DESIGN: CHARACTERISTICS AND METRICS

Software Engineering
CS 130

Donald J. Patterson

Content adapted from Essentials of Software Engineering 3rd edition by Tsui, Karam, Bernal
Jones and Bartlett Learning

CHARACTERIZING "GOOD" DESIGN

- **Besides the obvious - - - design should match the requirements - - - there are two "basic" characteristics:**
  - **Consistency across design:**
    - **Common UI**
      - **looks**
      - **Logical flow**
    - **Common error processing**
    - **Common reports**
    - **Common system interfaces**
    - **Common help**
    - **All design carried to the same depth level** **(what do you think?)**
  - **Completeness of the design**
    - **All requirements are accounted for**
    - **All parts of the design is carried to its completion, to the same depth level**

INTUITIVELY, COMPLEXITY IS RELATED TO "GOOD/BAD" DESIGN

- **Some "Legacy Characterization" of Design Complexity**
  - **Halstead metrics**
  - **McCabe's Cyclomatic Complexity metric (most broadly used)**
  - **Henry-Kafura Information Flow (Fan-in/Fan-out) metrics**
  - **Card and Glass design complexity metrics**

HALSTEAD METRICS

- **Developed by Maurice Halstead of Purdue in the 1970's to mostly analyze program source code complexity.**
- **Used 4 fundamental units of measurements from code:**
  - **n1 = number of distinct operators**
  - **n2 = number of distinct operands**
  - **N1 = sum of all occurrences of the n1**
  - **N2 = sum of all occurrences of the n2**
- **Program vocabulary, n = n1 + n2**
- **Program length, N = N1 + N2**
- **Using these, he defined 4 metrics:**
  - **Volume , $V = N * (\log_2 n)$**
  - **Potential volume , $V@ = (2 + n2@) \log_2 (2+n2@)$** ***(based on most "succinct" program's n2 --- thus n2@)***
  - **Program Implementation Level, $L = V@/ V$**
  - **Effort, $E = V / L$**

Halstead metrics

- Four basic metrics of Halstead

	Total	Unique
Operators	N1	n1
Operands	N2	n2

- Length: $N = N1 + N2$
- Vocabulary: $n = n1 + n2$
- Volume: $V = N \log_2 n$
  - Insensitive to lay-out

Halstead metrics: Example

```
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;
              }
        }
   }
}
```

- Ignore the function definition
- Count operators and operands

3	<	3	{
5	=	3	}
1	>	1	+
1	-	2	++
2	,	2	for
9	;	2	if
4	(	1	int
4	)	1	return
6	[]		

