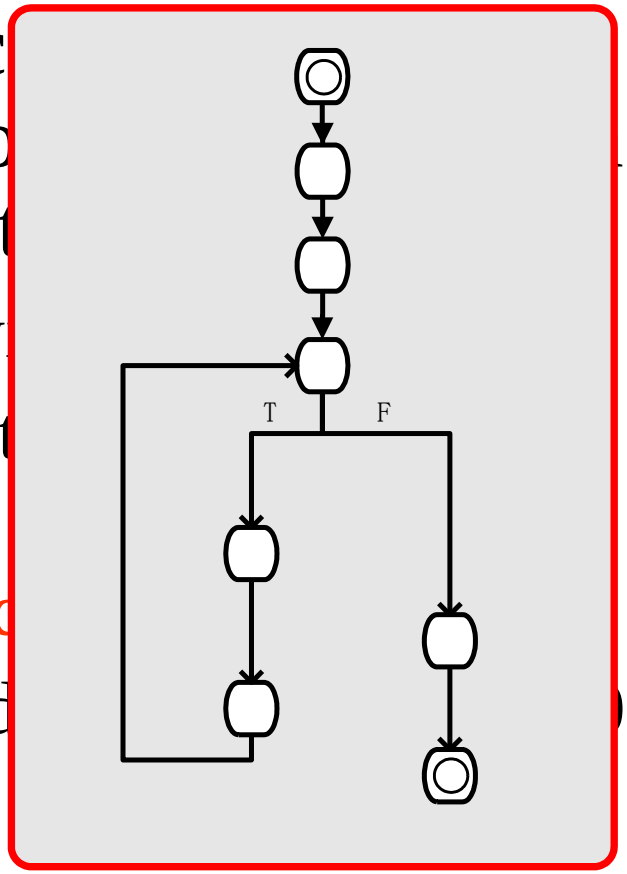
e is usu
li-graph
, S_{entry},
t of noc
ram sta



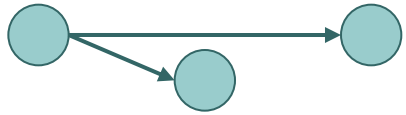
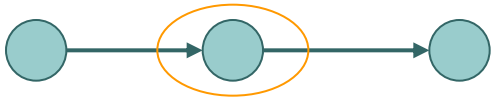
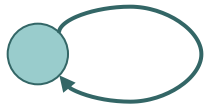
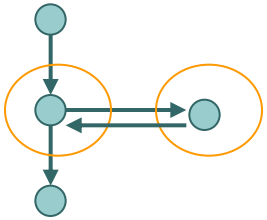
Control Flow Graph /2

- E… or direc… se… ates flo… or… ogram t…
  - … nodes w…
  - … no… wit…
  - … with in-d…
  - **des:** nod…





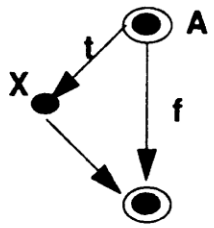
Control Flow Graph /3

Sequence	
Selection	
Iteration	
Procedure/ function call	
Recursion	
Interrupt	

Common Control Constructs

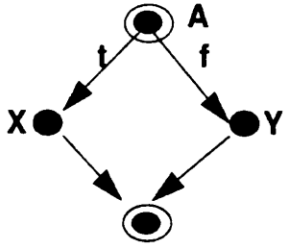
P_n or $P_n(X1,X2,...,Xn)$

X1;X2;...;Xn



D_0 or $D_0(A,X)$

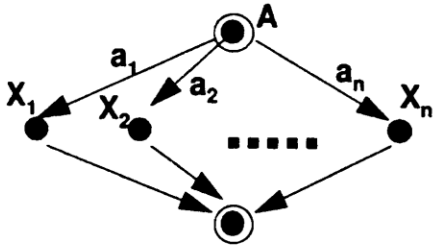
If A then X



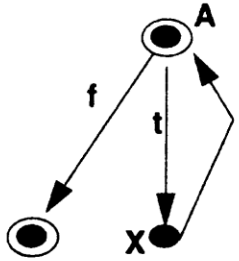
D_1 or $D_1(A,X,Y)$

If A then X
else Y

C_n or $C_n(A,X_1,...,X_n)$

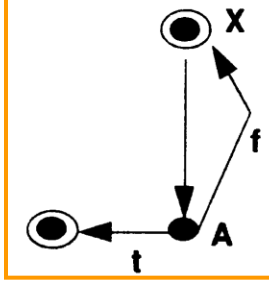


case A of
a1 : X1
a2 : X2
....
an : Xn



D_2 or $D_2(A,X)$

while A do X



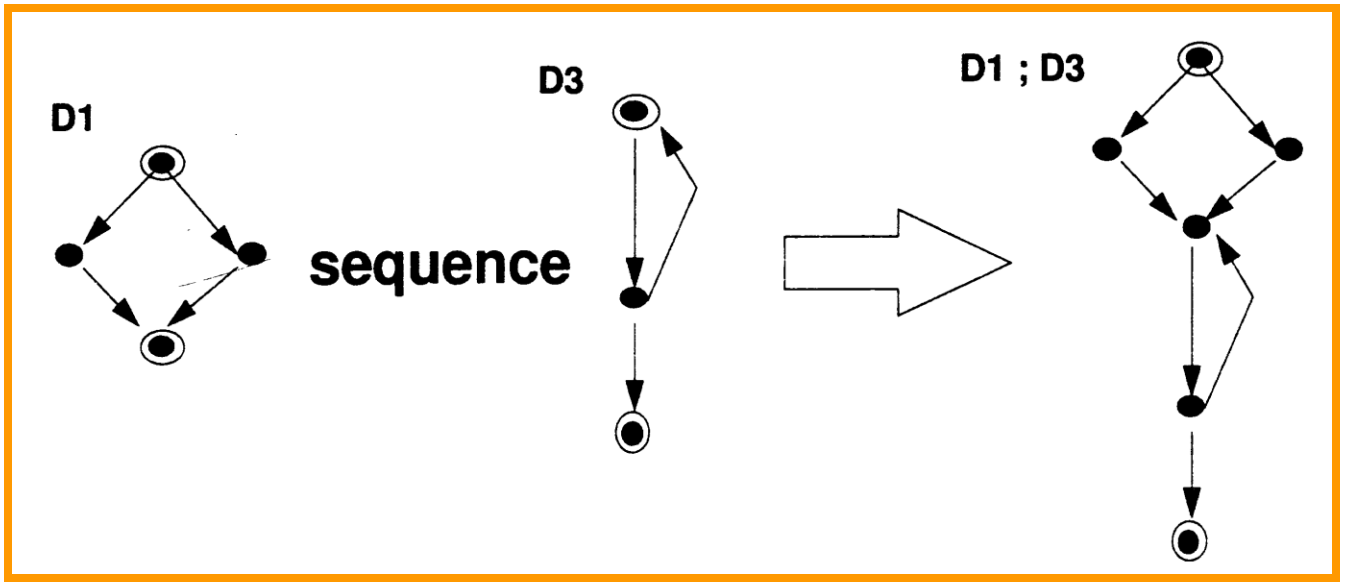
D_3 or $D_3(A,X)$

repeat X until A

Sequencing & Nesting /1

- The sequencing of **F1** and **F2**

F1; F2 Seq(**F1; F2**) P_2(**F1; F2**)



Sequencing & Nesting /2

- The nesting of **F2** onto **F1** at *x*

$$F(F_1 \text{ on } x_1, F_2 \text{ on } x_2, \ldots, F_n \text{ on } x_n)$$

$$F(F_1, F_2, \ldots, F_n)$$

Prime Flowgraphs

- Prime flowgraphs are flowgraphs that cannot be decomposed non-trivially by sequencing and nesting
- Any flowgraph can be uniquely decomposed into a hierarchy of sequencing and nesting primes

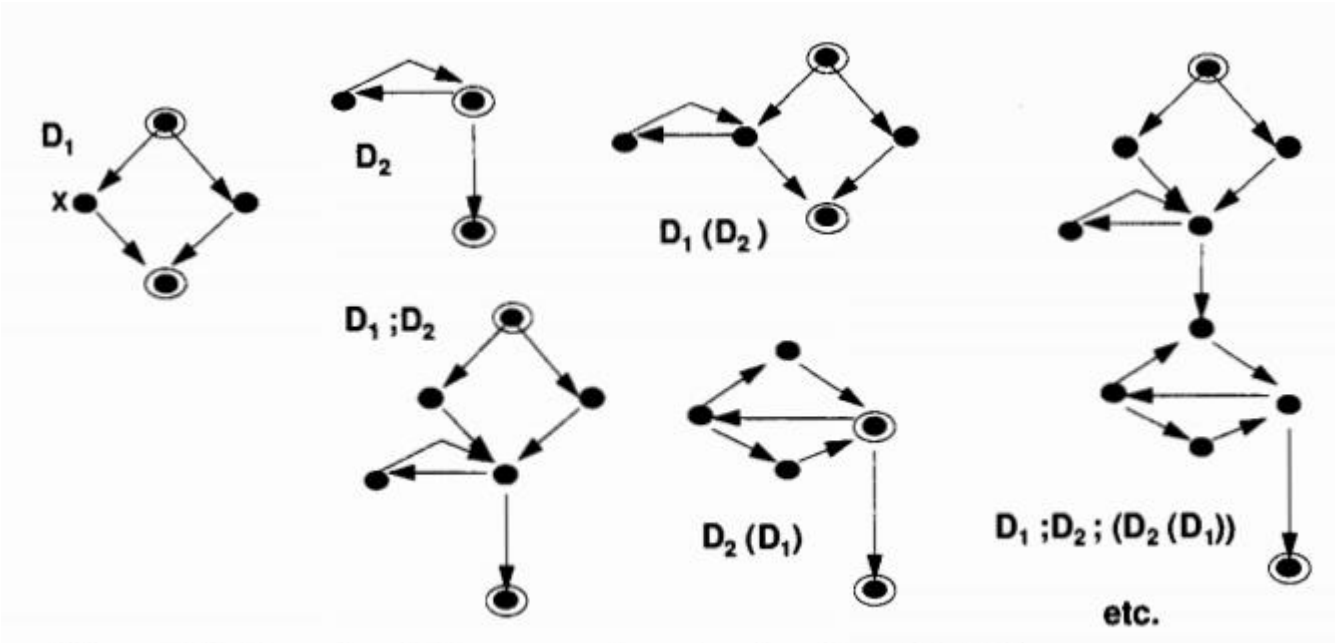
- **Examples:**
  - P_n
  - D_0, D_1, D_2 and D_3
  - C_n

S-structured Graph

- A family of flowgraphs is called **S-structured graph** (or **S-graph**) if it satisfies
  1) Each member of S is S-structured
  2) If F and G are S-structured flowgraphs, so is the sequences F;G and nesting of F(G)
  3) No flowgraph is S-structured unless it can be generated by finite number of application of the above (step 2) rules

Examples

For S = $\{P_1\}$, the set of S-structured graphs is $\{P_1, P_2, \ldots, P_n, \ldots\}$



