2IS55 Software Evolution

Software metrics (2)

Alexander Serebrenik

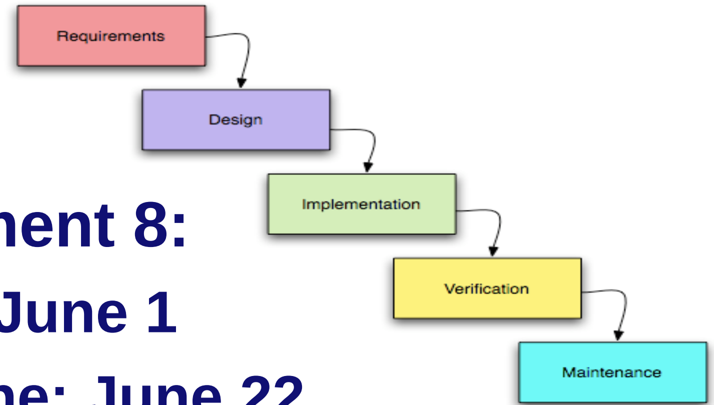
Technische Universiteit
Eindhoven
University of Technology

Where innovation starts

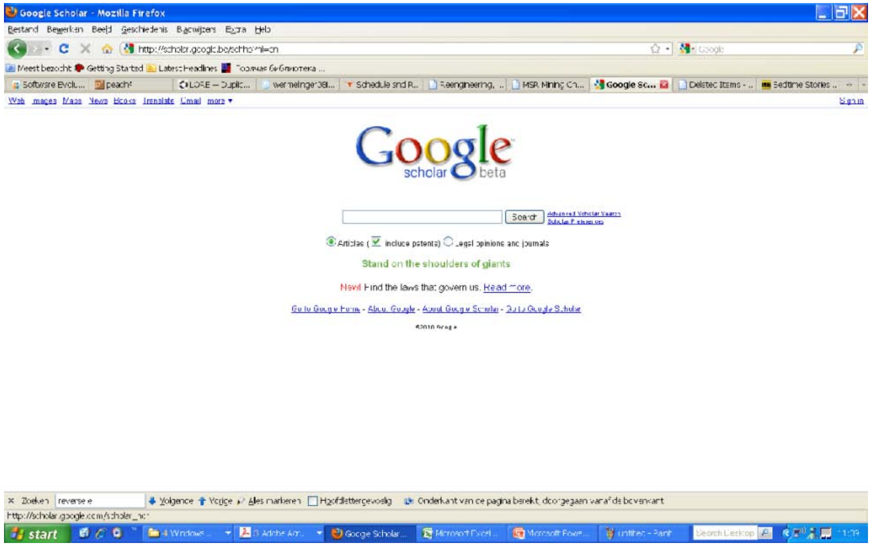
Assignment 6

- **Assignment 6:**
  - **Deadline: May 11**
  - **1-2 students**

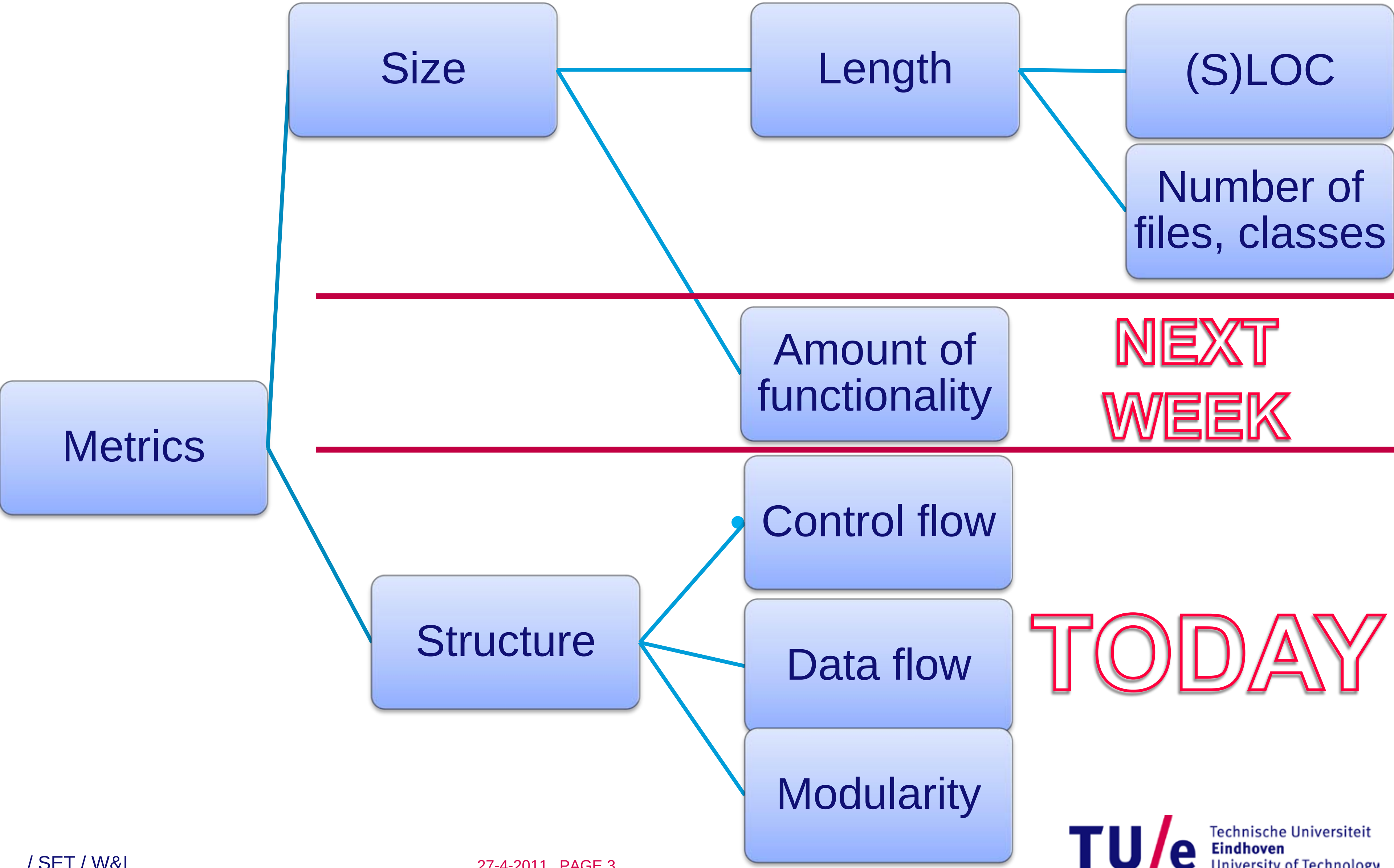
- **Assignment 8:**
  - **Open: June 1**
  - **Deadline: June 22**
  - **1-2 students**
  - **ReqVis** http://www.student.tue.nl/Q/w.j.p.v.ravensteijn/index.html
    - **Try it!**
    - **Give us feedback before June 1!**
    - **Mac-fans: Talk to Wiljan!**



Sources

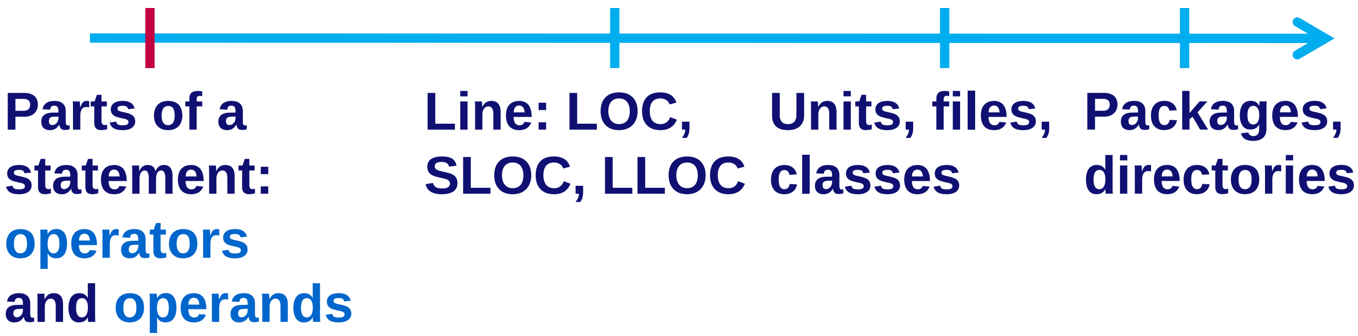

Google Scholar - Mozilla Firefox
Google
scholar beta
Stand on the shoulders of giants
New! Find the laws that govern us. Read more.

So far


Metrics
Size
Length
(S)LOC
Number of files, classes
Amount of functionality
NEXT WEEK
Structure
Control flow
Data flow
Modularity
TODAY

Complexity metrics: Halstead (1977)

- **Sometimes is classified as size rather than complexity**
- **Unit of measurement**



- **Operators:**
  - **traditional (+,++, >), keywords (return, if, continue)**
- **Operands**
  - **identifiers, constants**

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**
  - **VerifySoft:**
    - **20 ≤ Volume(function) ≤ 1000**
    - **100 ≤ Volume(file) ≤ 8000**

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]

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

Technische Universiteit Eindhoven University of Technology

Halstead metrics are sensitive to…

- **What would be your answer?**

- **Syntactic sugar:**

i = i+1	Total	Unique
Operators	N1 = 2	n1 = 2
Operands	N2 = 3	n2 = 2

i++	Total	Unique
Operators	N1 = 1	n1 = 1
Operands	N2 = 1	n2 = 1

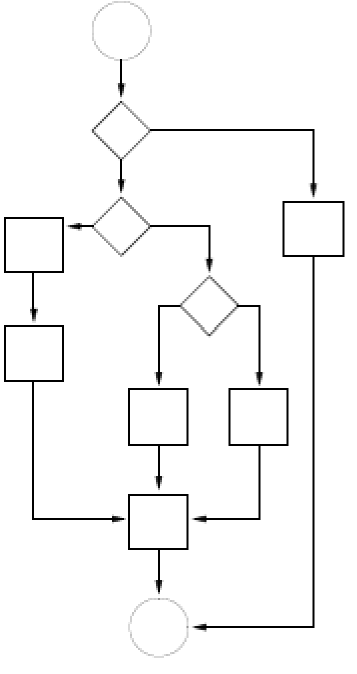
- **Solution: normalization (see the code duplication slides)**