1	0
2	1
1	2
6	a
8	i
7	j
3	n
3	t

	Total	Unique
Operators	N1 = 50	n1 = 17
Operands	N2 = 30	n2 = 7

$V = 80 \log_2(24) \approx 392$

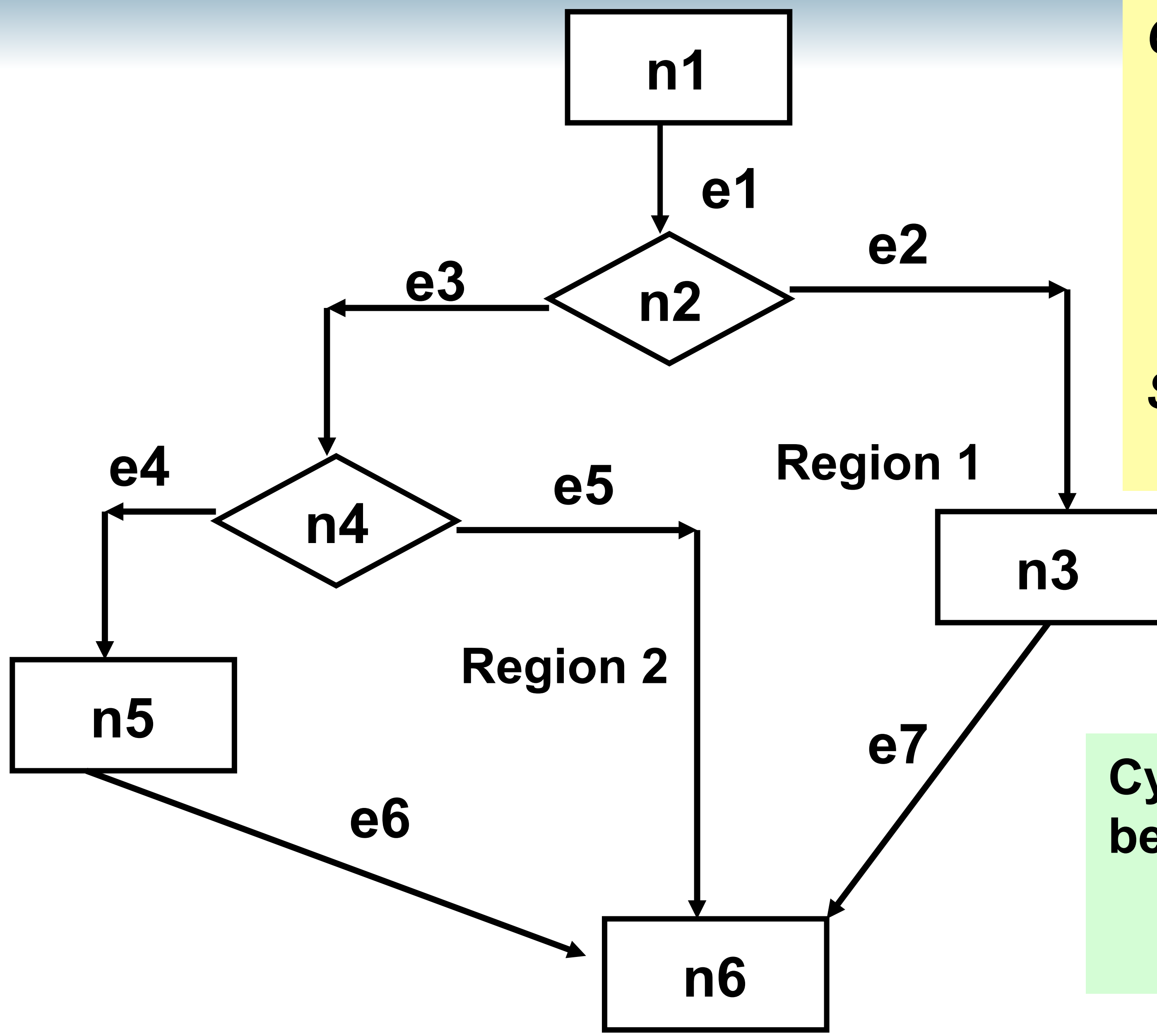
Inside the boundaries [20;1000]

Software metrics by Alexander Serebrenik

Further Halstead metrics

	Total	Unique
Operators	N1	n1
Operands	N2	n2

- Volume: V = N $\log_2 n$
- Difficulty: D = (n1 / 2) * (N2 / n2)
  - Sources of difficulty: new operators and repeated operands
  - Example: 17/2 * 30/7 ≈ 36
- Effort: E = V * D
- Time to understand/implement (sec): T = E/18
  - Running example: 793 sec ≈ 13 min
  - Does this correspond to your experience?
- Bugs delivered: $E^{2/3}/3000$
  - For C/C++: known to underapproximate
  - Running example: 0.19



Cyclomatic complexity = E - N + 2p
where E = number of edges
N= number of nodes
p = number of connected components (usually 1)