$S = \{D_1, D_2\}$

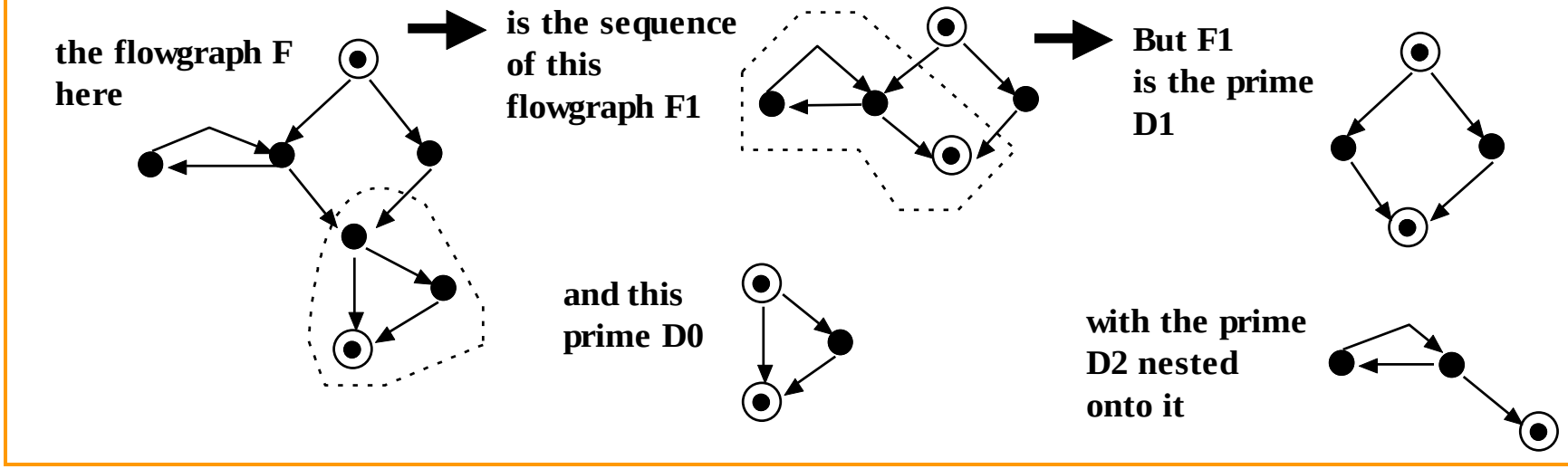
$S^D = \{P_1, D_0, D_2\}$ **D-Structured**

任意一个程序都可以用P1, D0, D2来构造

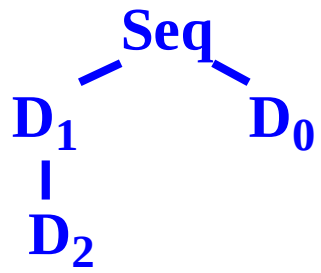
$S^D = \{P_1, D_0, D_1, D_2, D_3\}$

常规意义下的结构化

Decomposition Tree



Thus, the hierarchical decomposition into primes is

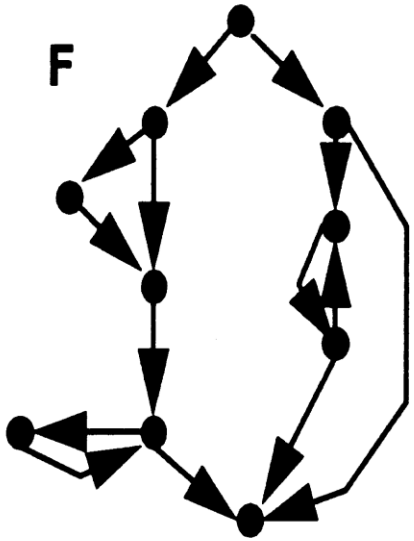


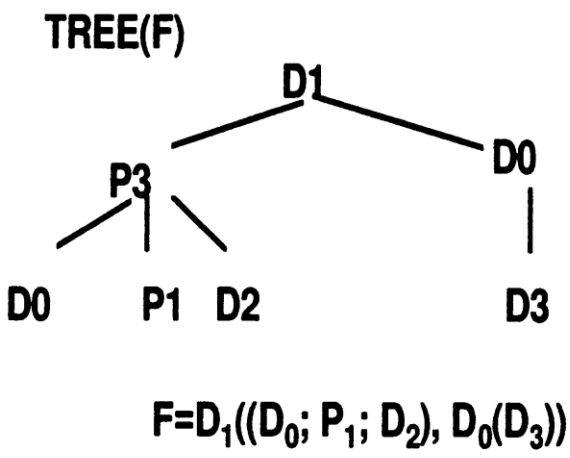
i.e. $F = (D_1(D_2));D_0$

Prime Decomposition

- Any flowgraph can be *uniquely* decomposed into a hierarchy of sequencing and nesting primes, called "**decomposition tree**"

```
if a
   then
     begin
       If b then do X;
       Y;
       while e do U
     end
   else
       if c
         then do
            repeat V until d
```





Hierarchical Metrics

- The decomposition tree is enough to measure a number of program characteristics, including:
  - Nesting factor (depth of nesting)
  - Structural complexity
  - etc.

Depth of Nesting /1

Depth of nesting $n(F)$ for a flowgraph F can be expressed in terms of:

- **Primes:**

  $n(P_1) = 0$;

  $n(D_0) = n(D_1) = n(D_2) = n(D_3) = 1$

- **Sequences:**

  $n(F_1;F_2;\ldots;F_k) = max(n(F_1), n(F_2), \ldots, n(F_k))$

- **Nesting:**

  $n(F(F_1,F_2,\ldots,F_k)) = 1 + max(n(F_1), n(F_2), \ldots, n(F_k))$

Depth of Nesting /2

Example:

$F = D_1((D_0;P_1;D_2), D_0(D_3))$

$n(F)=n(D_1((D_0;P_1;D_2), D_0(D_3)))$

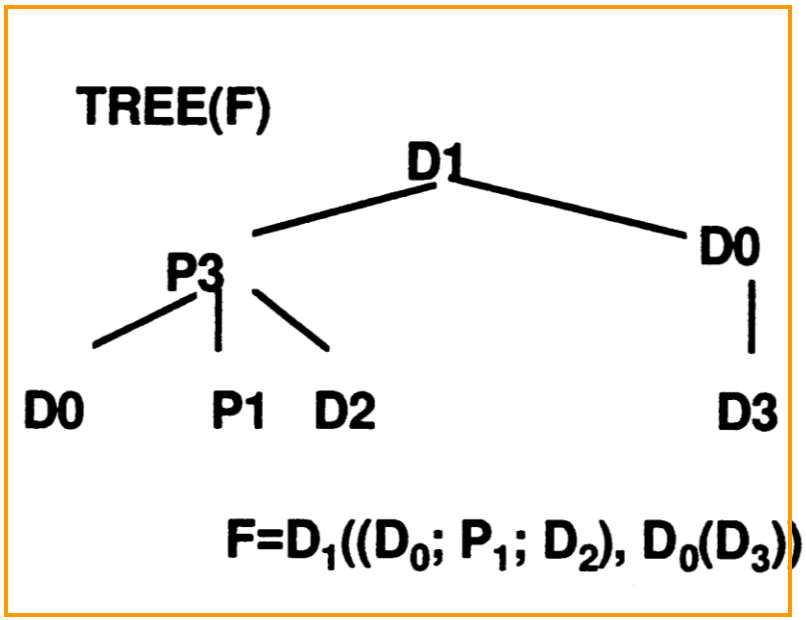
$=1+\max(n(D_0;P_1;D_2),n(D_0(D_3))$

$=1+\max(\max(n(D_0), n(P_1),$

$n(D_2)), 1+n(D_3))$

$= 1+\max(\max(1,0,1),2)$

$=1+\max(1,2)=3$



Number of statements /1

The number of statements of a program can be expressed in terms of :

过程性节点的数目

- **Primes:**

  $v(P_1) = 1$ and for each $F \neq P_1$, $v(F) = n+1$

- **Sequences:**

  $v(F_1;F_2;\ldots;F_k) = \sum v(F_i)$

- **Nesting:**

  $v(F(F_1,F_2,\ldots,F_k)) = 1 + \sum v(F_i)$ for each prime $F \neq P_1$

Number of statements /2

Example:

$V(F) = v(D_1((D_0;P_1;D_2), D_0(D_3)))$

$= 1 + v(D_0;P_1;D_2) + v(D_0(D_3))$

$= 1 + (v(D_0) + v(P_1) + v(D_2)) + (1 + v(D_3))$

$= 1 + (2 + 1 + 2) + (1 + 2)$

$= 9$

```
if a
   then
     begin
       If b then do X;
       Y;
       while e do U
     end
   else
      if c
        then do
           repeat V until d
```

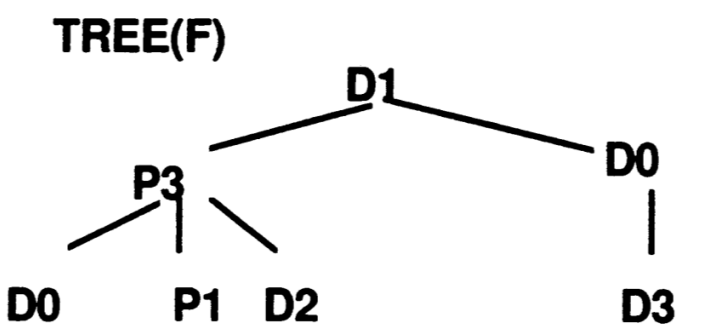


Table 8.1 Some hierarchical measures

(P292)

Number of nodes measure n

M_1: $m(F)$ = number of nodes in F for each prime F

$$M_2 : n(F_1;...F_k) = \sum n(F_i) - k + 1$$

$$M_3 : n(F(F_1,..., F_k)) = n(F) + \sum n(F_i) - 2k$$ for each prime F

Number of edges measure e

M_1: $e(F)$ = number of edges in F for each prime F

$$M_2 : e(F_1;...F_n) = \sum e(F_i)$$

$$M_3 : e(F(F_1,..., F_n)) = e(F) + \sum e(F_i) - n$$ for each prime F

The 'largest prime' measure *k*:

M_1: $k(F)$ = number of predicates in F for each prime F

$$M_2 : k(F_1;...F_n) = \max(k(F_1),..., k(F_n))$$

$$M_3 : k(F(F_1,..., F_n)) = \max(k(F), k(F_1),..., k(F_n))$$ for each prime F

. . . . . .

Section 1

Control flow complexity

- Control-flow structure
- Cyclomatic complexity
- Essential complexity

Cyclomatic Complexity

- A program's complexity can be measured by the cyclomatic number of the program flowgraph
- The cyclomatic number can be calculated in 2 different ways:
  - Flowgraph-based
  - Code-based

Cyclomatic Complexity /1

- For a program with the program flowgraph *G*, the cyclomatic complexity *v(G)* is measured as:

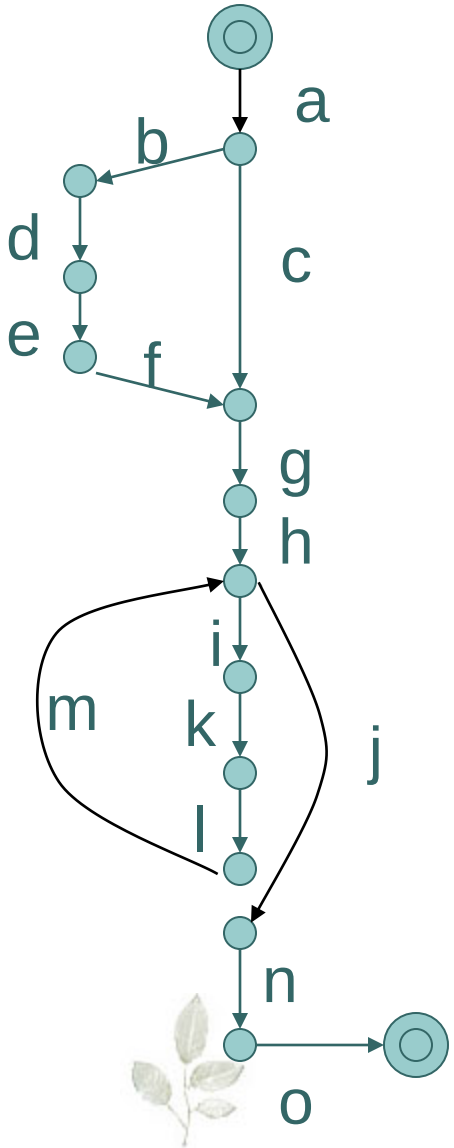
$$v(G) = e - n + 2p$$

  - *e* : number of edges
    - Representing branches and cycles
  - *n* : number of nodes
    - Representing block of sequential code
  - *p* : number of connected components
    - For a single component, *p=1*