Structural complexity

- **Structural complexity:**
  - **Control flow**
  - **Data flow**

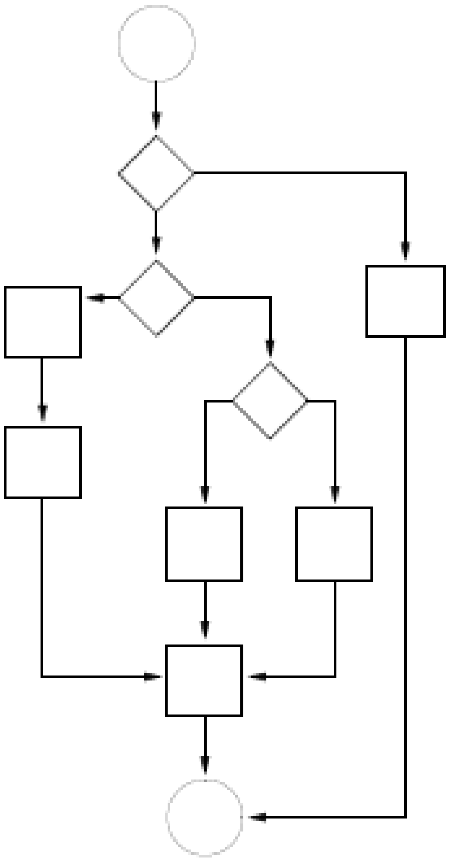
  } **Commonly represented as graphs** → **Graph-based metrics**

  - **Modularity**



- **Number of vertices**
- **Number of edges**
- **Maximal length (depth)**

McCabe complexity (1976)



In general

- v(G) = #edges - #vertices + 2

For control flow graphs

- v(G) = #binaryDecisions + 1, or
- v(G) = #IFs + #LOOPs + 1

Number of paths in the control flow graph.

A.k.a. "cyclomatic complexity"

Each path should be tested!

v(G) – a testability metrics

Boundaries
- v(function) $\leq$ 15
- v(file) $\leq$ 100

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;
                    }
          }
   }
}
```

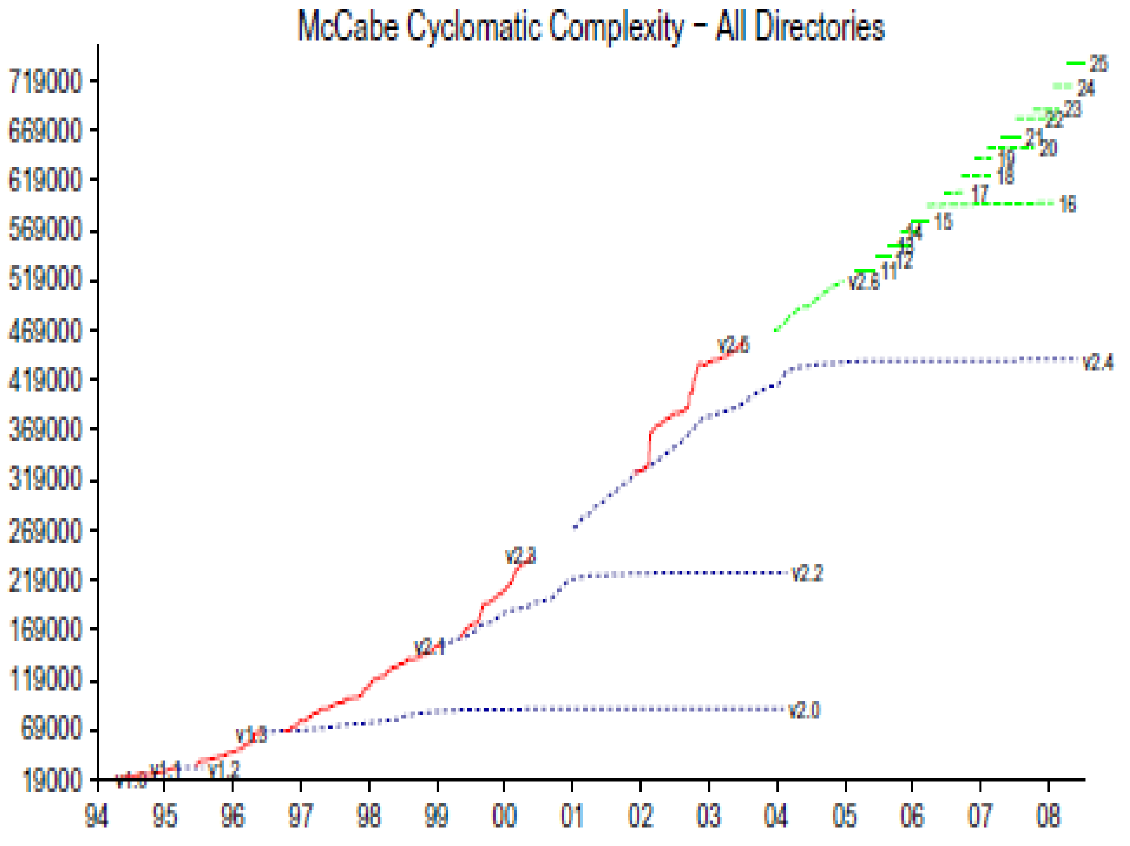
- Count IFs and LOOPs
- IF: 2, LOOP: 2
- v(G) = 5
- Structural complexity

Question to you

- **Is it possible that the McCabe's complexity is higher than the number of possible execution paths in the program?**

- **Lower than this number?**

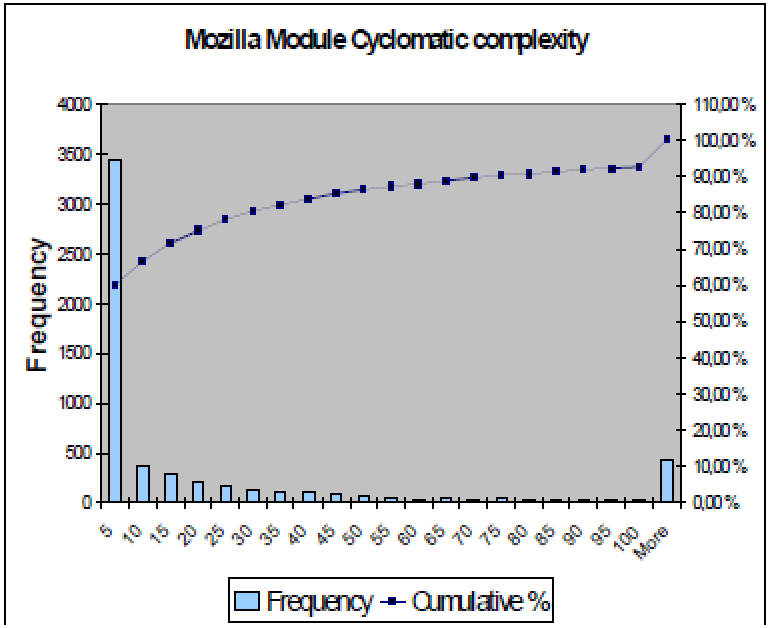
McCabe's complexity in Linux kernel

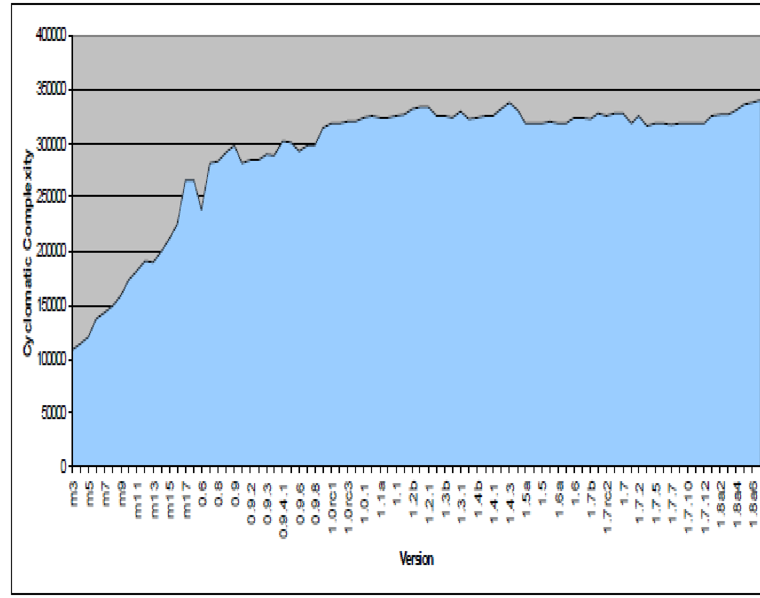


A. Israeli, D.G. Feitelson 2010

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

McCabe's complexity in Mozilla [Røsdal 2005]





- **Most of the modules have low cyclomatic complexity**
- **Complexity of the system seems to stabilize**

Technische Universiteit Eindhoven University of Technology

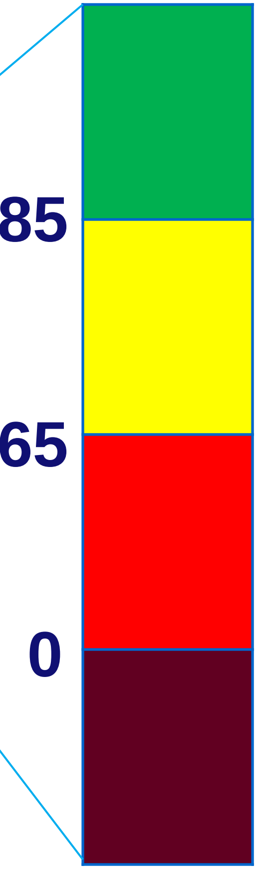
Summarizing: Maintainability index (MI) [Coleman, Oman 1994]

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

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 not big systems!

McCabe complexity: Example

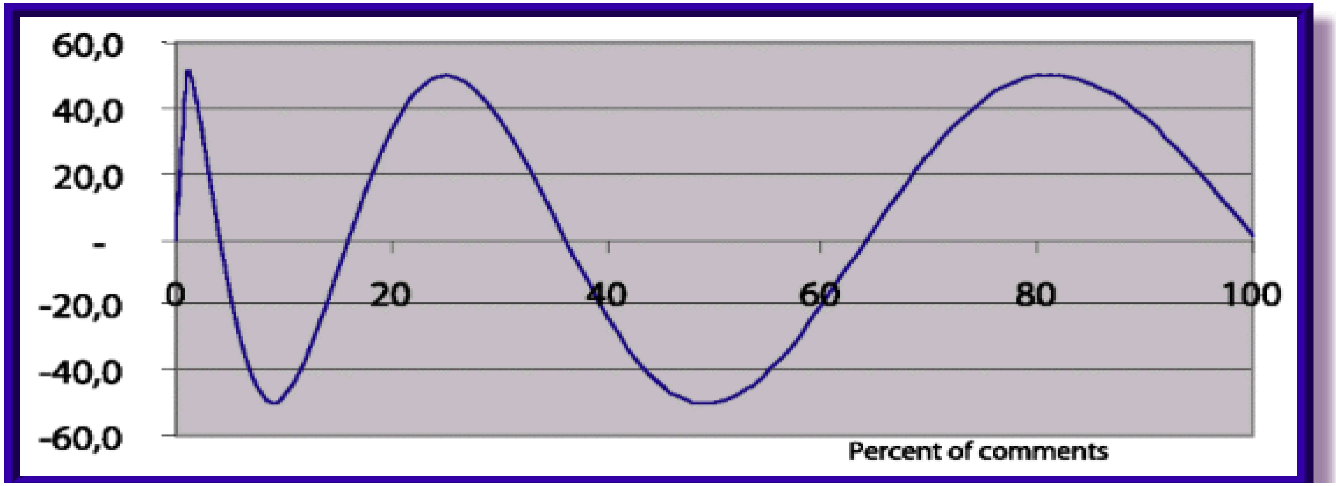
```
void sort ( int *a, int n ) {
int i, j, t;