So, for this control flow :
7 edges – 6 nodes + 2 = 3

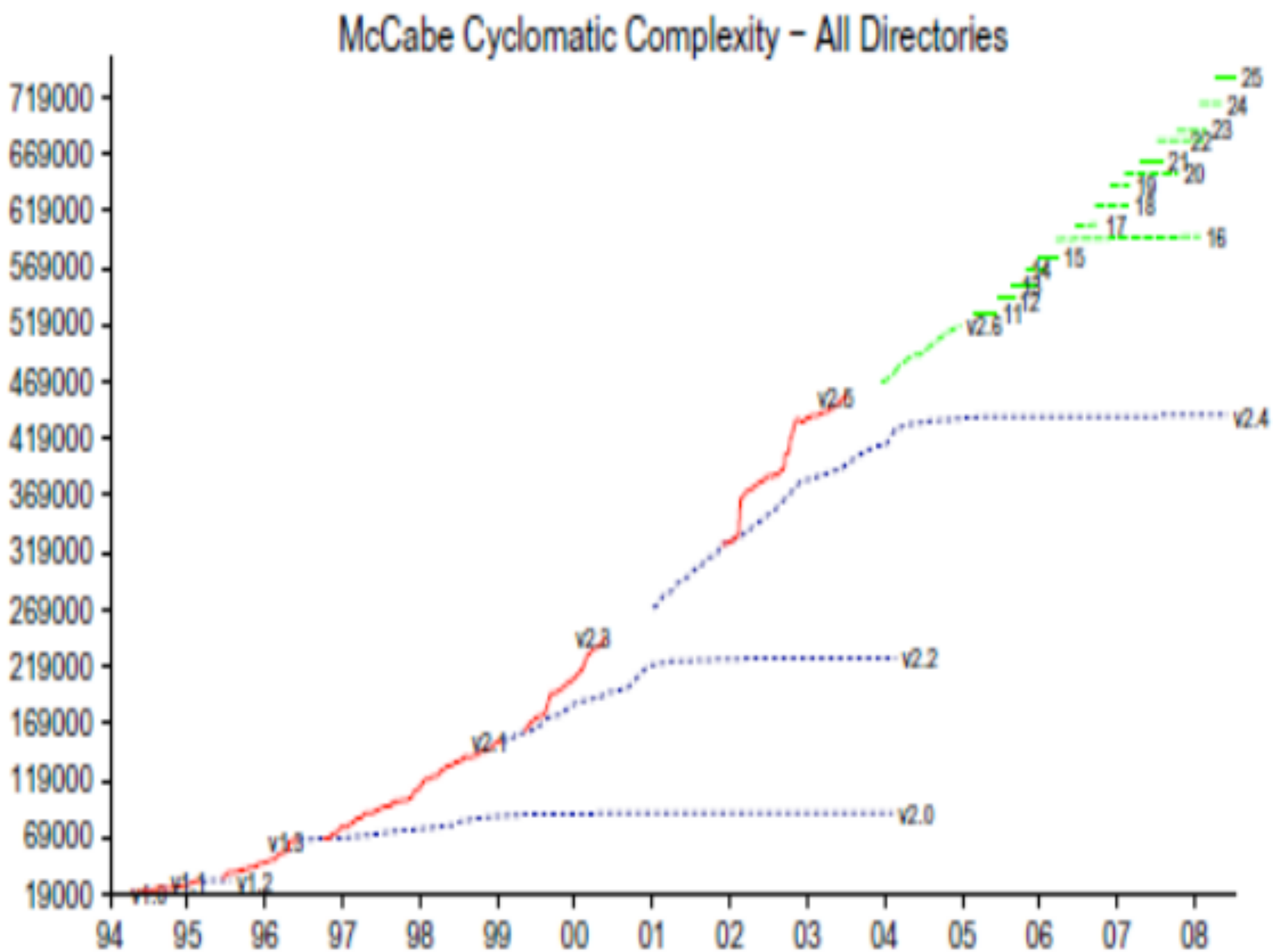
Cyclomatic complexity number can also be computed as follows:

- number of binary decision +1
- number of closed regions + 1

- T.J. **McCabe's Cyclomatic complexity** metric is based on the belief that ***program quality*** is **related to** the ***complexity of the program "control flow".***

Can be computed with static analysis

McCabe's complexity in Linux kernel



- Linux kernel
- Multiple versions and variants
  - Production (blue dashed)
  - Development (red)
  - Current 2.6 (green)