Several properties

- v(G) ≥ 1.
- v(G) is the maximum number of linearly independent paths in G; it is the size of a basis set.
- Inserting or deleting functional statements to G does not affect v(G).
- G has only one path if and only if v(G) = 1.
- Inserting a new edge in G increases v(G) by unity
- v(G) depends only on the decision structure of G

Cyclomatic Complexity /2



Path B1: <a, c, g, h, j, n, o>

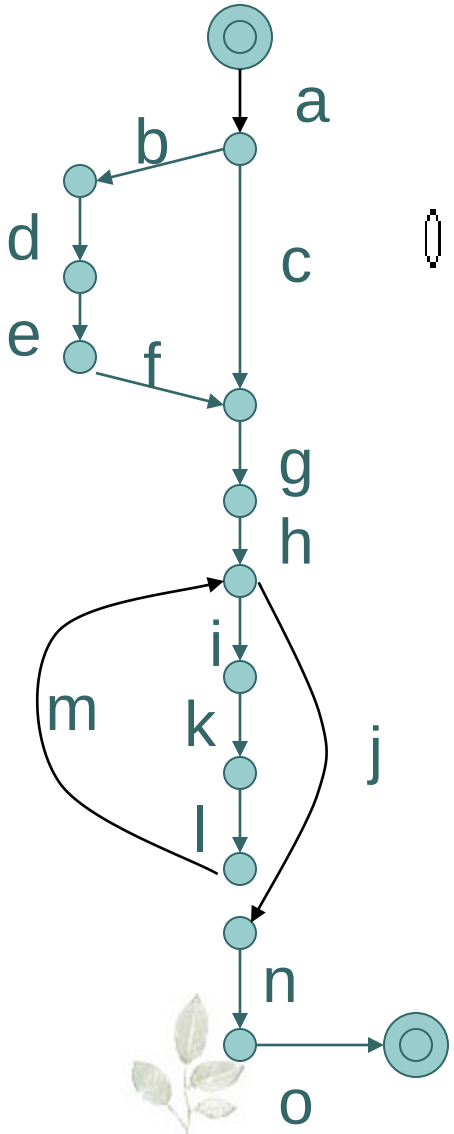
Path B2: <a, b, d, e, f, g, h, j, n, o>

Path B3: <a, c, g, h, i, k, l, m, j, n, o>

Path P: B2-2*B1+2*B3

Path/Edge	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
B1	1	0	1	0	0	0	1	1	0	1	0	0	0	1	1
B2	1	1	0	1	1	1	1	1	0	1	0	0	0	1	1
B3	1	0	1	0	0	0	1	1	1	1	1	1	1	1	1
P	1	1	0	1	1	1	1	1	2	1	2	2	2	1	1

Cyclomatic Complexity /3



- Not every vector has a corresponding path
- Linear combinations of vectors that corresponding to actual paths may be vectors that do not correspond to actual paths
- There are a potentially infinite number of basis sets of paths for a given module

Simplified Complexity Calculation/1

- For a program with the program flowgraph *G*, the cyclomatic complexity *v(G)* is measured as:

  $$v(G) = 1 + d$$

  - *d* : number of predicate nodes (i.e., nodes with out-degree other than 1)
    - d represents number of loops in the graph

      or number of decision points in the program

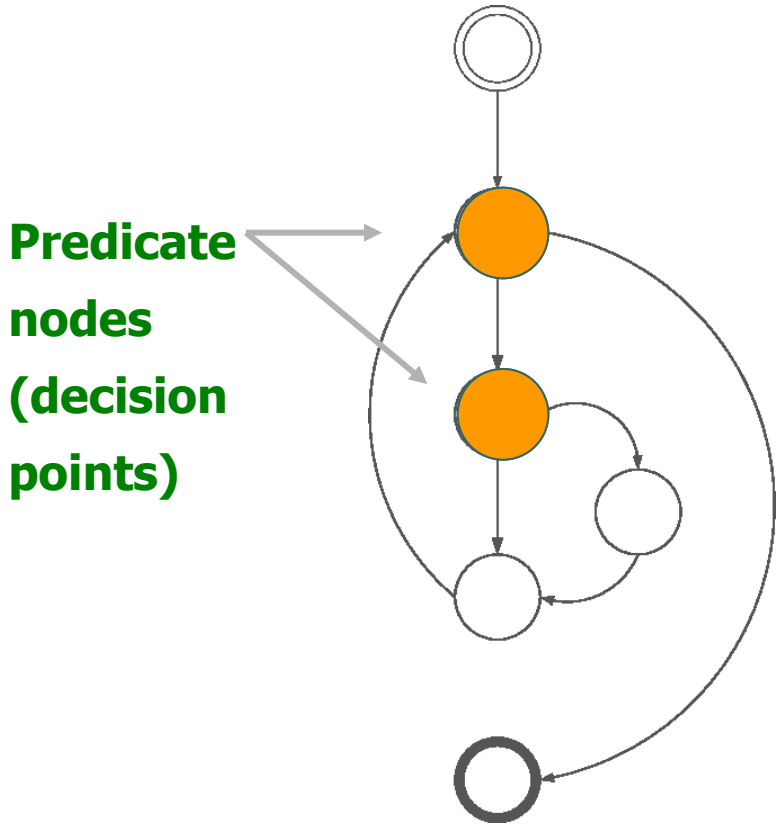
i.e., The complexity of primes depends only on the predicates (decision points or BCSs) in them

Example1:simplified calculation

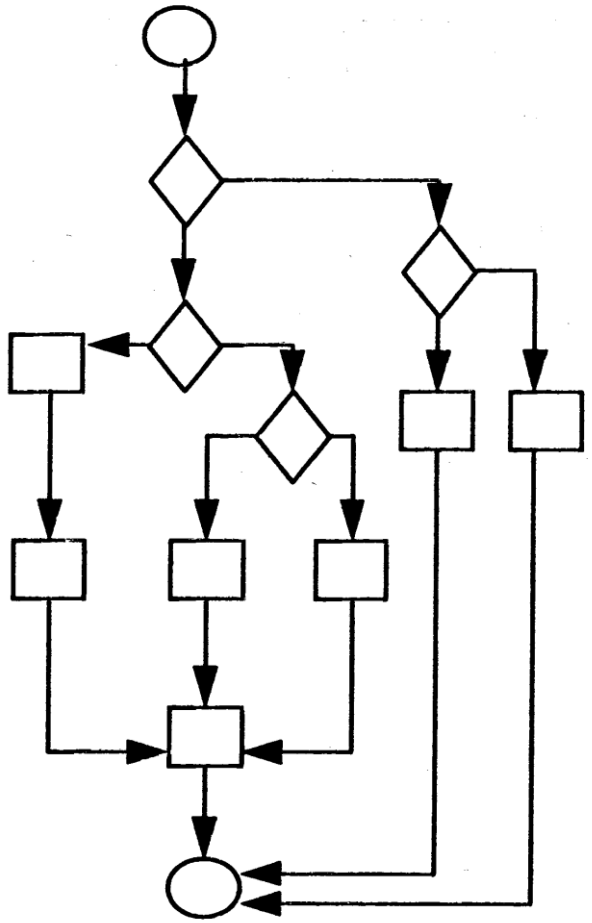
$$v(G) = e - n + 2p$$
$$= 7 - 6 + 2 \times 1$$
$$= 3$$

Or

$$v(G) = 1 + d$$
$$= 1 + 2 = 3$$



Example2:simplified calculation



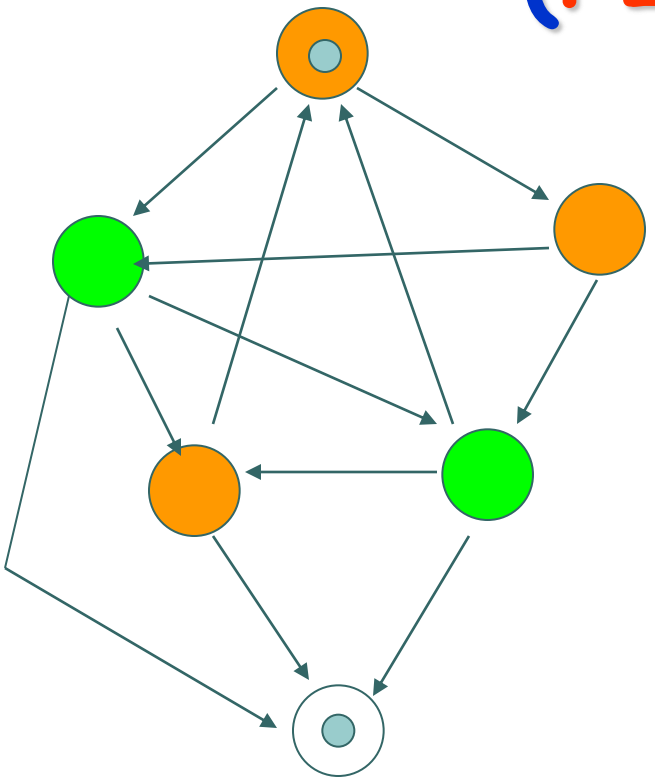
$$v(G) = 16 - 13 + 2 = 5$$

or

$$v(G) = 4 + 1 = 5$$

Example3:simplified calculation

(P295)



$$v(G) = 12\text{-}6+2$$

$$=8$$

$$v(G) = 1 + 3*1 + 2*2$$

$$= 8$$

Example 4: Code Based

```
#include <stdio.h>
main()
{
int a ;
scanf (“%d”, &a);
if (a >= 10)
   if (a < 20) printf ("10<a< 20 %d\n" , a);
   else        printf ("a >= 20 %d\n" , a);
else      printf ("a <= 10%d\n" , a);
}
```

$$v(G) = 1+2 = 3$$

Example 5: Code Based

```
Complexity(int n)
{
    if (n>0) {
        switch (n) {
            case    0:    printf("dd");
  break;
            case    1:    printf("eee");
  break;
            case    3:    printf("fff");
  break;
        }
     }
}
```

$$v(G) = 1+1 + 3 =5$$

Simplified Complexity Calculation/2

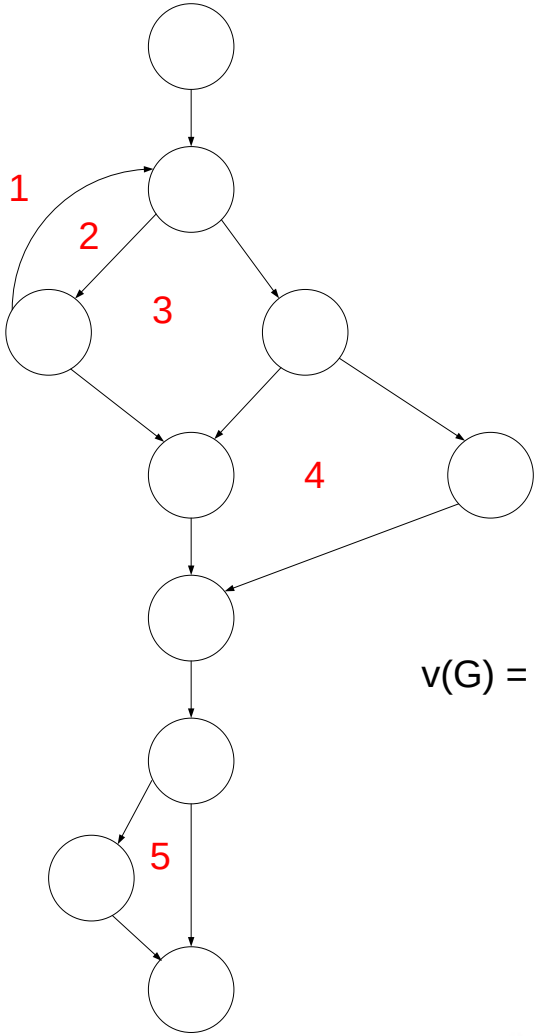
- The second way is by using Euler's formula:
  - If G is a connected plane graph with n vertices, e edges, and r regions, then

$$n - e + r = 2$$

  - So **the number of regions** is equal to the cyclomatic complexity.

