2IMP25 Software Evolution

Software metrics

Alexander Serebrenik

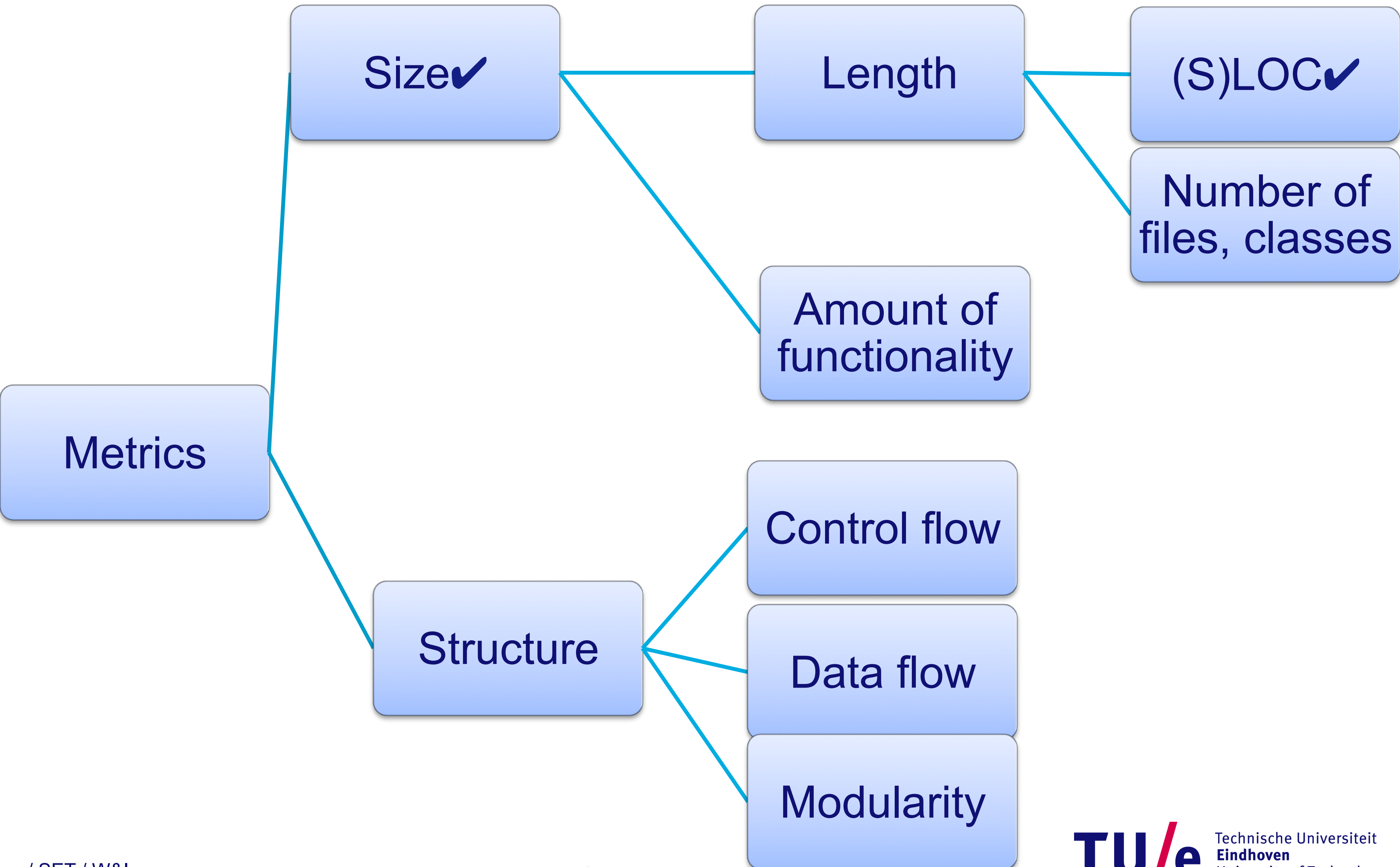
TU/e Technische Universiteit Eindhoven University of Technology

Where innovation starts

Assessement

- **Assignment 2: Thank you for submitting!**
- **Assignment 3: March 30, 23:59**
- **Exam: April 14, 9:00-12:00**
  - **Mockup exam is online**

Classification of metrics [à la Fenton, Pfleeger 1996]



TU/e Technische Universiteit Eindhoven University of Technology

Length: #components

- Number of files, classes, packages
- Intuitive: "number of volumes in an encyclopaedia"