A. Israeli, D.G. Feitelson 2010

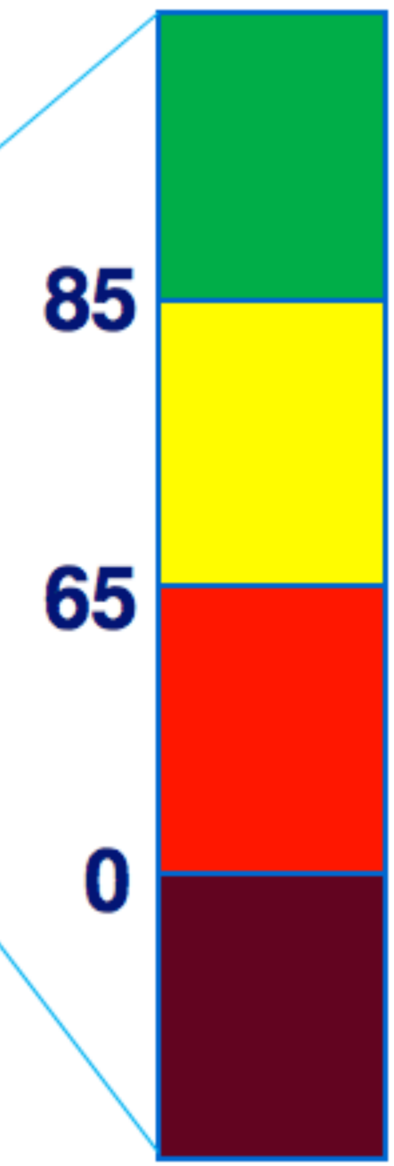
MAINTAINABILITY INDEX

$$MI_1 = 171 - 5.2\ln(V) - 0.23V(g) - 16.2\ln(LOC)$$

Halstead **McCabe** **LOC**

$$MI_2 = MI_1 + 50\sin\sqrt{2.46\,perCM}$$

% comments



- **MI_2 can be used only if comments are meaningful**
- **If more than one module is considered – use average values for each one of the parameters**
- **Parameters were estimated by fitting to expert evaluation**
  - **BUT: few middle-sized systems!**

MAINTAINABILITY INDEX

McCabe complexity: Example

```
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;
                }
         }
   }
}
```

- Halstead's V ≈ 392
- McCabe's v(G) = 5
- LOC = 14
- $MI_1 \approx 96$
- Easy to maintain!

Software metrics by Alexander Serebrenik

STATIC ANALYSIS VS RUN-TIME ANALYSIS

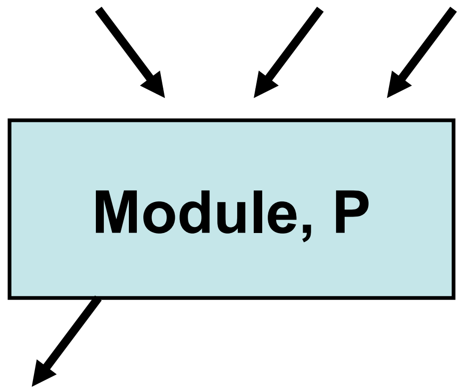
WHAT IS THE DIFFERENCE?

- Static Analysis
  - Automatic analysis conducted on source code
  - Every IDE worth it’s salt does this

- Dynamic Analysis
  - Automatic analysis conducted on running code
    - “profiling”

HENRY-KAFURA (FAN-IN AND FAN-OUT)

- Henry and Kafura metric measures the inter-modular flow, which includes:
  - Parameter passing
  - Global variable access
  - inputs
  - outputs
- Fan-in : number of inter-modular flow into a program
- Fan-out: number of inter-modular flow out of a program