Simplification

G:



v(G) = 5

Complexity: Basic Questions

1. What is the complexity of the primes?

The complexity of primes is the number of predicates (decision points or BCSs) plus one.

$$v(G) = 1 + d$$

2. What is the complexity of sequencing?

Complexity of a sequence is equal to the sum of the complexities of the components minus the number of components plus one.

$$v(F_1; F_2; \ldots; F_n) = \sum_{i=1}^{n} v(F_i) - n + 1$$

Complexity: Basic Questions

3. What is the complexity of nesting onto a given prime?

Complexity of nesting components on a prime F is equal to the complexity of F plus the sum of complexities of the components minus the number of components.

$$v\left(F\left(F_1; F_2; \ldots; F_n\right)\right) = v\left(F\right) + \sum_{i=1}^{n} v\left(F_i\right) - n$$

Cyclomatic Complexity: Critics

- **Advantages:**
  - Objective measurement of complexity
- **Disadvantages:**
  - Can only be used at the component level
  - Two programs having the same cyclomatic complexity number may need different programming effort
  - Requires complete design or code visibility

Section 1

Control flow complexity

- Control-flow structure
- Cyclomatic complexity
- Essential complexity

McCabe's Essential Complexity

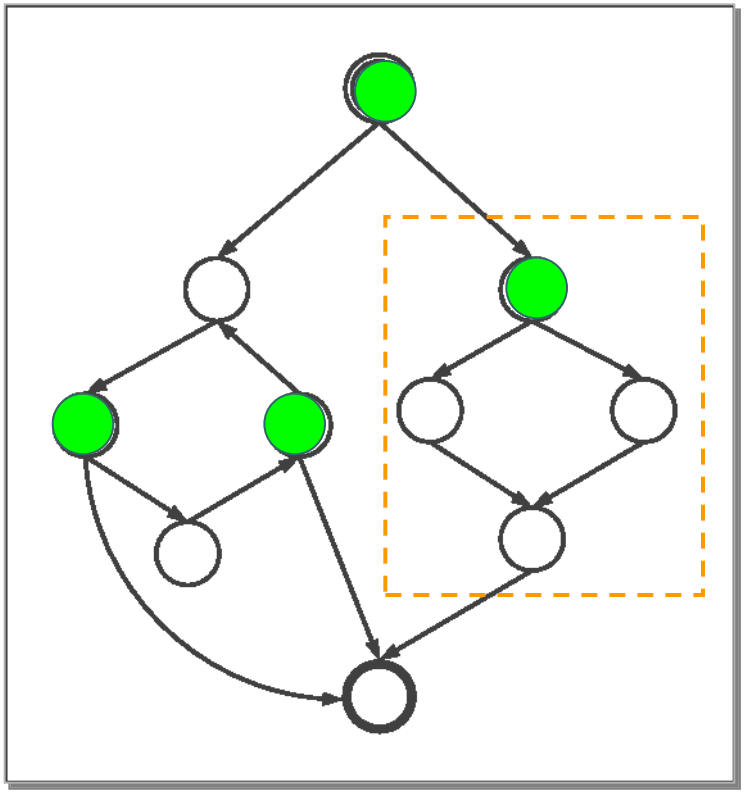
Essential complexity of a program with flow graph G is given by:

$ev(G) = v(G) - m$

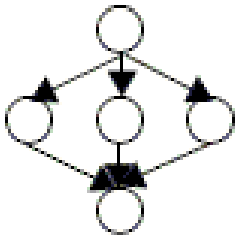
- m is the number of D_0, D_1, D_2 and D_3 sub-flowgraphs of G
- Example:

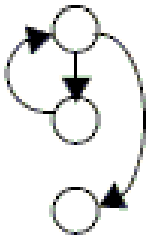
  $v(G) = 5$

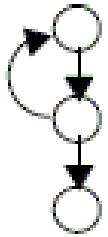
  $ev(G) = 5 - 1 = 4$



Example: Essential Complexity



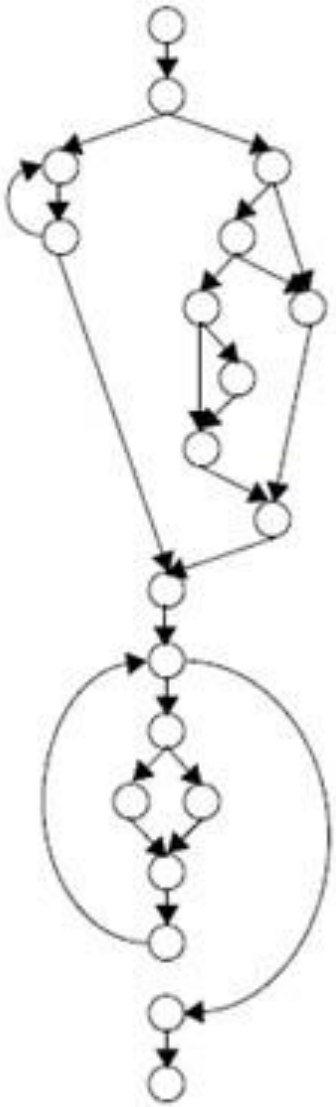




"if" "case" "while" "repeat"

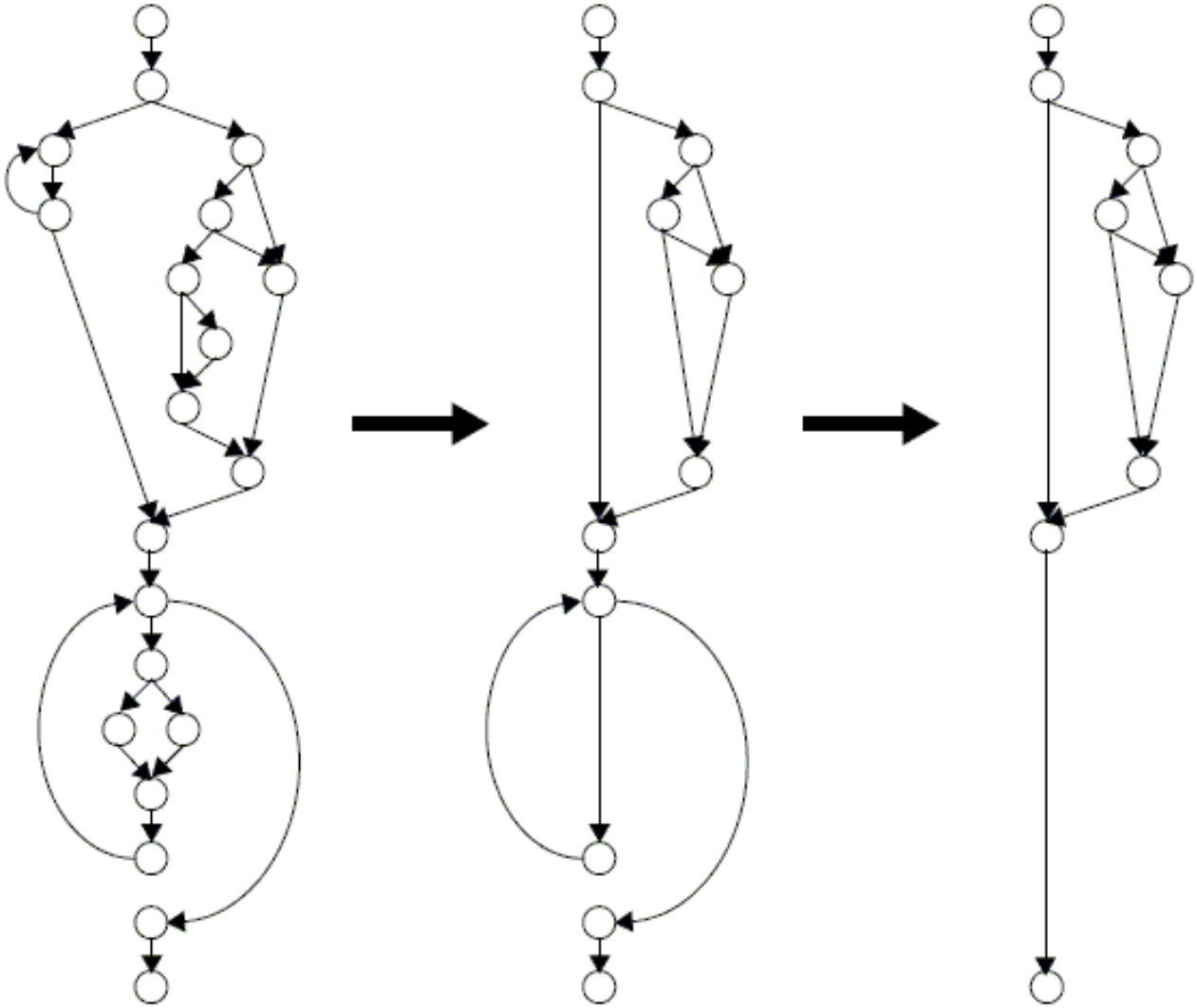
Sequence **Selection** **Iteration**

Example: Essential Complexity



ev(G) = ?

Example: Essential Complexity



v(G) = 8
ev(G) = 4

v(G) = 4

McCabe_IQ

http://www.mccabe.com/

Software Quality Metrics to Identify Risk

补充

The Relationship of Cyclomatic Complexity, Essential Complexity and Error Rates

Mike Chapman and Dan Solomon

chapman@ivv.nasa.gov

solomon@ivv.nasa.gov

2002

Hypothesis

Essential complexity is a better predictor of error than cyclomatic complexity.

Complexity Definitions

- Cyclomatic Complexity, or v(G), measures the number of linearly independent paths through a given program.
- Essential complexity is the measure of the degree to which a module contains unstructured constructs.
- Module design complexity, iv(G), is a measure of the module's decision structure as it relates to calls to other modules.

Lines of Code

- McCabe finds the open bracket and starts counting until it finds the close bracket. This metric was used to show LOC error prediction in this study.