  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;
              }
        }
  }
}
```

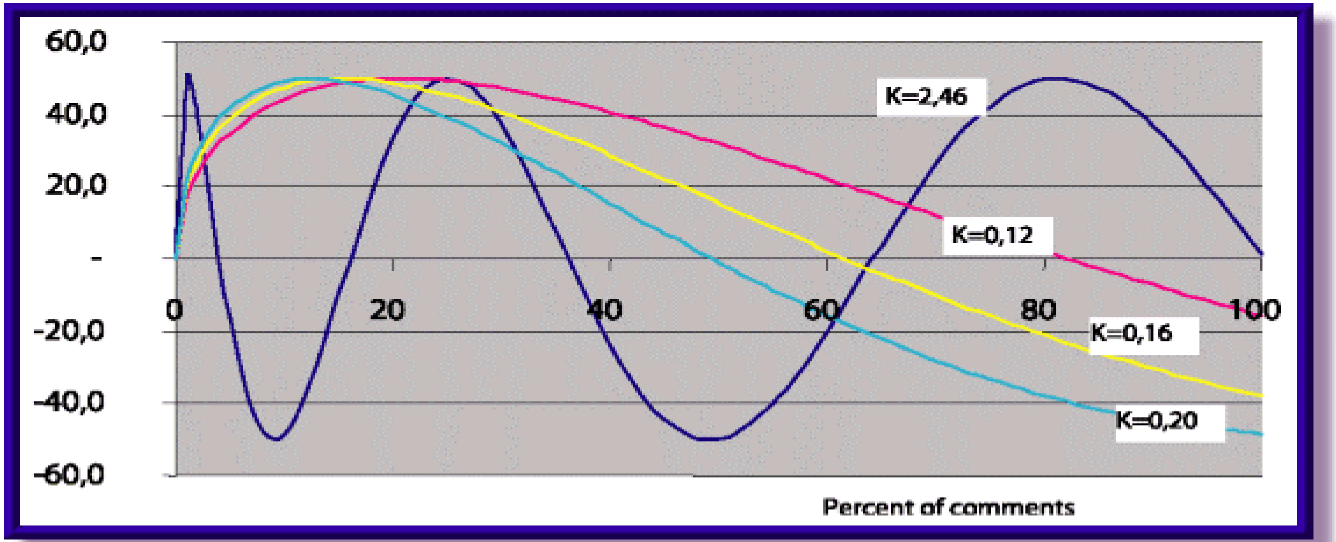
- Halstead's V ≈ 392
- McCabe's v(G) = 5
- LOC = 14
- $MI_1 \approx 96$
- Easy to maintain!

Comments?

$$50 \sin \sqrt{2.46\, perCM}$$

[Liso 2001]





Peaks:

- **25% (OK),**
- **1% and 81% - ???**

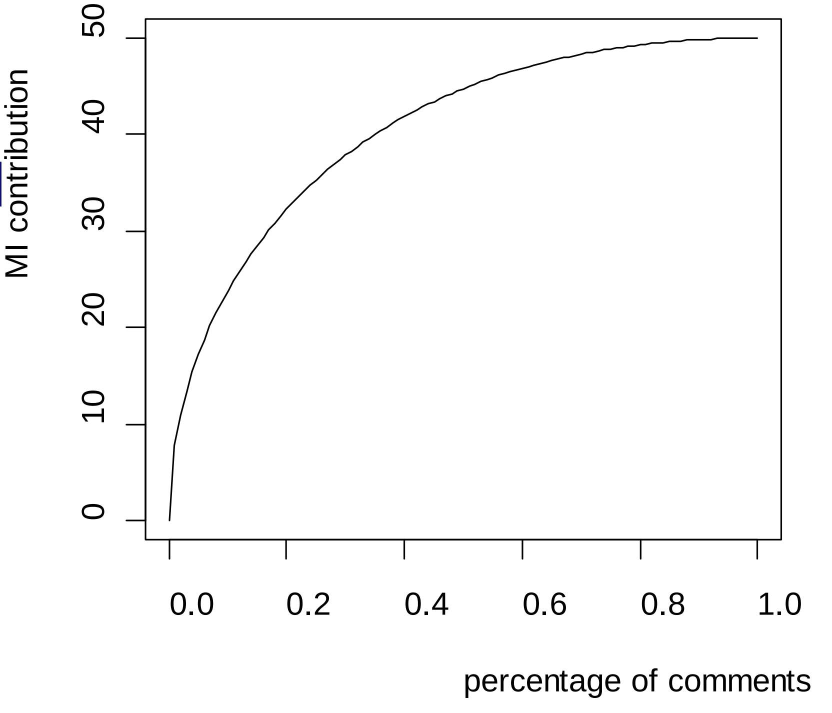
Better:

- **$0.12 \leq K \leq 0.2$**

Another alternative:

- **Percentage as a fraction [0;1] – [Thomas 2008, Ph.D. thesis]**

- **The more comments – the better?**



Technische Universiteit Eindhoven University of Technology

Evolution of the maintainability index in Linux

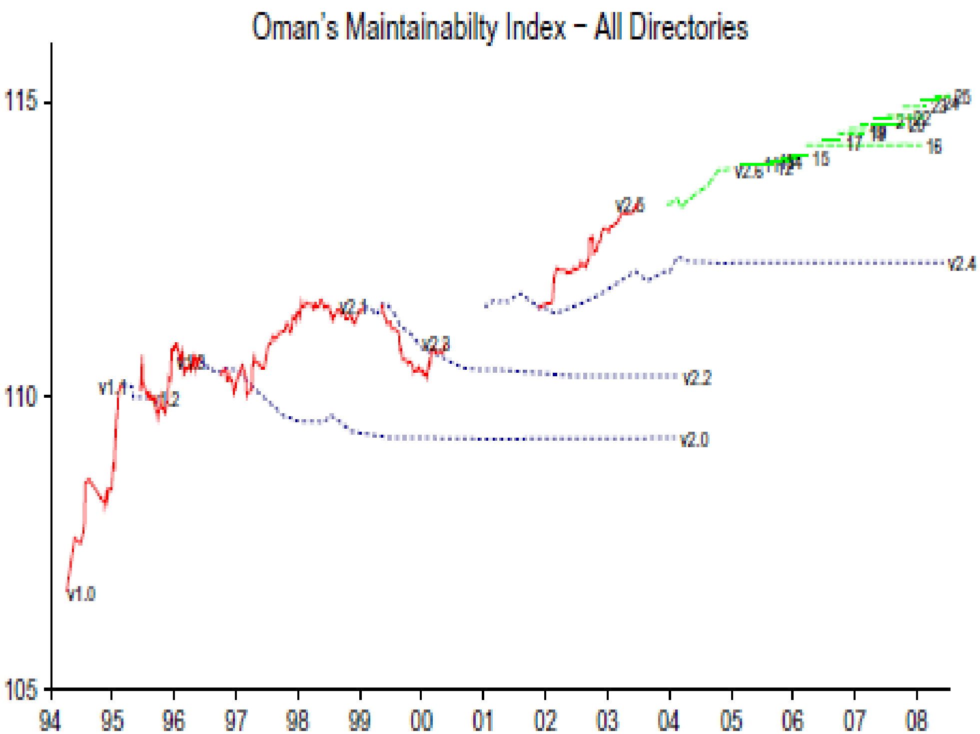


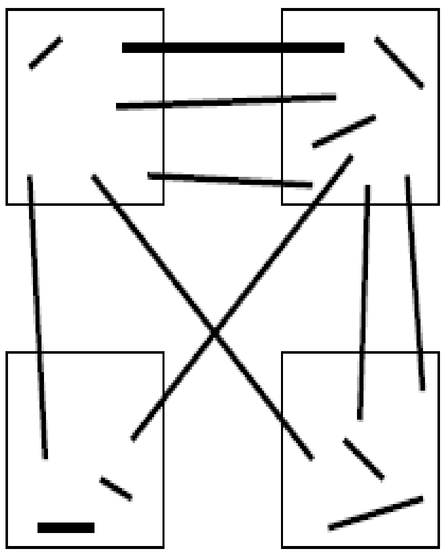
Fig. 9. Evolution of Oman's maintainability index.

A. Israeli, D.G. Feitelson 2010

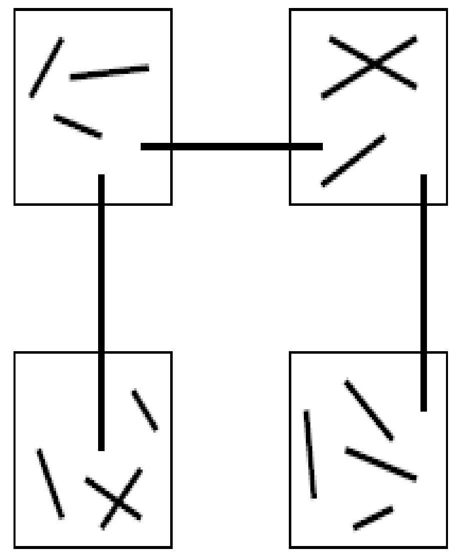
- **Size, Halstead volume and McCabe complexity decrease**
- **% comments decreases as well**
  - **BUT they use the [0;1] definition, so the impact is limited**

What about modularity?

Design A



Design B

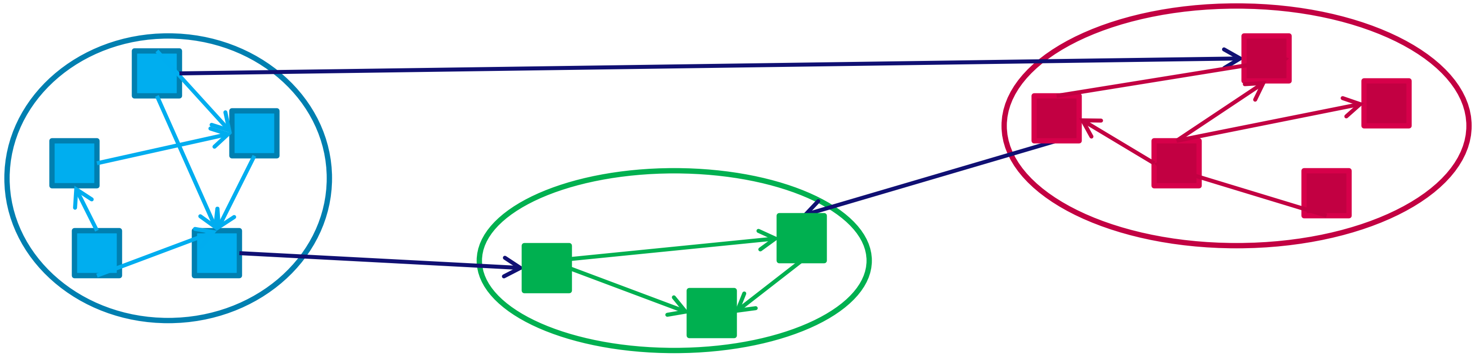


- **Cohesion: calls inside the module**
- **Coupling: calls between the modules**

	A	B
Cohesion	Lo	Hi
Coupling	Hi	Lo

- **Squares are modules, lines are calls, ends of the lines are functions.**
- **Which design is better?**

Do you still remember?



- **Many intra-package dependencies: high cohesion**

$$A_i = \frac{\mu_i}{N_i^2} \quad \text{or} \quad A_i = \frac{\mu_i}{N_i(N_i - 1)}$$

- **Few inter-package dependencies: low coupling**

$$E_{i,j} = \frac{\varepsilon_{i,j}}{2N_i N_j}$$

- **Joint measure**

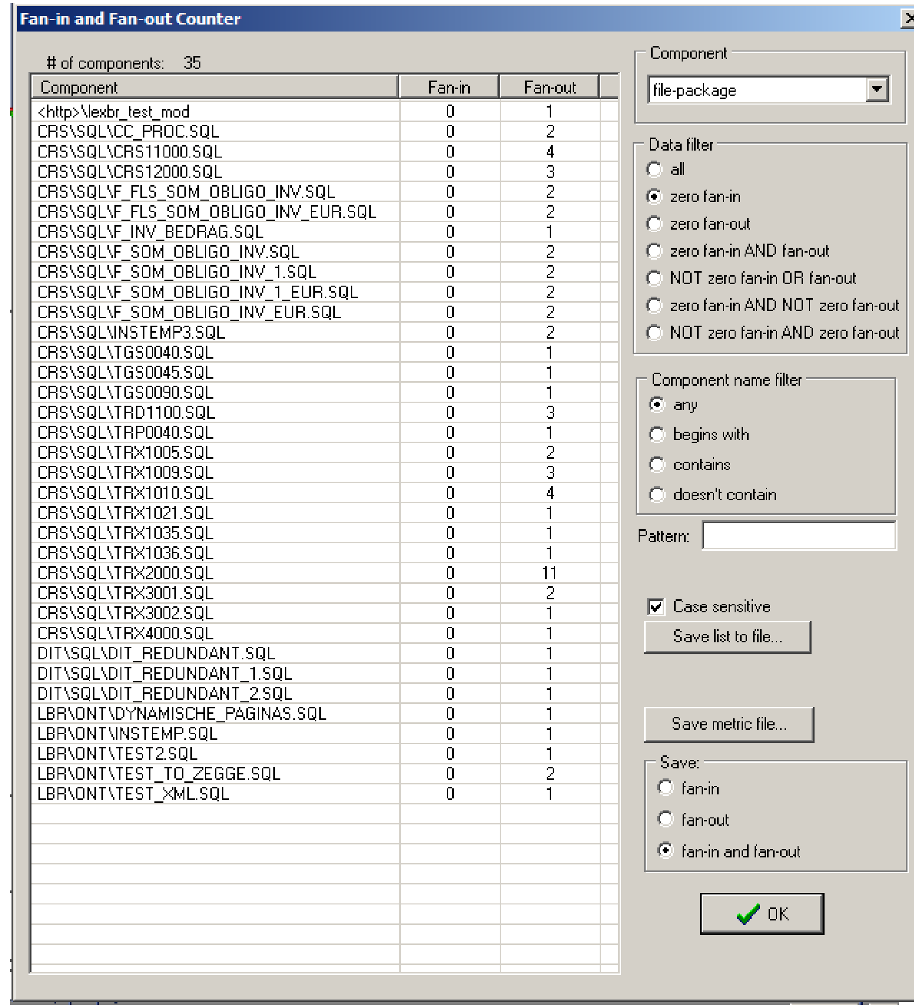
$$MQ = \frac{1}{k}\sum_{i=1}^{k} A_i - \frac{2}{k(k-1)}\sum_{i=1}^{k-1}\sum_{j=i+1}^{k} E_{i,j}$$

k **- Number of packages**

Modularity metrics: Fan-in and Fan-out

- **Fan-in** of M: number of modules calling functions in M
- **Fan-out** of M: number of modules called by M