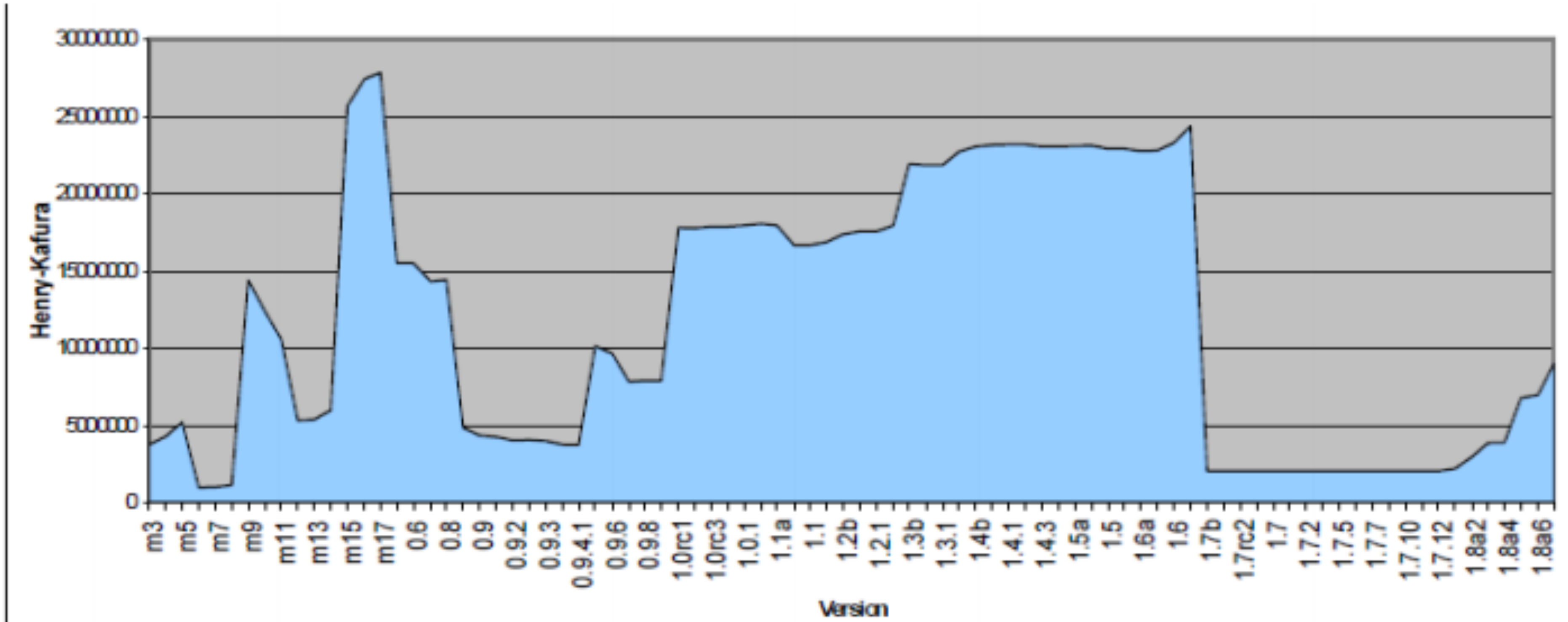


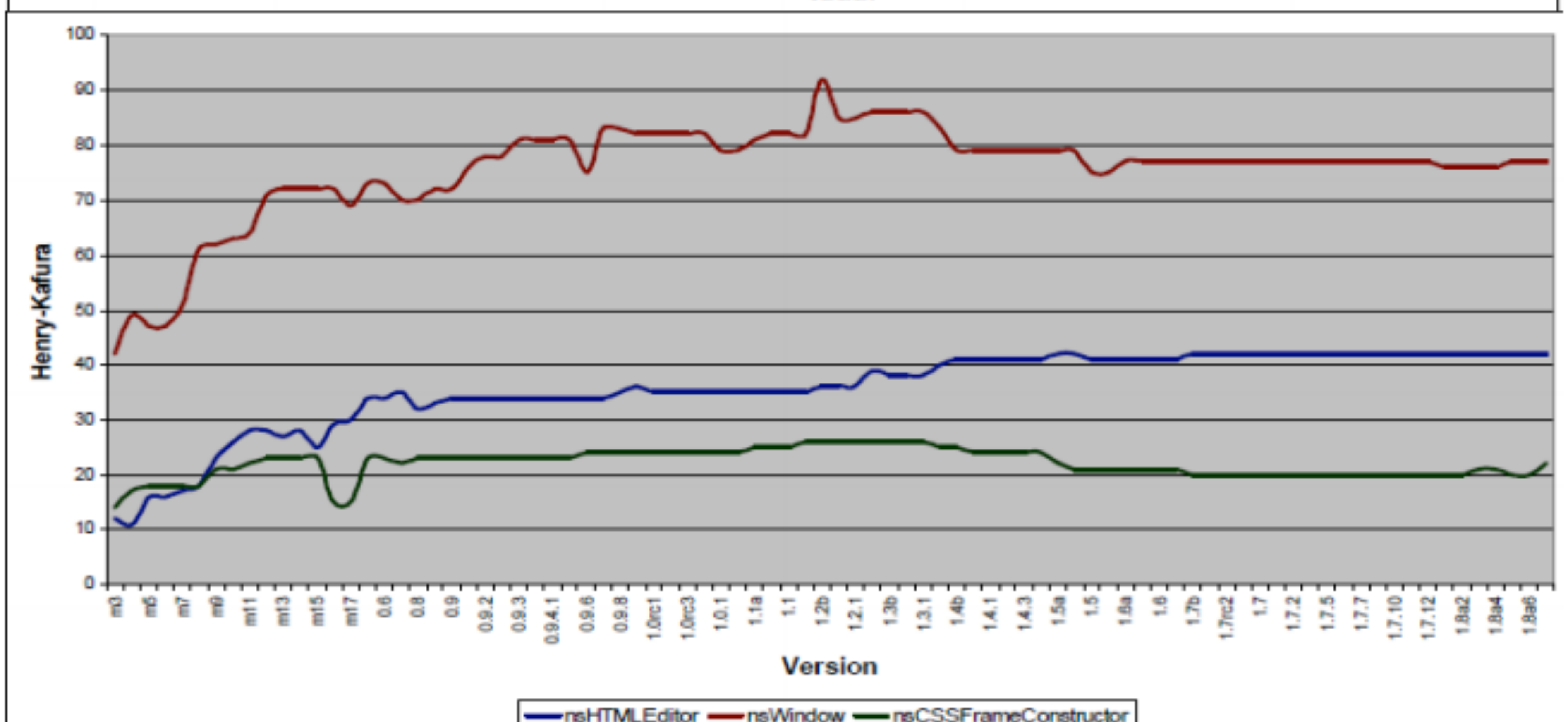
non-linear

Module's Complexity, Cp = (fan-in x fan-out)^2

for the "picture" above: Cp = $(3 \times 1)^2$ = 9

Evolution of the information flow complexity





- **Mozilla**
- **Shepperd version**
- **Above: Σ the metrics over all modules**
- **Below: 3 largest modules**
- **What does this tell?**

Technische Universiteit Eindhoven University of Technology

CARD AND GLASS (HIGHER LEVEL COMPLEXITY)

- **Card and Glass used the same concept of fan-in and fan-out to describe design complexity:**
  - **Structural complexity of module x**
    - **Sx = (fan-out)^2**
  - **Data complexity**
    - **Dx = Px / (fan-out +1), where Px is the number of *variables* passed to and from the module**
  - **System complexity**
    - **Cx = Sx + Dx**

Note: Except for Px, fan-in is not considered here

A LITTLE “DEEPER” ON GOOD DESIGN ATTRIBUTES

- **Easy to**:
  - **Understand**
  - **Change**
  - **Reuse**
  - **Test**
  - **Integrate**
  - **Code**
- **Believe that we can get many of these “easy to’ s” if we consider:**
  - ***Cohesion***
  - ***Coupling***

COHESION

- **Cohesion of a unit, of a module, of an object, or a component addresses the attribute of "*degree of relatedness*" within that unit, module, object, or component.**

Functional ← **Performing 1 single function**

Sequential

Communicational

Procedural