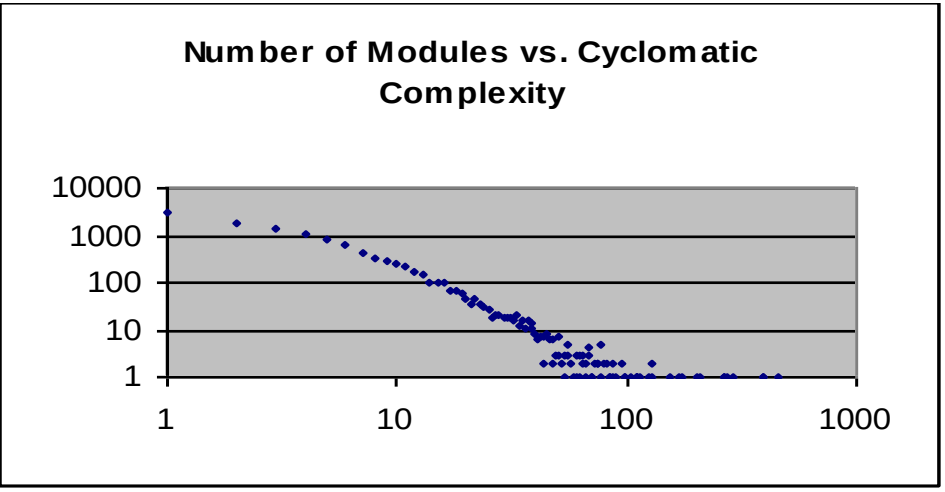
The Dataset

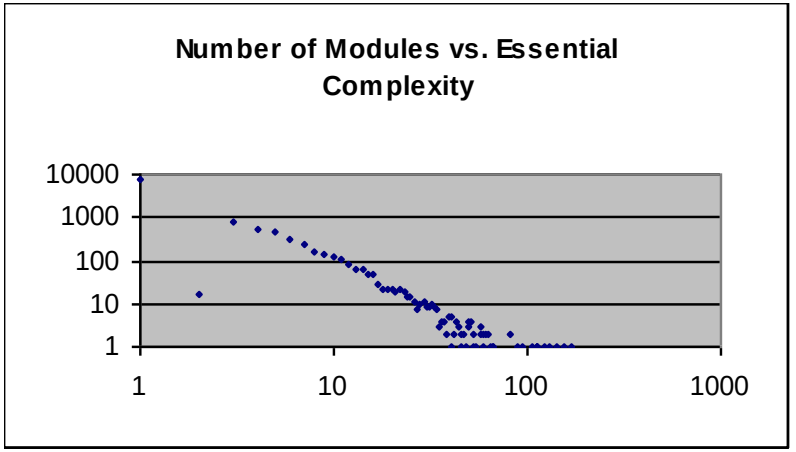
- 312K executable lines of C code
- 8 years of problem reports (life of project)
- 1652 Software problem reports whose fix included a change to at least one software module.
- 2078 modules changed at least once due to a problem report
- 4221 total module changes as the result of problem reports
- 12,094 modules as identified by McCabe IQ

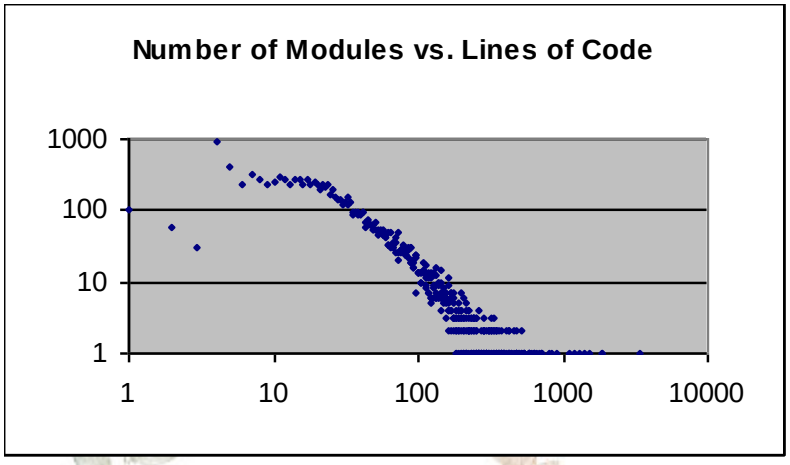
What Was Eliminated

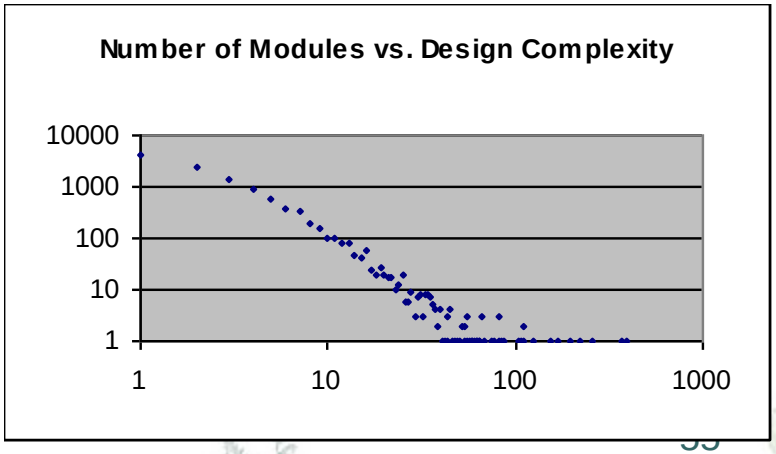
- A CSCI that had recently been redesigned using C++
- 1007 modules that no longer exist in the baseline or were ambiguous in their naming
- A CSCI that consisted of multiple directories of essentially the same code
- **Leaving 11,494 modules with 3154 errors**