- Modules with fan-in = 0
- What are these modules?
    - Dead-code
    - Outside of the system boundaries
    - Approximation of the “call” relation is imprecise



Component	Fan-in	Fan-out
<http>\lexbr_test_mod	0	1
CRS\SQL\CC_PROC.SQL	0	2
CRS\SQL\CRS11000.SQL	0	4
CRS\SQL\CRS12000.SQL	0	3
CRS\SQL\F_FLS_SOM_OBLIGO_INV.SQL	0	2
CRS\SQL\F_FLS_SOM_OBLIGO_INV_EUR.SQL	0	2
CRS\SQL\F_INV_BEDRAG.SQL	0	1
CRS\SQL\F_SOM_OBLIGO_INV.SQL	0	2
CRS\SQL\F_SOM_OBLIGO_INV_1.SQL	0	2
CRS\SQL\F_SOM_OBLIGO_INV_1_EUR.SQL	0	2
CRS\SQL\F_SOM_OBLIGO_INV_EUR.SQL	0	2
CRS\SQL\INSTEMP3.SQL	0	2
CRS\SQL\TGS0040.SQL	0	1
CRS\SQL\TGS0045.SQL	0	1
CRS\SQL\TGS0090.SQL	0	1
CRS\SQL\TRD1100.SQL	0	3
CRS\SQL\TRP0040.SQL	0	1
CRS\SQL\TRX1005.SQL	0	2
CRS\SQL\TRX1009.SQL	0	3
CRS\SQL\TRX1010.SQL	0	4
CRS\SQL\TRX1021.SQL	0	1
CRS\SQL\TRX1035.SQL	0	1
CRS\SQL\TRX1036.SQL	0	1
CRS\SQL\TRX2000.SQL	0	11
CRS\SQL\TRX3001.SQL	0	2
CRS\SQL\TRX3002.SQL	0	1
CRS\SQL\TRX4000.SQL	0	1
DIT\SQL\DIT_REDUNDANT.SQL	0	1
DIT\SQL\DIT_REDUNDANT_1.SQL	0	1
DIT\SQL\DIT_REDUNDANT_2.SQL	0	1
LBR\ONT\DYNAMISCHE_PAGINAS.SQL	0	1
LBR\ONT\INSTEMP.SQL	0	1
LBR\ONT\TEST2.SQL	0	1
LBR\ONT\TEST_TO_ZEGGE.SQL	0	2
LBR\ONT\TEST_XML.SQL	0	1

Henry and Kafura's information flow complexity [HK 1981]

- **Fan-in and fan-out can be defined for procedures**
  - **HK: take global data structures into account:**
    - **read for fan-in,**
    - **write for fan-out**

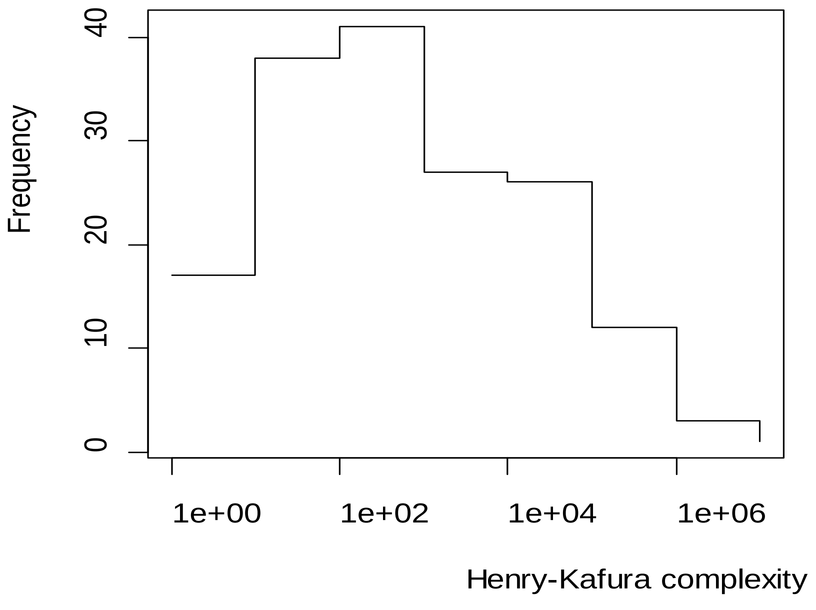
- **Henry and Kafura: procedure as HW component connecting inputs to outputs**

$$hk = sloc * (fanin * fanout)^2$$

- **Shepperd**

$$s = (fanin * fanout)^2$$

Information flow complexity of Unix procedures

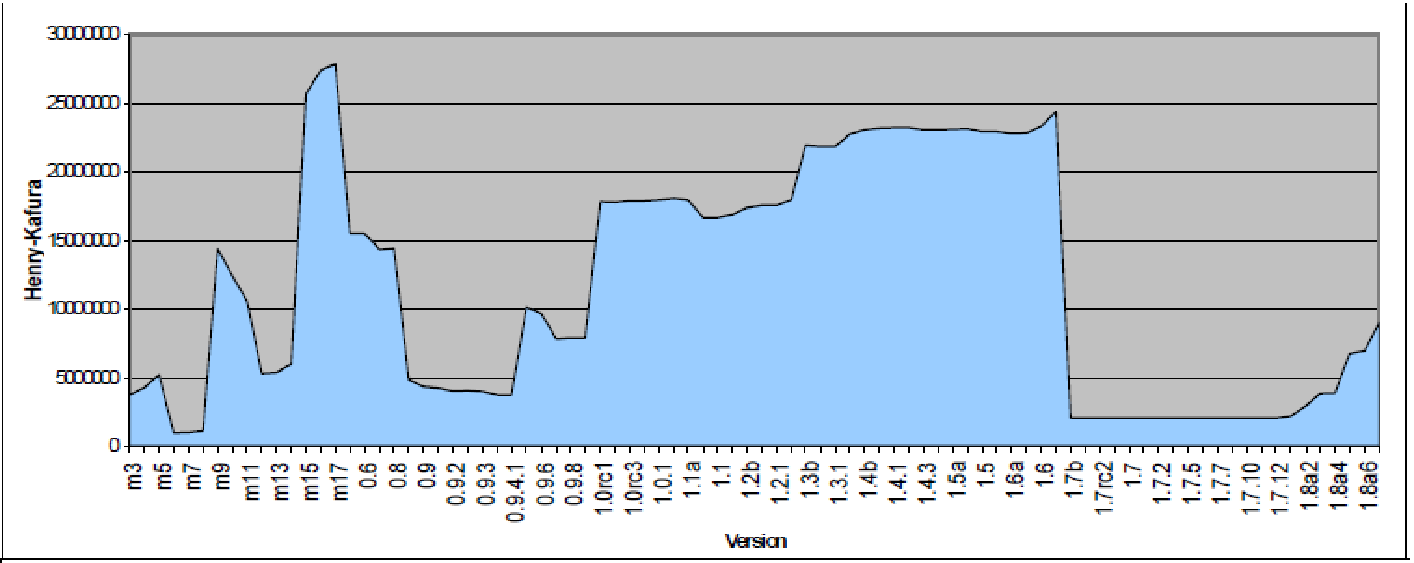


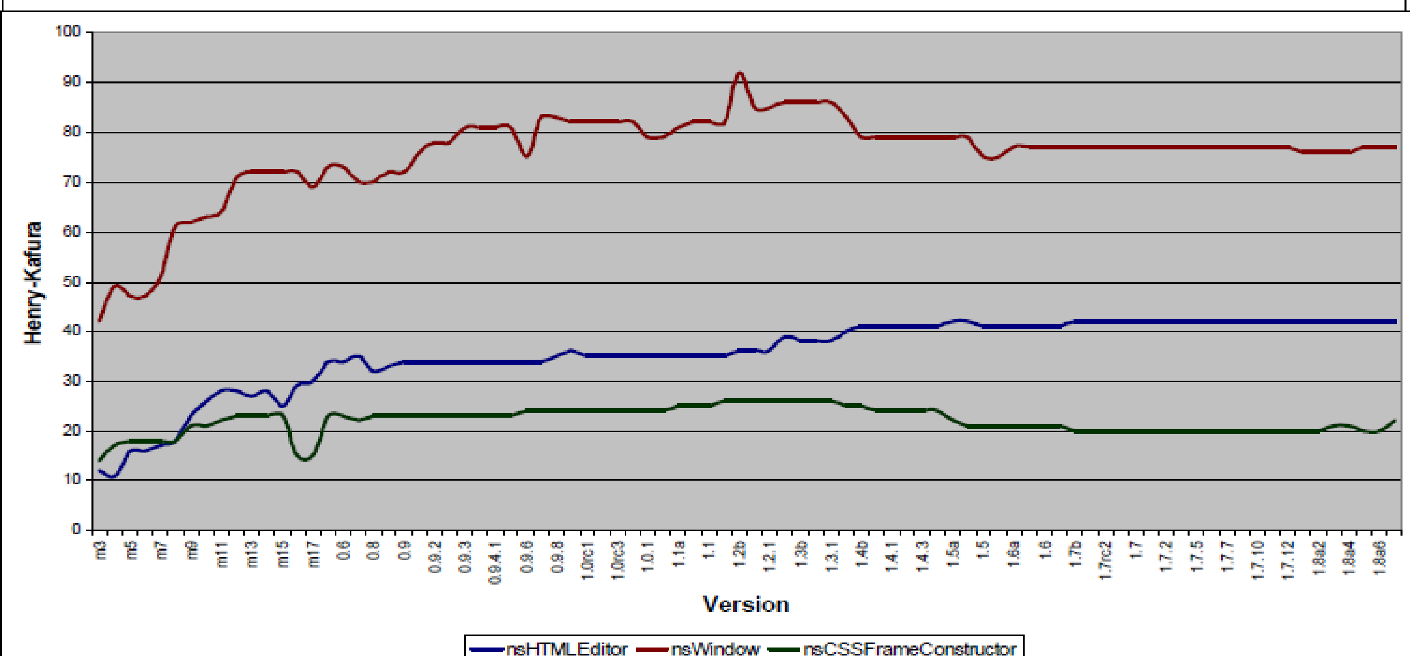
- **Solid – #procedures within the complexity range**
- **Dashed - #changed procedures within the complexity range**

- **Highly complex procedures are difficult to change but they are changed often!**
- **Complexity comes the "most complex" procedures**