Temporal

Logical

Coincidental ← **Performing more than 1 unrelated functions**

Levels of Cohesion where *Functional* is the "highest"

Higher the better

USING PROGRAM AND DATA SLICES TO MEASURE PROGRAM COHESION

- **Bieman and Ott introduced a measure of program cohesion using the following concepts from program and data slices:**
  - **A data token is any occurrence of variable or constant in the program**
  - **A slice within a program is the collection of all the statements that can affect the value of some specific variable of interest.**
  - **A data slice is the collection of all the data tokens in the slice that will affect the value of a specific variable of interest.**
  - **Glue tokens are the data tokens in the program that lie in more than one data slice.**
  - **Super glue tokens are the data tokens in the program that lie in every data slice of the program**

Measure Program Cohesion through 2 metrics:

- weak functional cohesion = (# of glue tokens) / (total # of data tokens)
- strong functional cohesion = (#of super glue tokens) / (total # of data tokens)

Finding the maximum and the minimum values procedure:

```
MinMax ( z, n)
Integer  end, min, max, i ;
end = n ;
max = z[0] ;
min = z[0] ;
For ( i = 0, i = < end , i++ ) {
     if  z[ i ] > max then max = z[ i ];
     if  z[ i ] < min then min = z[ i ];
                                      }
  return max, min;
```