Number of Modules

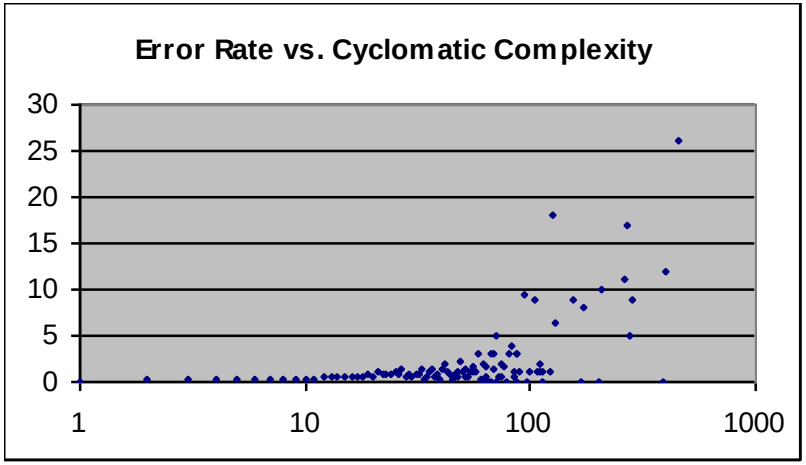


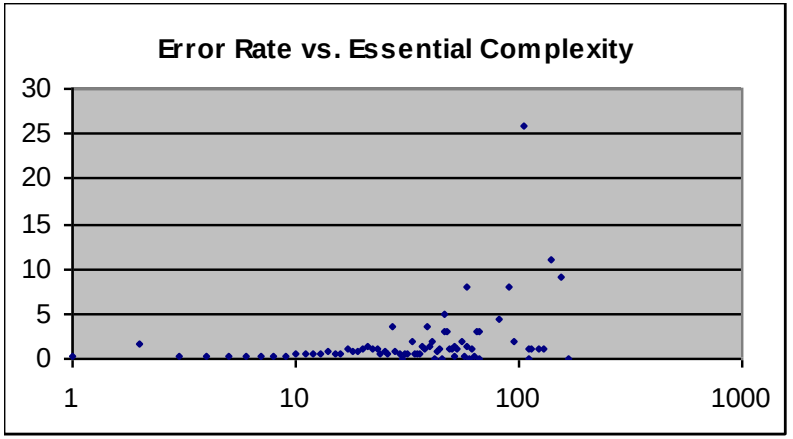


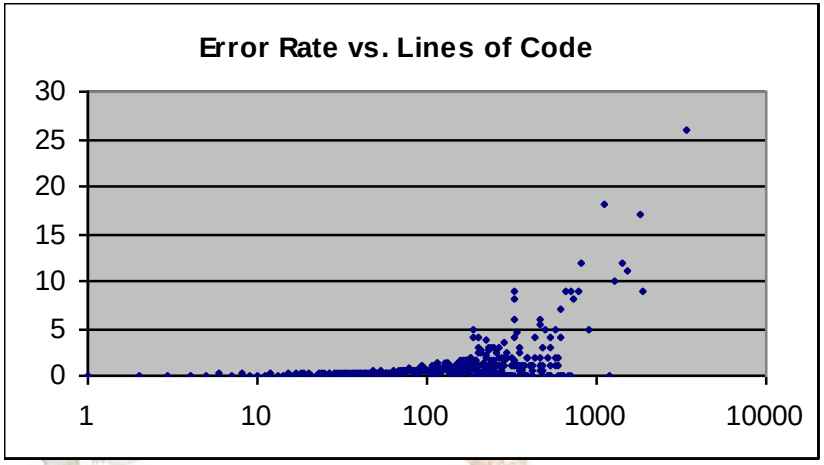


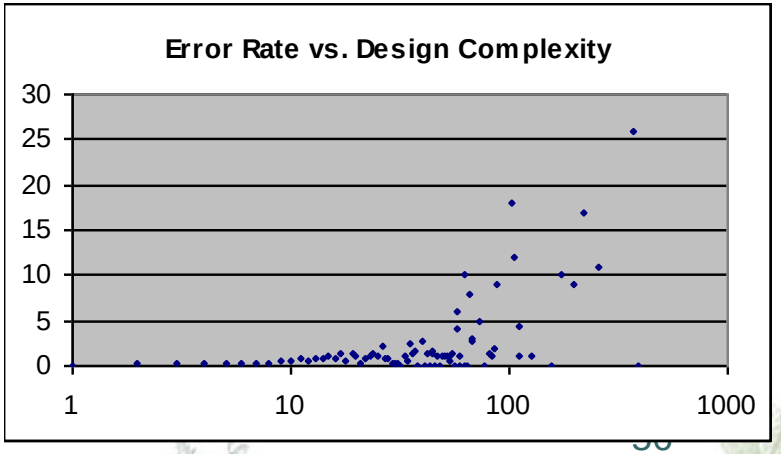


Error Rate


Error Rate vs. Cyclomatic Complexity
30
25
20
15
10
5
0
1
10
100
1000


Error Rate vs. Essential Complexity
30
25
20
15
10
5
0
1
10
100
1000

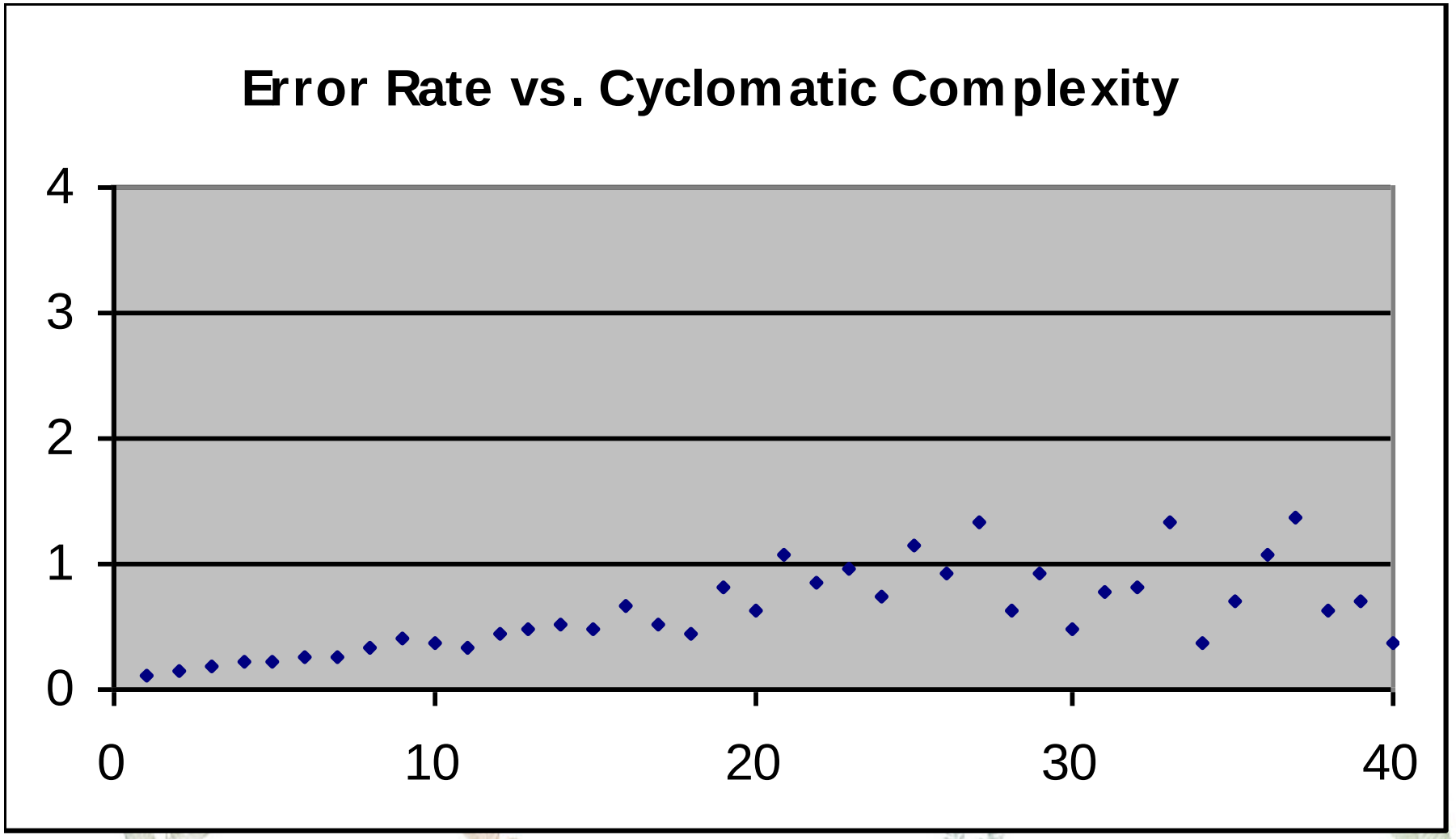

Error Rate vs. Lines of Code
30
25
20
15
10
5
0
1
10
100
1000
10000


Error Rate vs. Design Complexity
30
25
20
15
10
5
0
1
10
100
1000

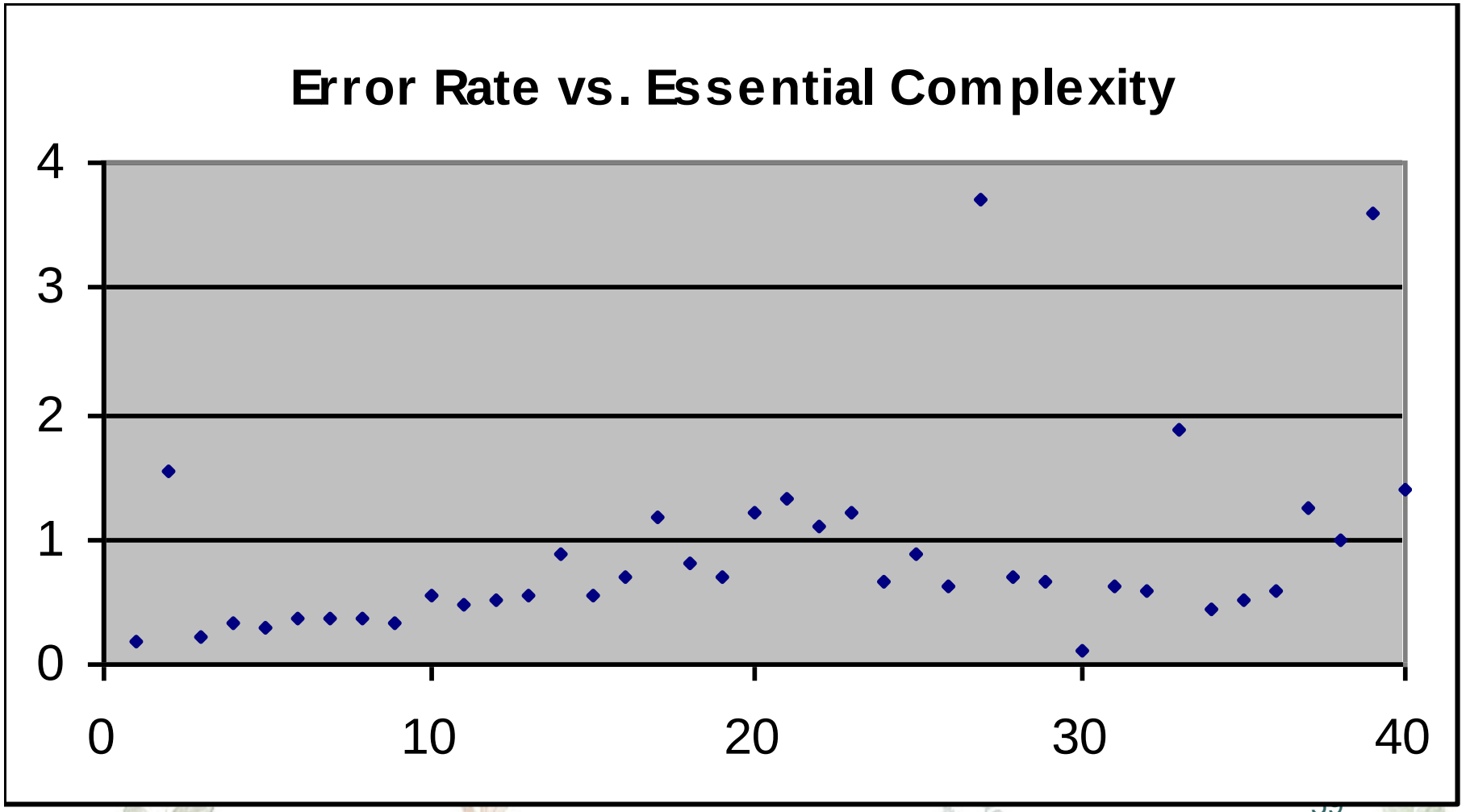
Error Prediction

- What does it mean to say that a metric is a good predictor of error(s)?
- Or that one is better than another?
- How does one select a threshold?

A Closer Look at v(G)

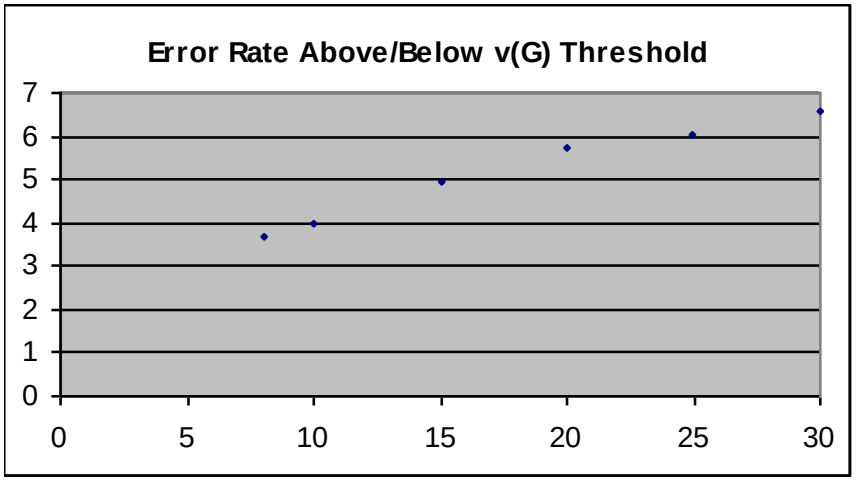


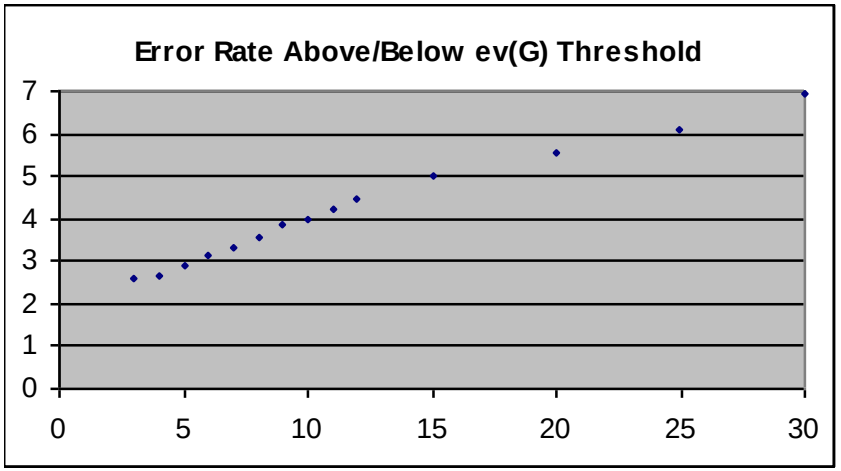
A Closer Look at ev(G)



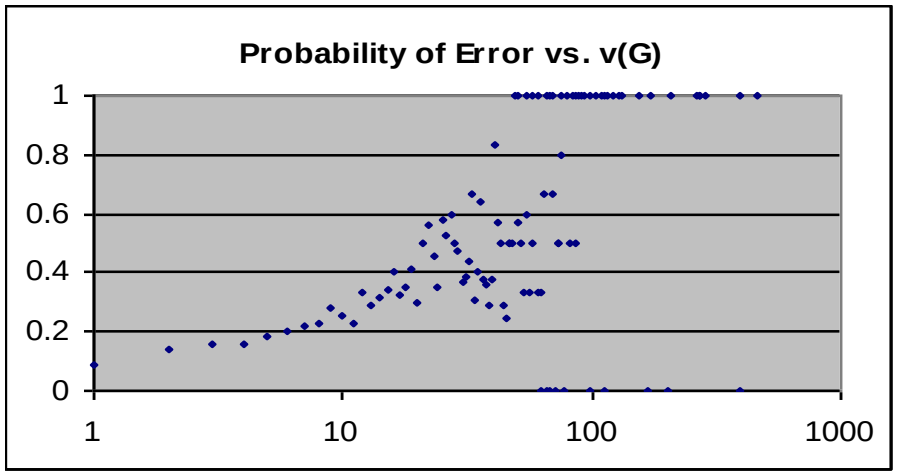
Thresholds

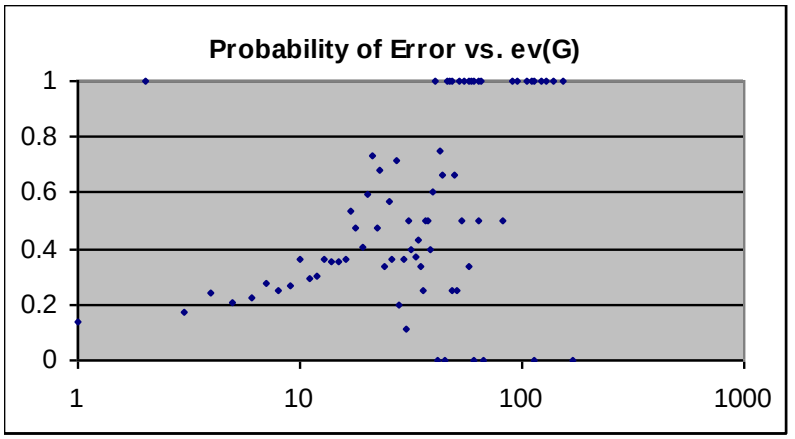
- Compare the error rate among modules above a threshold with that of those below