Technische Universiteit Eindhoven University of Technology

Evolution of the information flow complexity





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

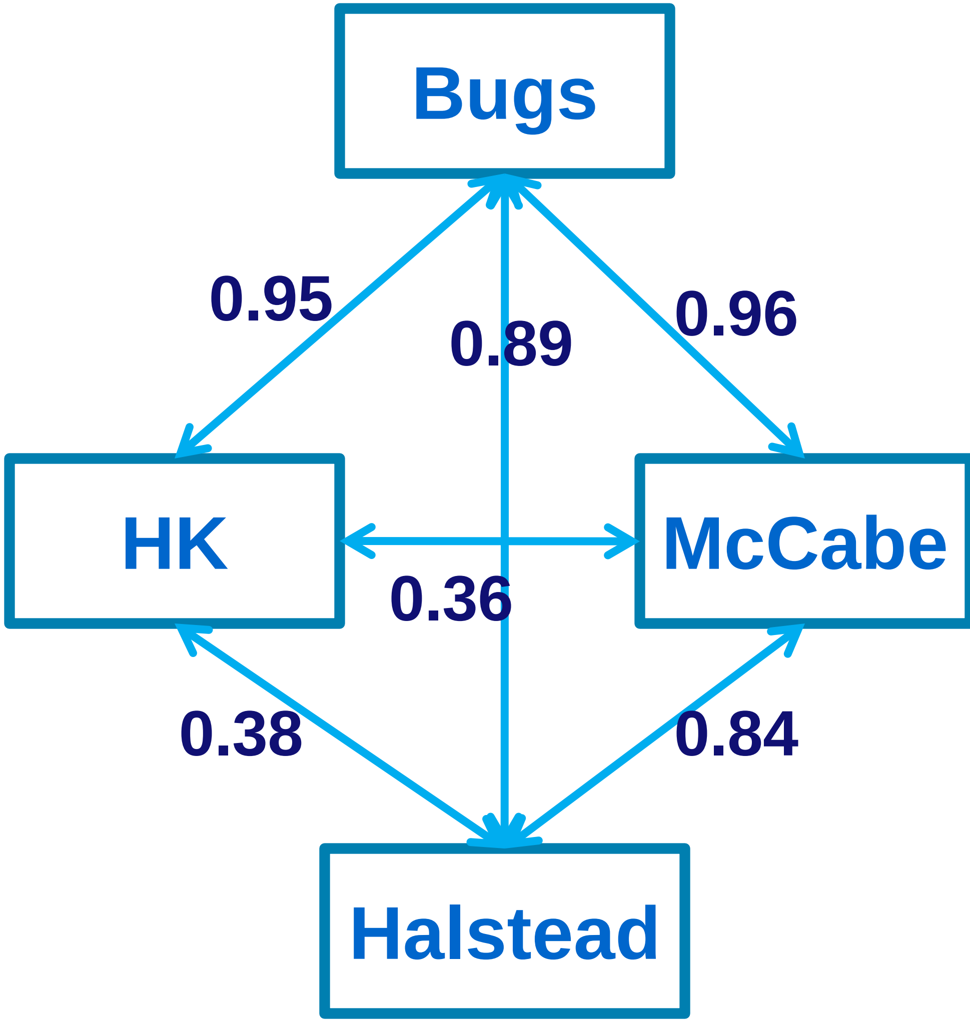
Technische Universiteit Eindhoven University of Technology

Summary so far…

- **Complexity metrics**
  - **Halstead's effort**
  - **McCabe (cyclomatic)**
  - **Henry Kafura/Shepperd (information flow)**

- **Are these related?**
- **And what about bugs?**

- **Harry,Kafura,Harris 1981**
  - **165 Unix procedures**
- **What does this tell us?**



From imperative to OO

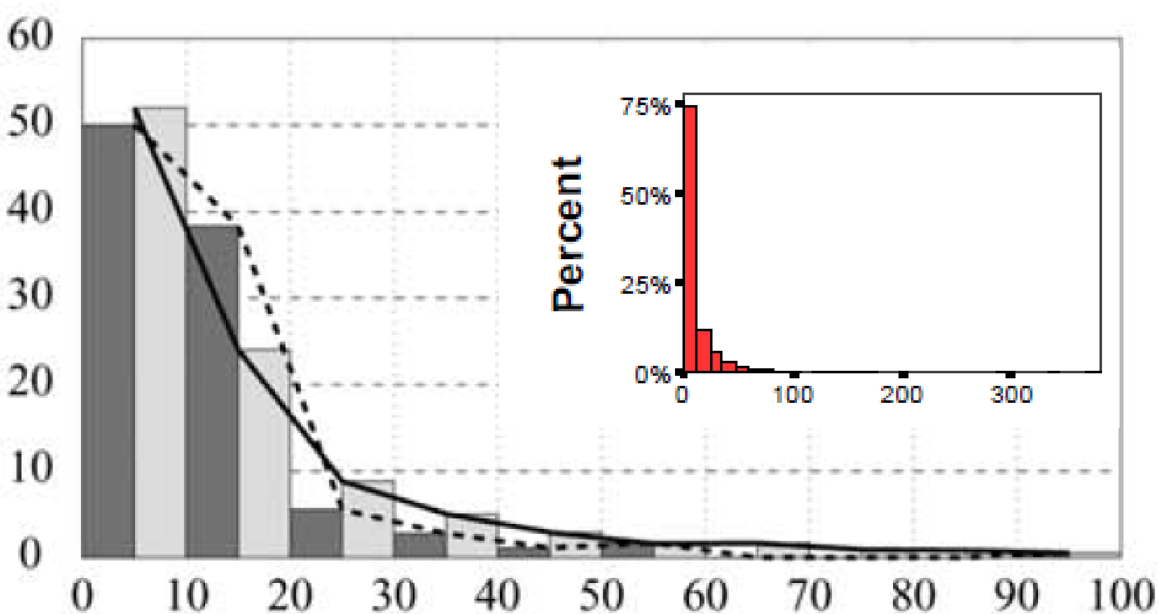
- All metrics so far were designed for imperative languages
  - Applicable for OO
    - On the method level
    - Also
      - Number of files → number of classes/packages
      - Fan-in → afferent coupling (C_a)
      - Fan-out → efferent coupling (C_e)
  - But do not reflect OO-specific complexity
    - Inheritance, class fields, abstractness, …
- Popular metric sets
  - Chidamber and Kemerer, Li and Henry, Lorenz and Kidd, Abreu, Martin

Chidamber and Kemerer

- **WMC – weighted methods per class**
  - **Sum of metrics(m) for all methods m in class C**
- **DIT – depth of inheritance tree**
  - **java.lang.Object? Libraries?**
- **NOC – number of children**
  - **Direct descendents**
- **CBO – coupling between object classes**
  - **A is coupled to B if A uses methods/fields of B**
  - **CBO(A) = | {B|A is coupled to B} |**
- **RFC - #methods that can be executed in response to a message being received by an object of that class.**

Chidamber and Kemerer

- **WMC – weighted methods per class**
  - Sum of metrics(m) for all methods m in class C
  - Popular metrics: McCabe's complexity and unity
  - WMC/unity = number of methods
  - Statistically significant correlation with the number of defects

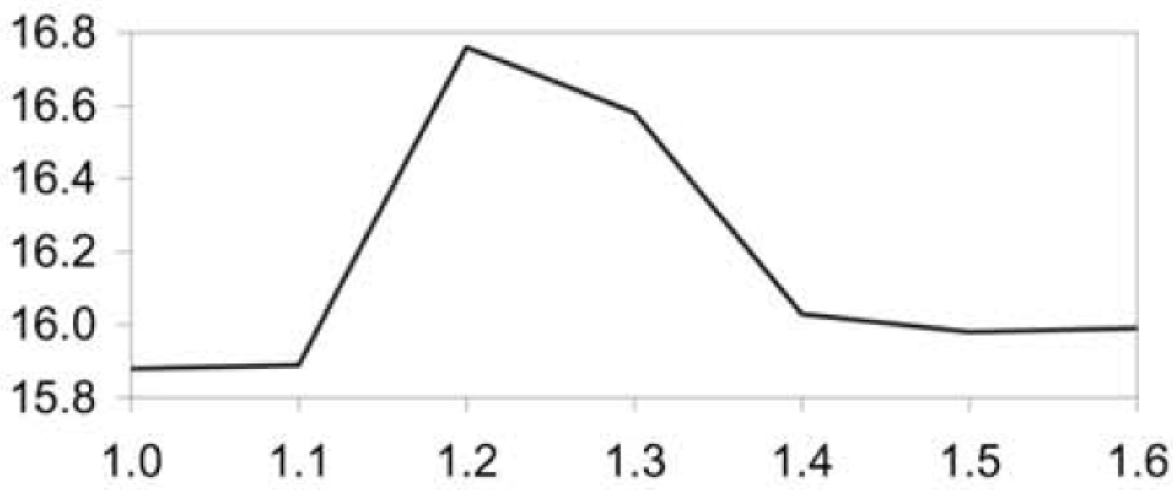


- WMC/unity
- Dark: Basili et al.
- Light: Gyimothy et al. [Mozilla 1.6]