Data Tokens:	Slice max:	Slice min:	Glue Tokens:	Super Glue:
z1	z1	z1	z1	z1
n1	n1	n1	n1	n1
end1	end1	end1	end1	end1
min1	max1	min1	i1	i1
max1	i1	i1	end2	end2
i1	end2	end2	n2	n2
end2	n2	n2	i2	i2
n2	max2	min2	03	03
max2	z2	z3	i3	i3
z2	01	02	end3	end3
01	i2	i2	i4 (11)	i4 (11)
min2	03	03		
z3	i3	i3		
02	end3	end3		
i2	i4	i4		
03	z4	z6		
i3	i5	i7		
end3	max3	min3		
i4	max4	min4		
z4	z5	z7		
i5	i6	i8		
max3	max5	min5		
max4	(22)	(22)		
z5				
i6				
z6				
i7				
min3				
min4				
z7				
i8				
max5				
min5 (33)				

A Pseudo-Code Example of Functional Cohesion Measure

EXAMPLE OF PSEUDO-CODE COHESION METRICS

- **For the example of finding min and max, the glue tokens are the same as the super glue tokens.**
  - **Super glue tokens = 11**
  - **Glue tokens = 11**
- **The data slice for min and data slice for max turns out to be the same number, 22**
- **The total number of data tokens is 33**

The cohesion metrics for the example of min-max are:
weak functional cohesion = 11 / 33 = 1/3
strong functional cohesion = 11 / 33 = 1/3

If we had only computed one function (e.g. max), then :
weak functional cohesion = 22 / 22 = 1
strong functional cohesion = 22/ 22 = 1

COUPLING

- **Coupling addresses the attribute of "*degree of interdependence*" between software units, modules or components.**