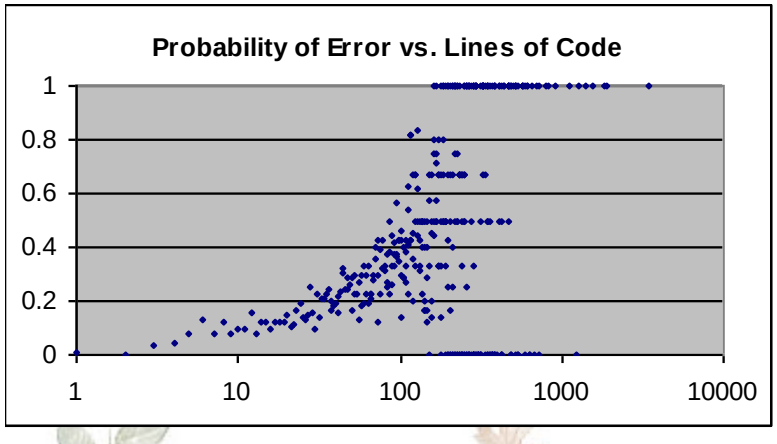


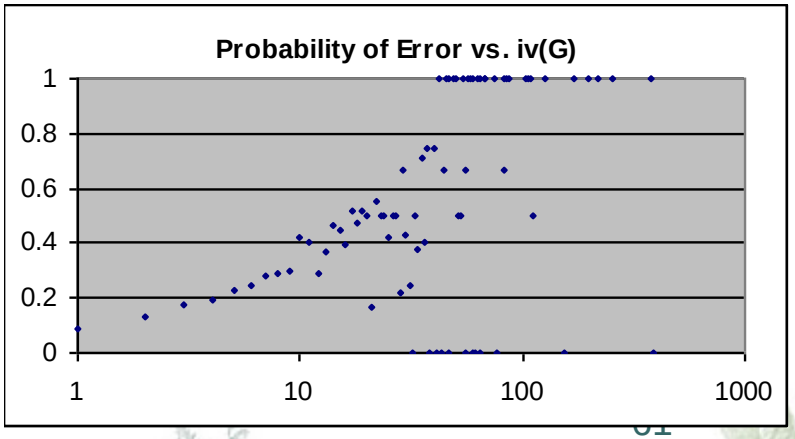


Probability of Error

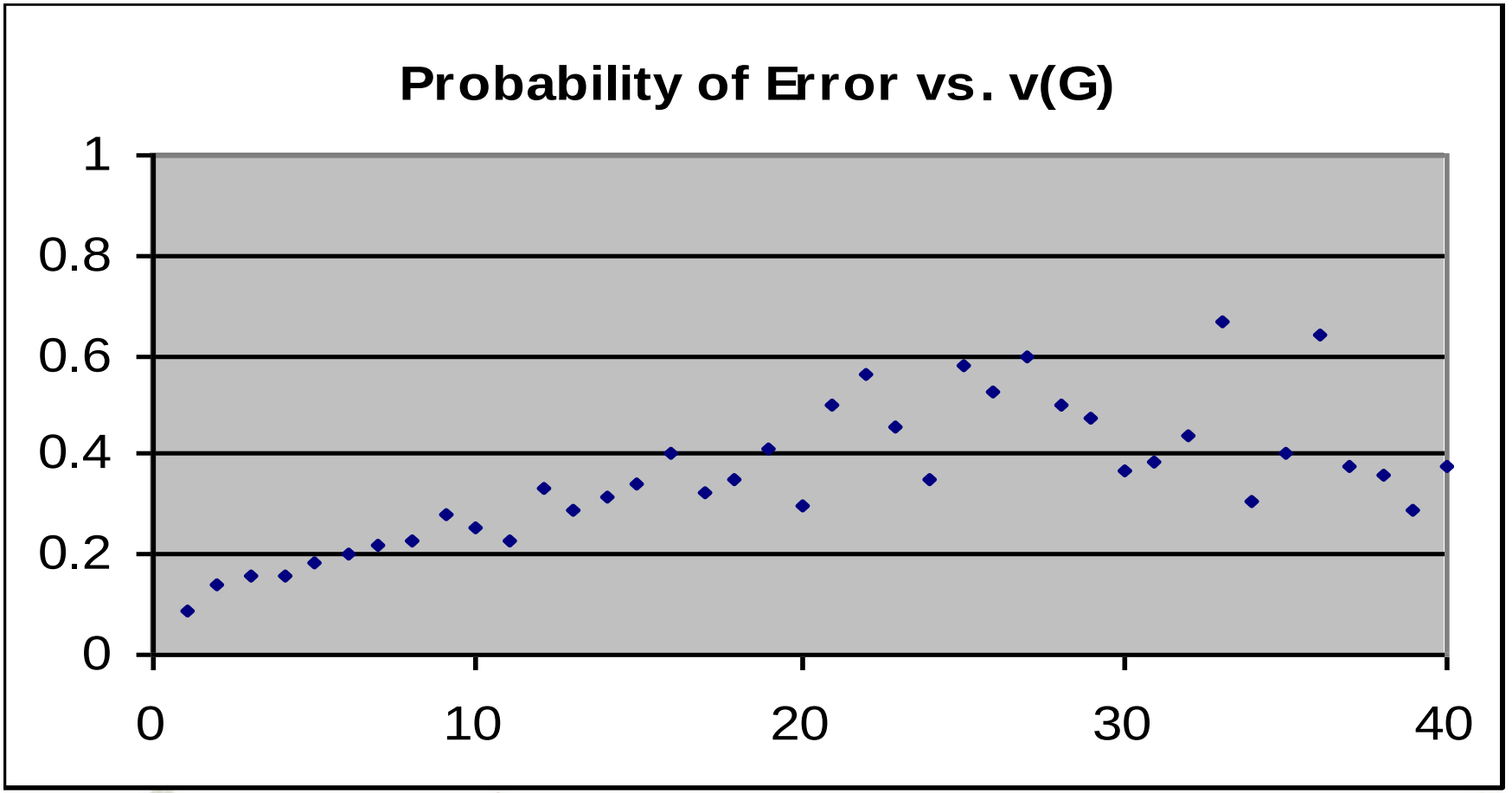




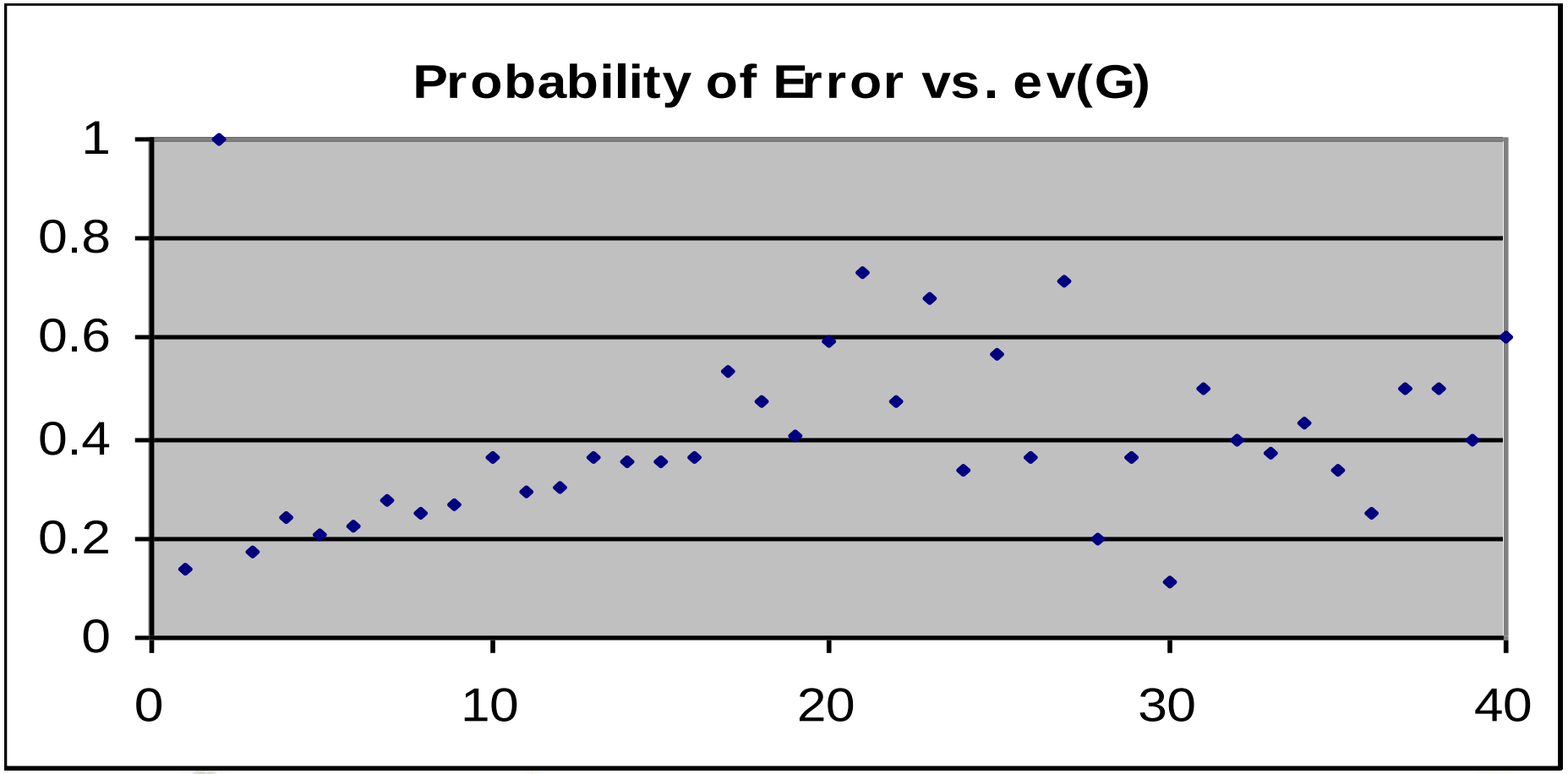




A Closer Look at v(G)

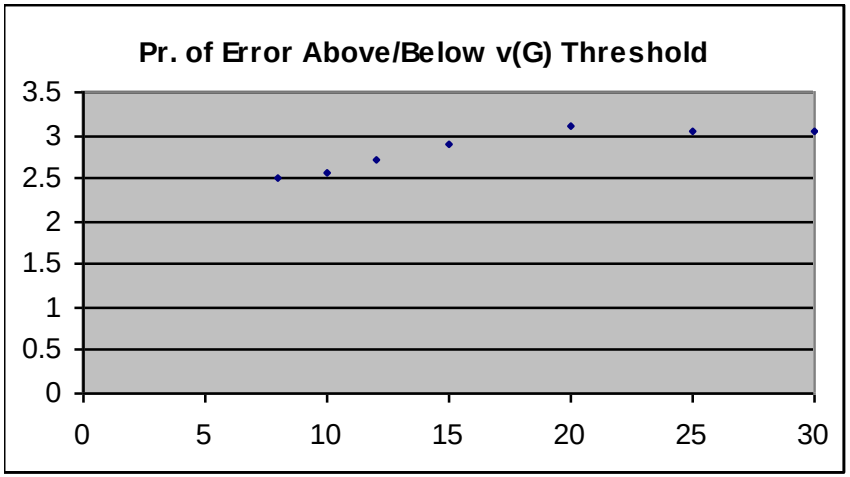


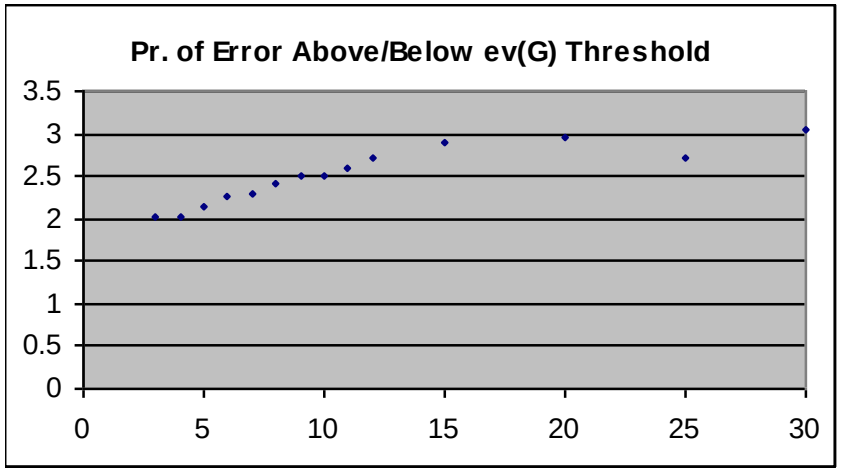
A Closer Look at ev(G)



Thresholds

- Compare the error probability for modules above a threshold with those below

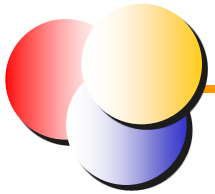




Conclusions

- Both cyclomatic and essential complexities indicate errors
- Cyclomatic is a better indicator
- The McCabe thresholds (10 and 4) aren't bad, but 20 and 10 look better