- Red: High-quality NASA system

Technische Universiteit Eindhoven University of Technology

Chidamber and Kemerer

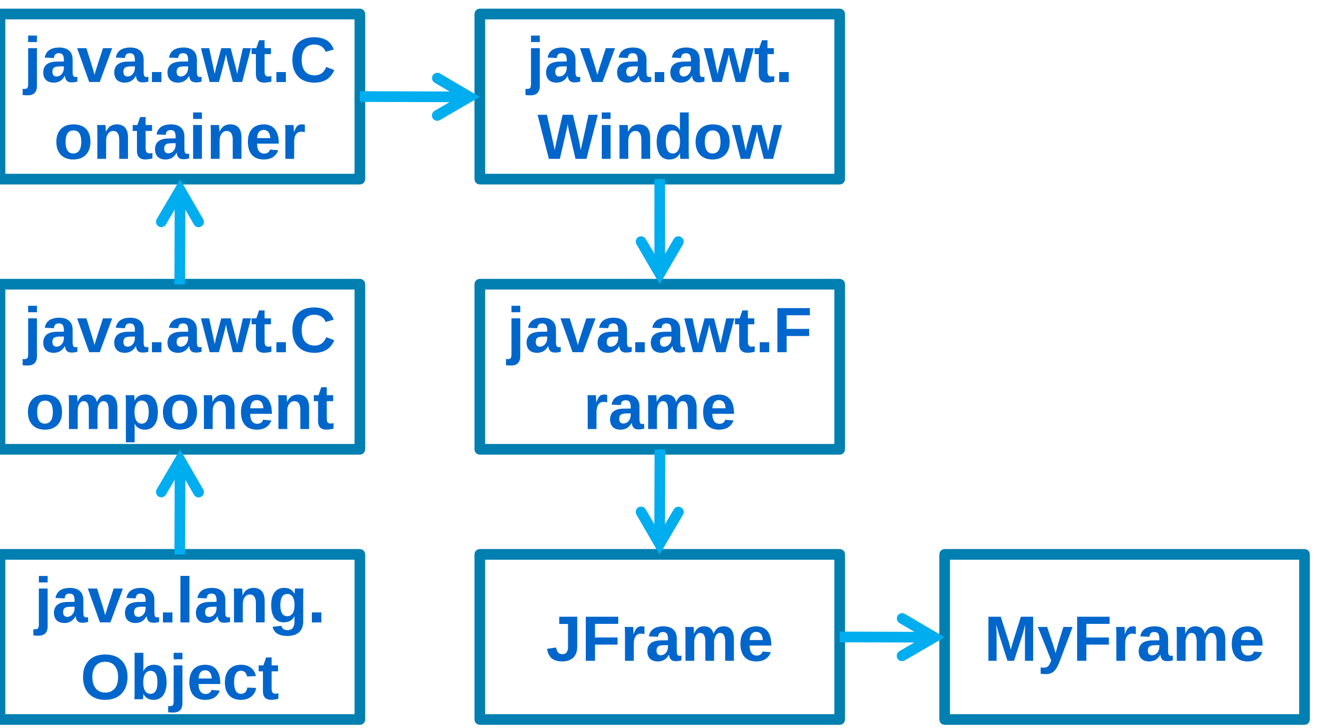
- **WMC – weighted methods per class**
  - **Sum of metrics(m) for all methods m in class C**
  - **Popular metrics: McCabe's complexity and unity**
  - **WMC/unity = number of methods**
  - **Statistically significant correlation with the number of defects**



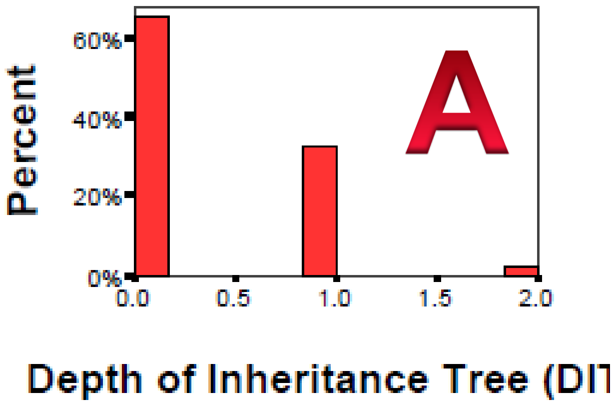
- **WMC/unity**
- **Gyimothy et al.**
- **Average**

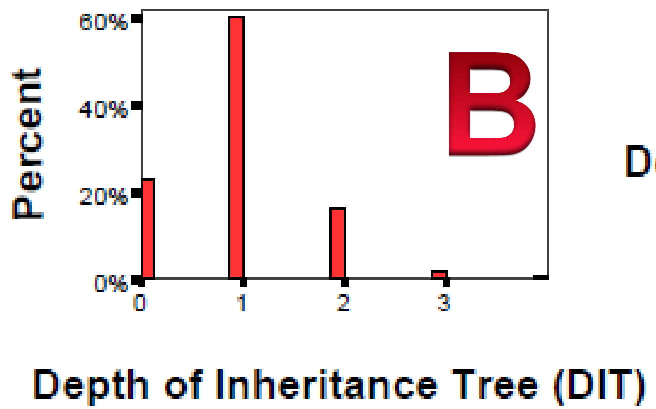
Depth of inheritance - DIT

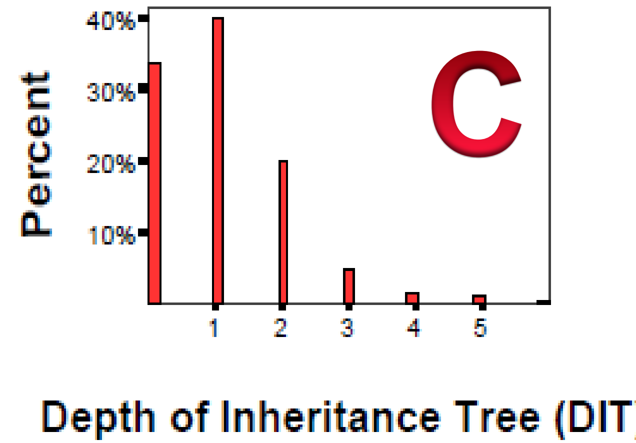
- Variants: Were to start and what classes to include?
  - 1, JFrame is a library class, excluded
  - 2, JFrame is a library class, included
  - 7



DIT – what is good and what is bad?

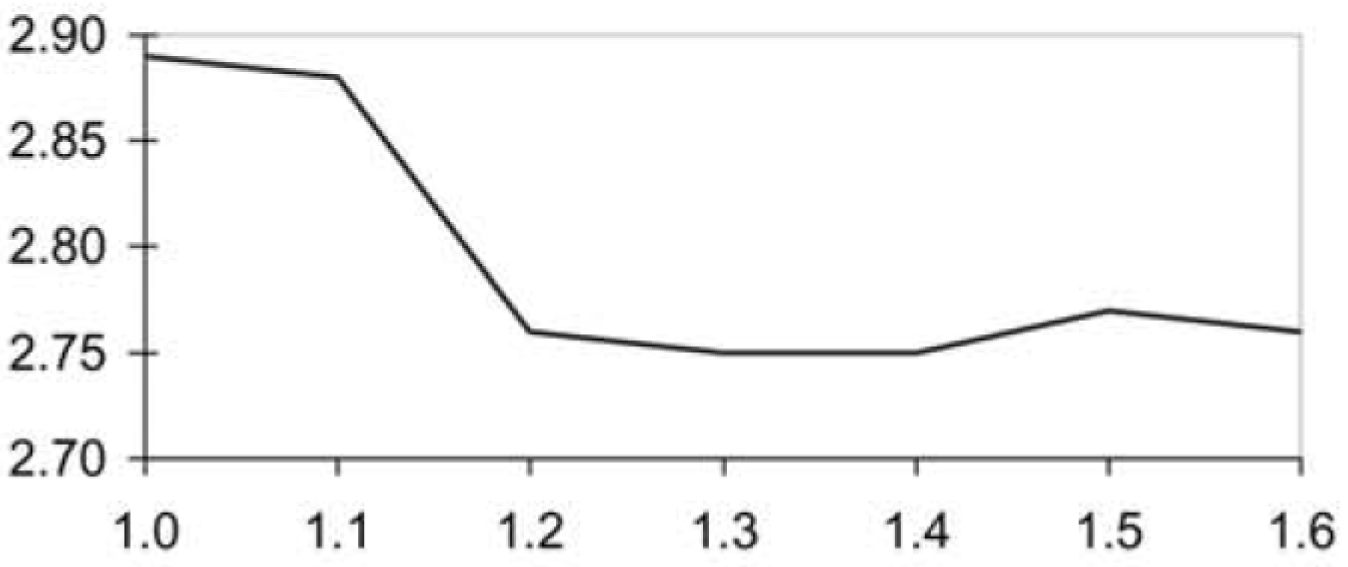






- **Three NASA systems**
- **What can you say about the use of inheritance in systems A, B and C?**
- **Observation: quality assessment depends not just on one class but on the entire distribution**

Average DIT in Mozilla



- **How can you explain the decreasing trend?**

Other CK metrics

- **NOC – number of children**
- **CBO – coupling between object classes**
- **RFC - #methods that can be executed in response to a message being received by an object of that class.**
- **More or less “exponentially” distributed**

Metric	Our results	[1]	[22]	[21]
WMC	++	+	++	++
DIT	+	++	0	−
RFC	++	++	+	
NOC	0	++	−−	
CBO	++	+	+	+

Significance of CK metrics to predict the number of faults

Modularity metrics: LCOM

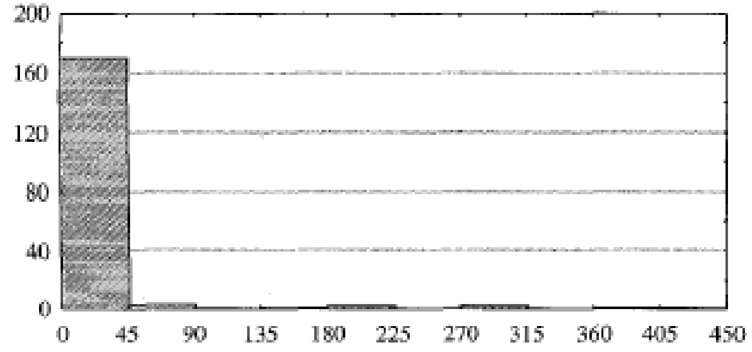
- **LCOM – lack of cohesion of methods**

- **Chidamber Kemerer:**

$$LCOM(C) = \begin{cases} P - Q & \text{if } P > Q \\ 0 & \text{otherwise} \end{cases}$$

where

- **P = #pairs of distinct methods in C that do not share variables**
- **Q = #pairs of distinct methods in C that share variables**



[BBM] 180 classes

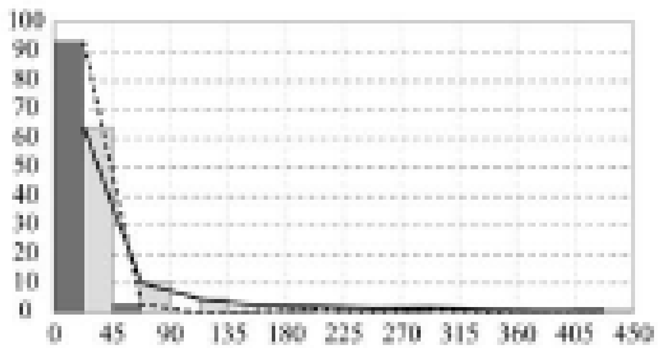