Content Coupling ← **Accessing the internal data or procedural information**

Common Coupling

Control Coupling

Stamp Coupling

Data Coupling ← **Passing only the necessary information**

No Coupling ← **Ideal, but not practical**

Levels of coupling where *Data coupling* is lowest

Lower the better

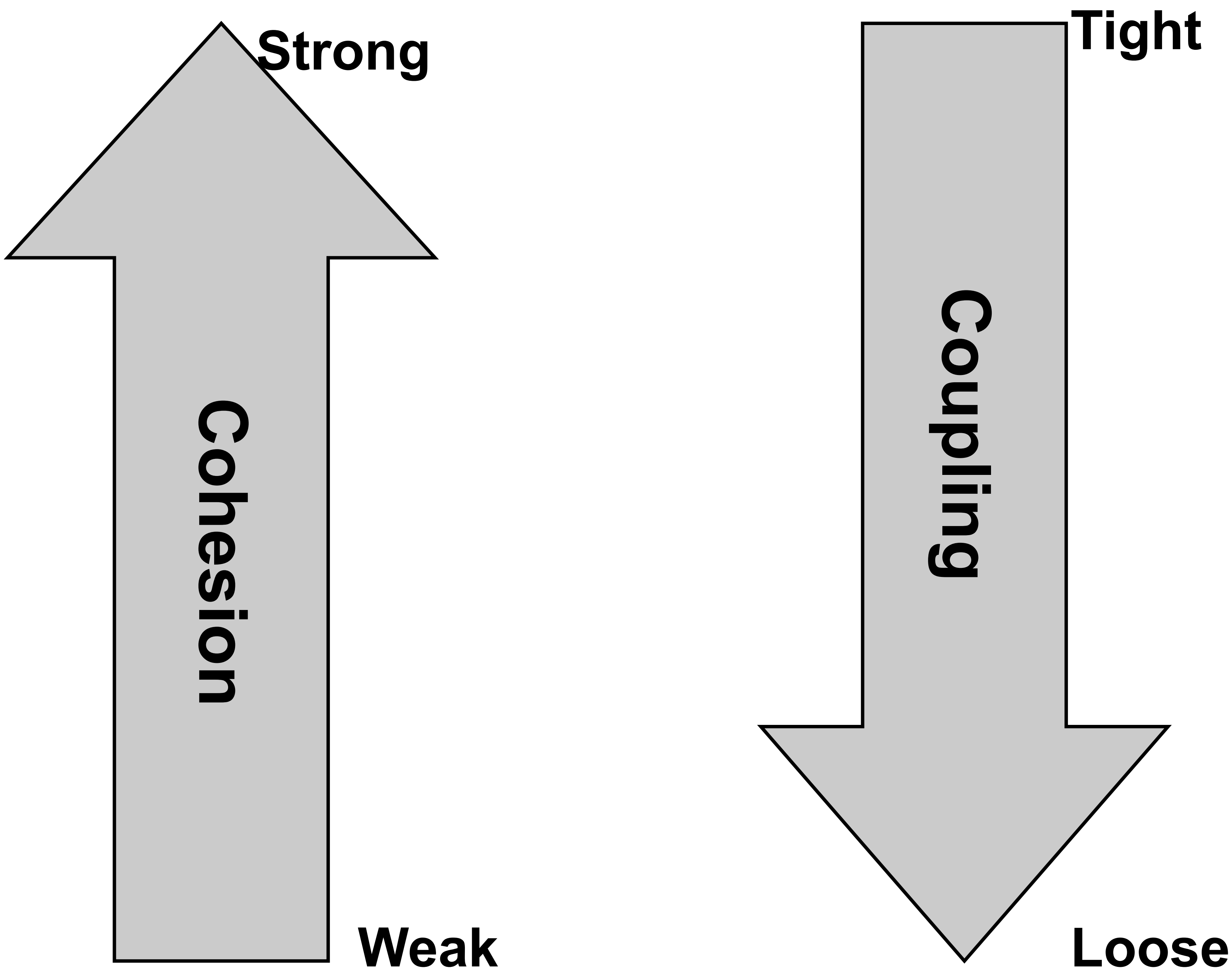
CHIDAMBER AND KEMERER (C-K) OO METRICS

- **Weighted Methods per class (WMC)**
- **Depth of Inheritance Tree (DIT)**
- **Number of Children (NOC)**
- **Coupling Between Object Classes (CBO)**
- **Response for a Class (RFC)**
- **Lack of Cohesion in Methods (LCOM)**

Note that LCOM is a <u>reverse measure</u> in that high LCOM indicates low cohesion and possibly high complexity. #p = number of pairs of methods in class that have no common instance variable; #q = number of pairs of methods in the class that have common instance variables
LCOM = #p - #q

COHESION AND COUPLING

High Level



Low Level

ORIGIN OF LAW OF DEMETER

- A design "guideline" for OO systems that originated from the Demeter System project at:
  - Northeastern University in the 1980's
  - Aspect-Oriented Programming Project
- Addresses the design coupling issue through placing constraints on messaging among the objects
  - Limit the sending of messages to objects that are directly known to it

LAW OF DEMETER

- **An object should send messages to only the following kinds of objects:**
  - ***the object itself***
  - ***the object's attributes (instance variables)***
  - ***the parameters of the methods in the object***
  - ***any object created by a method in the object***
  - ***any object returned from a call to one of the methods of the object***
  - ***any object in any collection that is one of the above categories***

USER INTERFACE

- Mandel's 3 "golden rules" for UI design
  - *Place the user in control*
  - *Reduce the users' memory load (G. Miller's 7 + or – 2)*
  - *Consistency (earlier - design completeness and consistency)*
- Shneiderman and Plaisant (8 rules for design)
  - *Consistency*
  - *Short cuts for frequent (or experienced) users*
  - *Informative feedback*
  - *Dialogues should result in closure*
  - *Strive for error prevention and simple error handling*
  - *Easy reversal of action ("undo" of action)*
  - *Internal locus of control*
  - *Reduce short term memory*

UI DESIGN PROTOTYPE AND "TEST"

- **UI design prototypes**:
  - Low fidelity (with cardboards)
  - High fidelity (with "story board" tools)

- **Usability "laboratories test"** and statistical analysis
  - # of subjects who can complete the tasks within some specified time
  - Length of time required to complete different tasks
  - Number of times "help" functions needed
  - Number of times "redo" used and where
  - Number of times "short cuts" were used

WESTMONT INSPIRED

COMPUTING LAB