Summary

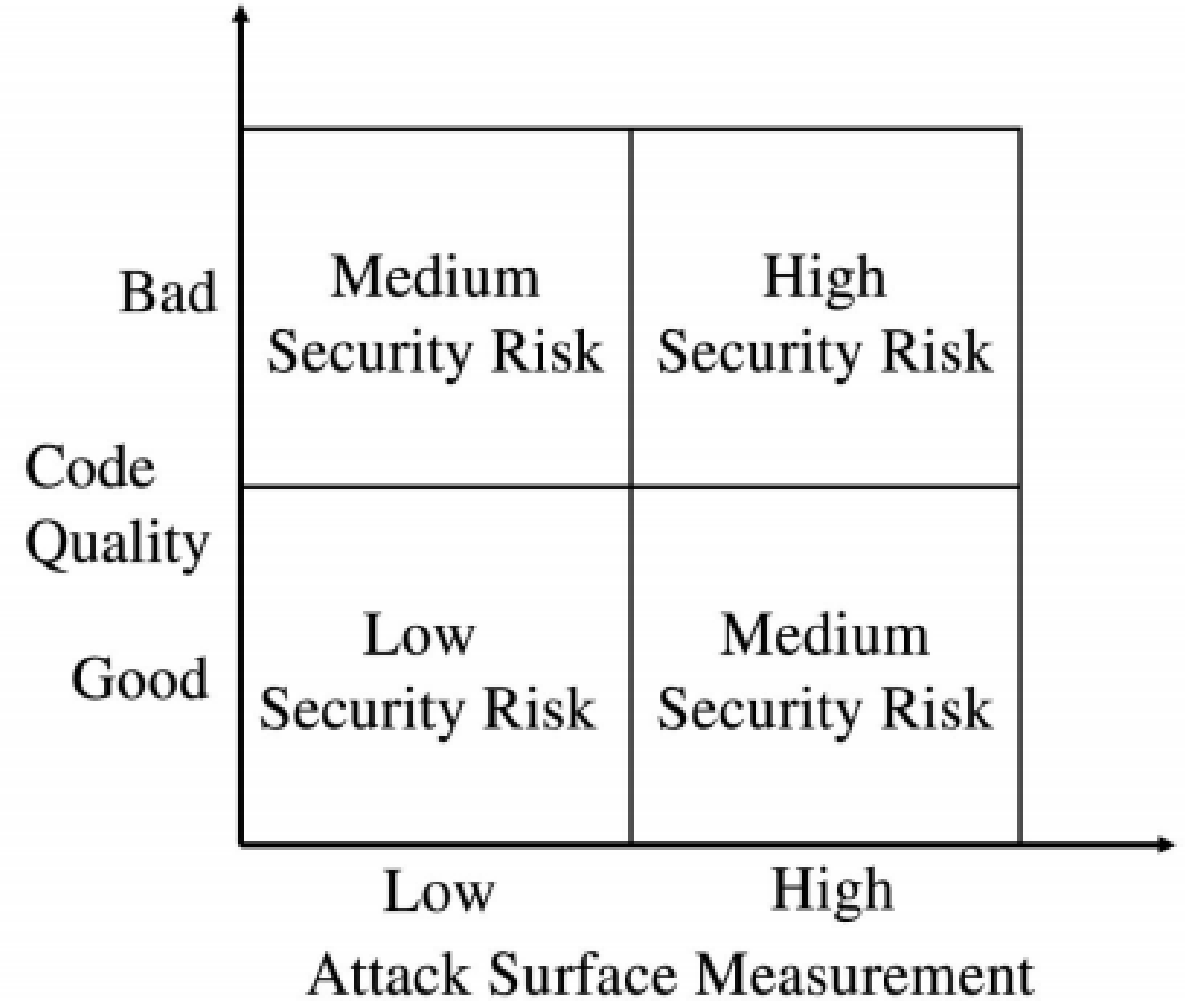


- **Control-flow structure ?**
- **Cyclomatic complexity ?**

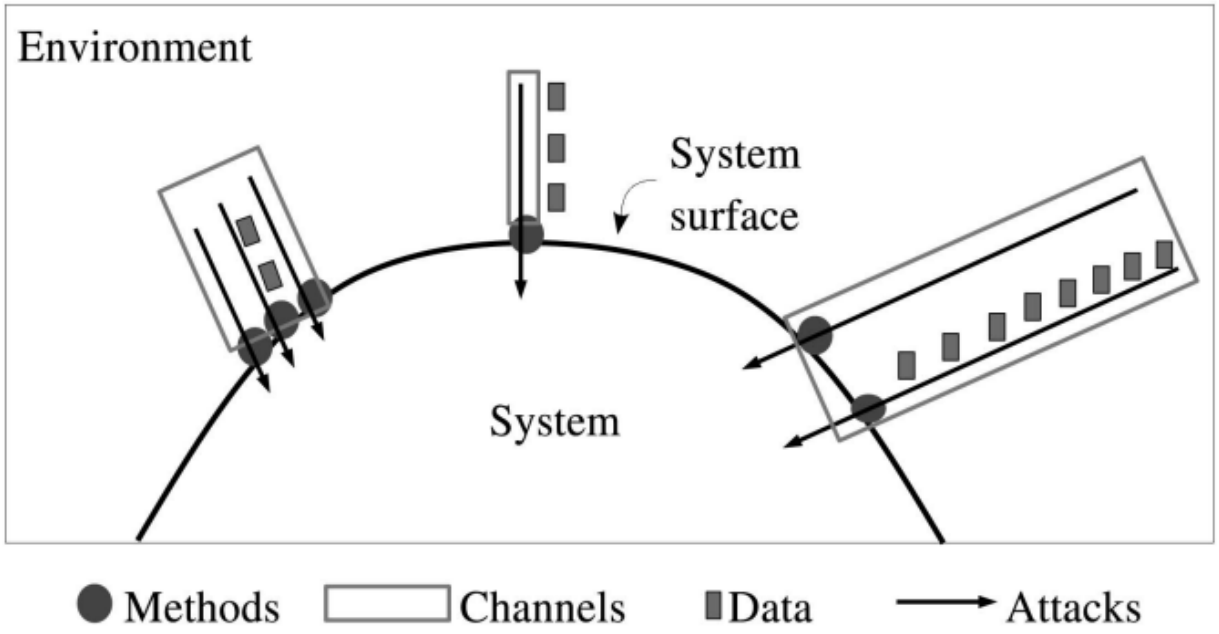



Thanks for your time and attention!

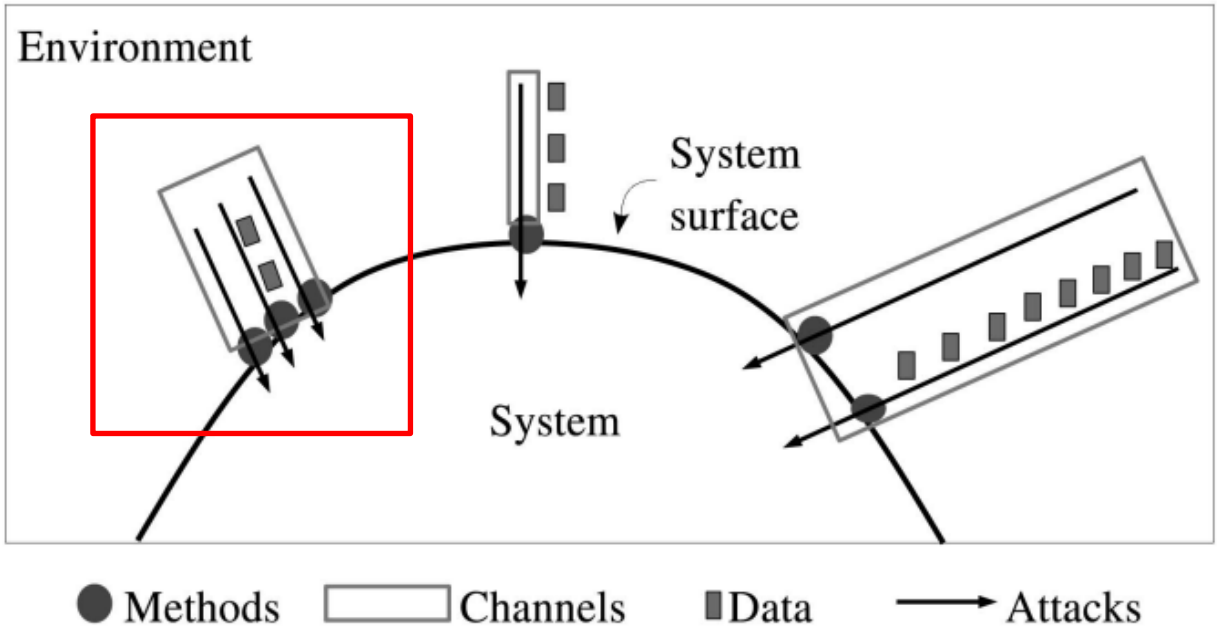
练习：软件攻击面计算


Bad
Code
Quality
Good
Medium
Security Risk
High
Security Risk
Low
Security Risk
Medium
Security Risk
Low
High
Attack Surface Measurement

练习：软件攻击面计算


Environment
System
surface
System
Methods
Channels
Data
Attacks

练习：软件攻击面计算


Environment
System
surface
System
Methods
Channels
Data
Attacks

练习：计算两个fpt服务器软件的攻击面

(1) 研读论文

IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 37, NO. 3, MAY/JUNE 2011 371

An Attack Surface Metric

Pratyusa K. Manadhata, *Member*, *IEEE*, and Jeannette M. Wing, *Fellow*, *IEEE*

Measuring the Attack Surfaces of Two FTP Daemons

Pratyusa Manadhata, Jeannette Wing
Carnegie Mellon University
{pratyus,wing}@cs.cmu.edu

Mark Flynn, Miles McQueen
Idaho National Laboratory
{Mark.Flynn, Miles.McQueen}@inl.gov

(2) 下载proftpd和wu-ftpd

https://github.com/proftpd/proftpd/archive/v1.3.5b.zip

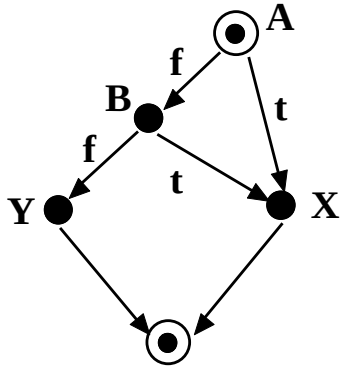
ftp://gd.tuwien.ac.at/infosys/servers/ftp/wu-ftpd/wu-ftpd/wu-ftpd-2.6.2.tar.gz

练习：计算两个fpt服务器软件的攻击面

(2) 计算proftpd和wu-ftpd在“method”维度上的攻击面

在12月18日前提交Perl脚本以及数据集

Less Common Control Constructs

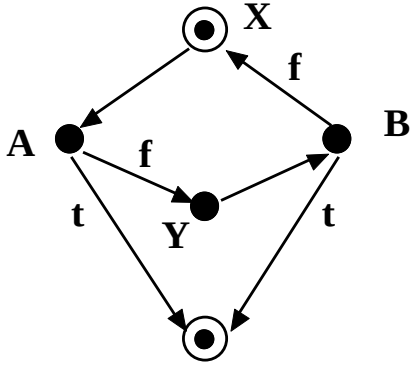


D_5 or $D_5(A, B, X, Y)$

'lazy Boolean evaluation OR'

If (A or B) then X else Y
in Turbo Pascal

If A or else B then X else Y
in Ada

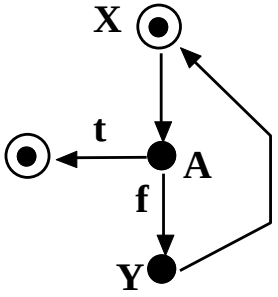


L_2 or $L_2(A, B, X, Y)$

'two-exit loop'

```
loop
  X; exit when A; Y; exit when B
loop end
```

in Ada



D_4 or $D_4(A, X, Y)$

'middle-exit loop'

```
loop
  X; exit when A; Y
loop end
```

in Ada