Discriminative ability is insufficient

What about get/set?

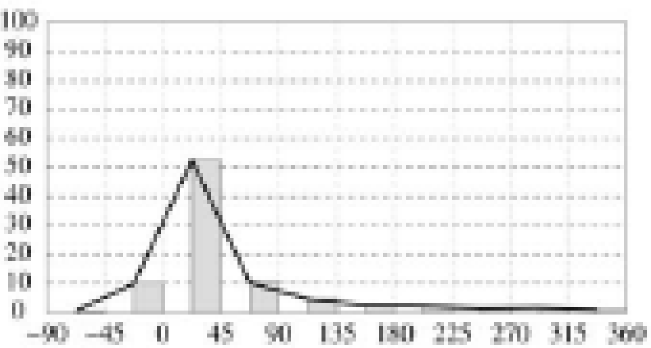
First solution: LCOMN

- **Defined similarly to LCOM but allows negative values**

$$LCOMN(C) = P - Q$$

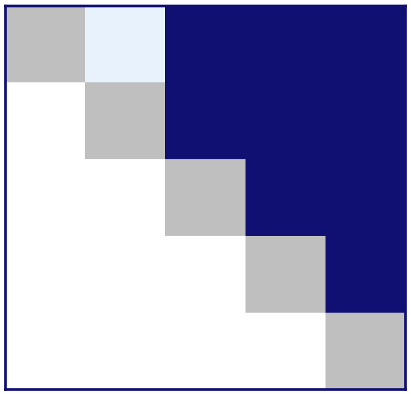


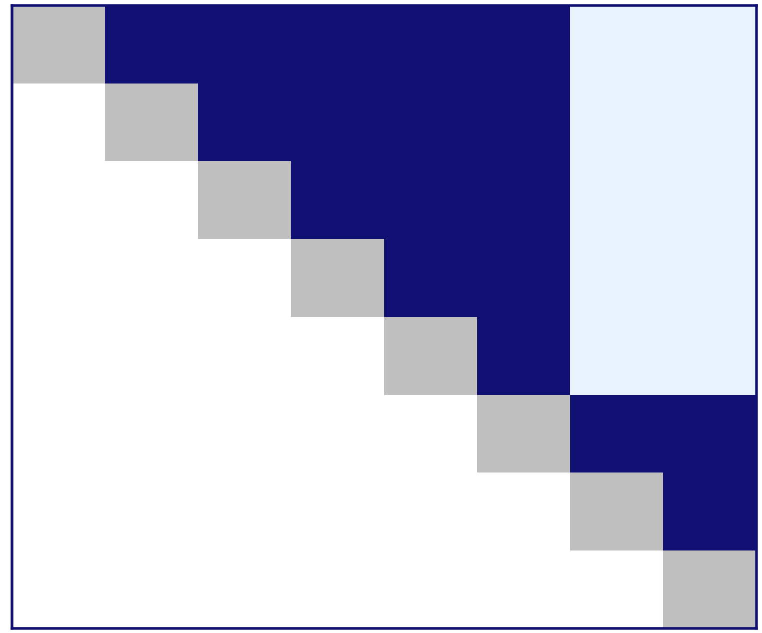
LCOM



LCOMN

Still…





- **Method * method tables**
  - **Light blue: Q, dark blue: P**
- **Calculate the LCOMs**
- **Does this correspond to your intuition?**

Technische Universiteit Eindhoven University of Technology

Henderson-Sellers, Constantine and Graham 1996

- **m – number of methods**
- **v – number of variables (attrs)**
- **$m(V_i)$ - #methods that access V_i**

$$\frac{\left(\frac{1}{v}\sum_{i=1}^{v} m(V_i)\right) - m}{1-m}$$

- **Cohesion is maximal: all methods access all variables**

  $m(V_i) = m$ **and** $LCOM = 0$

- **No cohesion: every method accesses a unique variable**

  $m(V_i) = 1$ **and** $LCOM = 1$

- **Can LCOM exceed 1?**

LCOM > 1?

- **If some variables are not accessed at all, then**

$$m(V_i) = 0$$

and

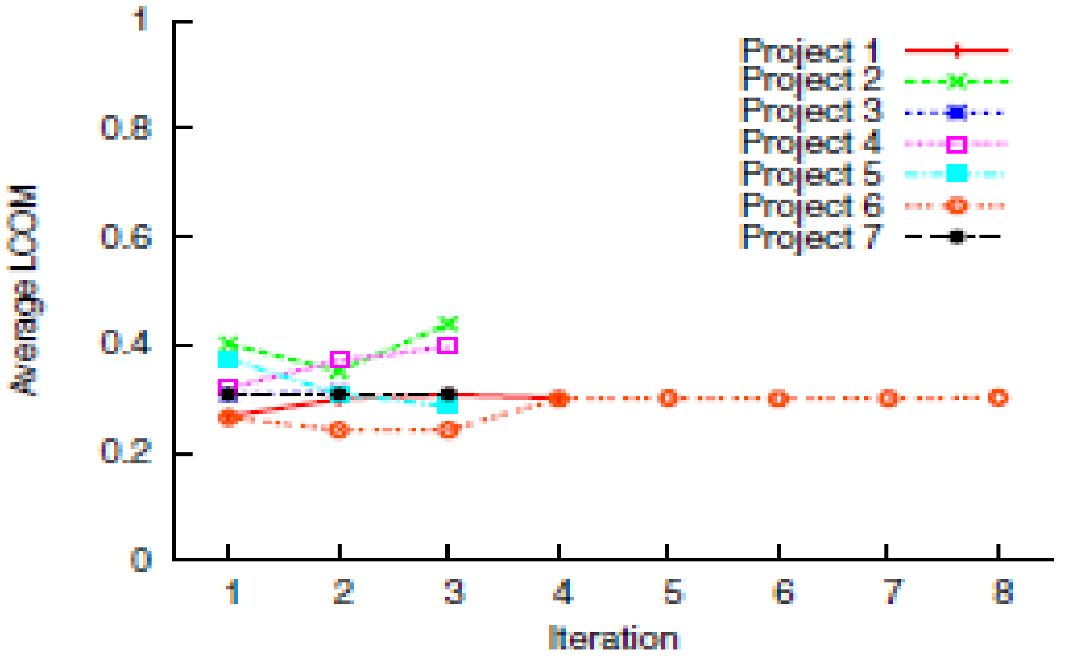
$$\frac{\left(\frac{1}{v}\sum_{i=1}^{v} m(V_i)\right) - m}{1-m} = \frac{-m}{1-m} = 1 + \frac{1}{m-1}$$

Hence

LCOM is undefined for m = 1

LCOM ≤ 2

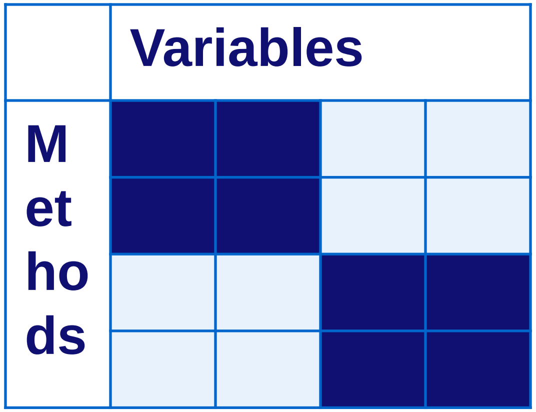
Evolution of LCOM [Henderson-Sellers et al.]

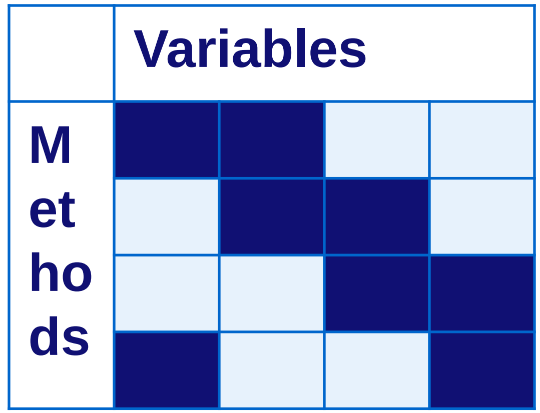


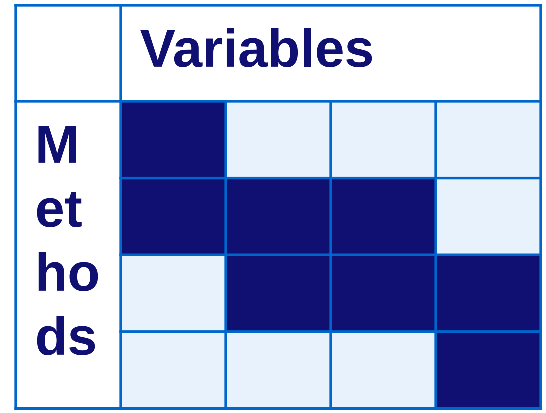
Sato, Goldman, Kon 2007

- **Project 6 (commercial human resource system) suggests stabilization, but no similar conclusion can be made for other projects**

Shortcomings of LCOM [Henderson-Sellers]







- **Due to [Fernández, Peña 2006]**
- **Method-variable diagrams: dark spot = access**
- **LCOM( ) = LCOM( ) = LCOM( ) = 0.67**

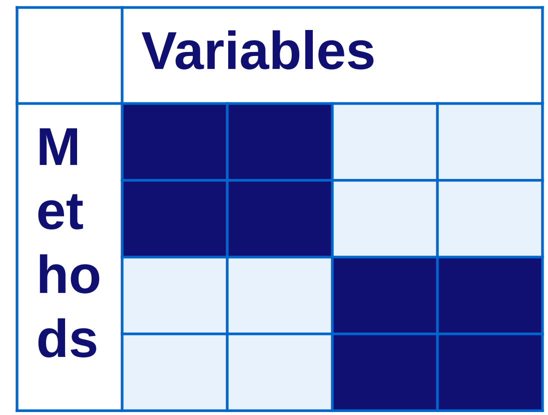
  seems to be less cohesive than and !

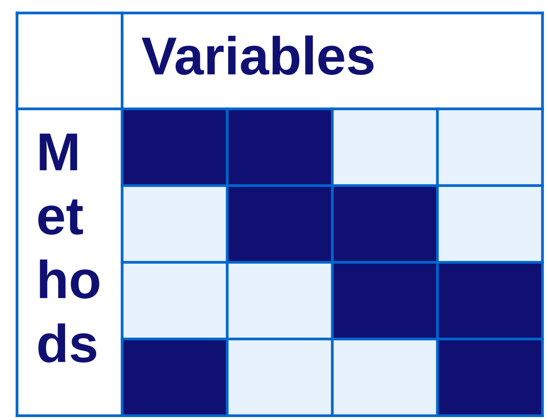
Alternative [Hitz, Montazeri 1995]

- **LCOM as the number of strongly connected components in the following graph**
  - **Vertices: methods**
  - **Edge between a and b, if**
    - **a calls b**
    - **b calls a**
    - **a and b access the same variable**

- **LCOM values**
  - **0, no methods**
  - **1, cohesive component**
  - **2 or more, lack of cohesion**

Question: LCOM?





Experimental evaluation of LCOM variants

Cox, Etzkorn and Hughes 2006	Correlation with expert assessment	
	Group 1	Group 2
Chidamber Kemerer	-0.43 (p = 0.12)	-0.57 (p = 0.08)
Henderson-Sellers	-0.44 (p = 0.12)	-0.46 (p = 0.18)
Hitz, Montazeri	-0.47 (p = 0.06)	-0.53 (p = 0.08)

Etzkorn, Gholston, Fortune, Stein, Utley, Farrington, Cox	Correlation with expert assessment	
	Group 1	Group 2
Chidamber Kemerer	-0.46 (rating 5/8)	-0.73 (rating 1.5/8)
Henderson-Sellers	-0.44 (rating 7/8)	-0.45 (rating 7/8)
Hitz, Montazeri	-0.51 (rating 2/8)	-0.54 (rating 5/8)

LCC and TCC [Bieman, Kang 1994]

- Recall: LCOM HM “a and b access the same variable”
- What if a calls a’, b calls b’, and a’ and b’ access the same variable?
- Metrics
  - NDP – number of pairs of methods directly accessing the same variable
  - NIP – number of pairs of methods directly or indirectly accessing the same variable
  - NP – number of pairs of methods: n(n-1)/2
- Tight class cohesion TCC = NDP/NP
- Loose class cohesion LCC = NIP/NP
- NB: Constructors and destructors are excluded

Experimental evaluation of LCC/TCC

Etzkorn, Gholston, Fortune, Stein, Utley, Farrington, Cox	Correlation with expert assessment	
	Group 1	Group 2
Chidamber Kemerer	-0.46 (rating 5/8)	-0.73 (rating 1.5/8)
Henderson-Sellers	-0.44 (rating 7/8)	-0.45 (rating 7/8)
Hitz, Montazeri	-0.51 (rating 2/8)	-0.54 (rating 5/8)
TCC	-0.22 (rating 8/8)	-0.057 (rating 8/8)
LCC	-0.54 (rating 1/8)	-0.73 (rating 1.5/8)

Conclusions: Metrics so far…

Level	Matrics
Method	LOC, McCabe, Henry Kafura
Class	WMC, NOC, DIT, LCOM (and variants), LCC/TCC
Packages	???

Next time:

- **Package-level metrics (Martin)**
- **Metrics of change**