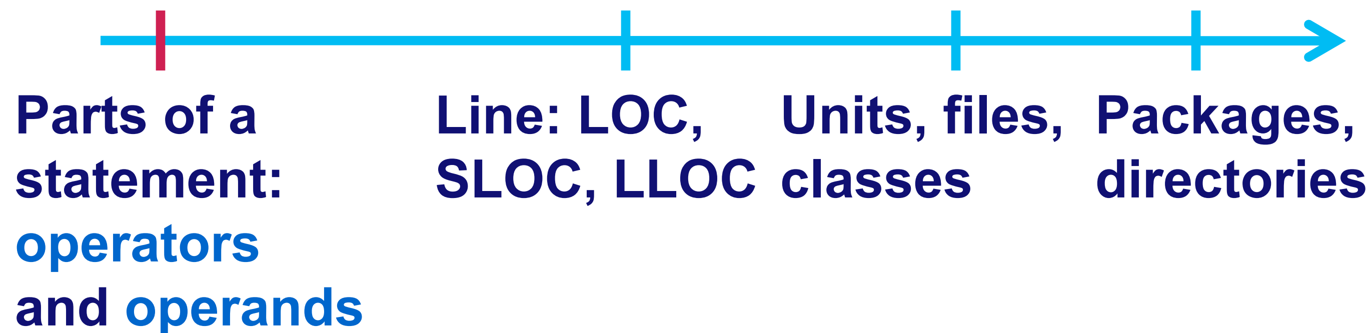
- Variants:
  - All files, classes, packages
  - No empty/library/third-party files, classes, packages
  - No nested/inner classes
  - No or only some auxiliary files (makefiles, header files)

- Correlation with the #post-release defects [Nagappan, Ball, Zeller 2006]
  - significant for modules A, B, C (strength:0.5-0.7), insignificant for modules D, E
  - for each module correlation with some other metrics!

TU/e Technische Universiteit Eindhoven University of Technology

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:
    - $20 \leq$ Volume(function) ≤ 1000
    - $100 \leq$ Volume(file) ≤ 8000

TU/e Technische Universiteit Eindhoven University of Technology

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

Operators:

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

Operands:

```
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]

TU/e Technische Universiteit Eindhoven University of Technology

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

TU/e 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)

TU/e Technische Universiteit Eindhoven University of Technology

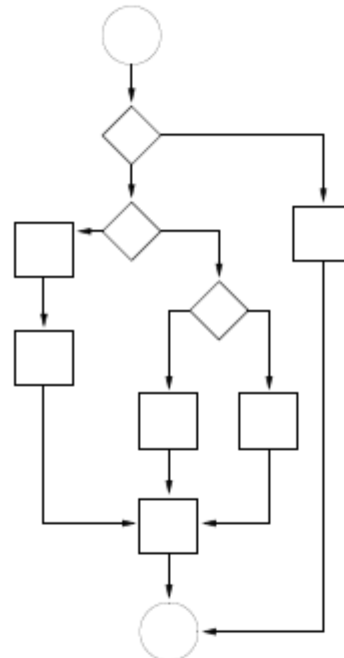
Structural complexity

- Structural complexity:
  - Control flow
  - Data flow

Commonly represented as graphs

Graph-based metrics

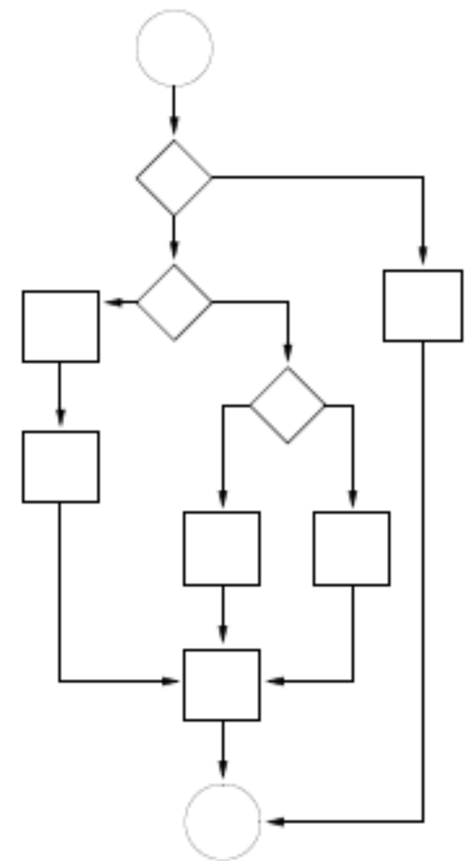
  - Modularity



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

TU/e Technische Universiteit Eindhoven University of Technology

McCabe’s 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) ≤ 15
- v(file) ≤ 100

McCabe's 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

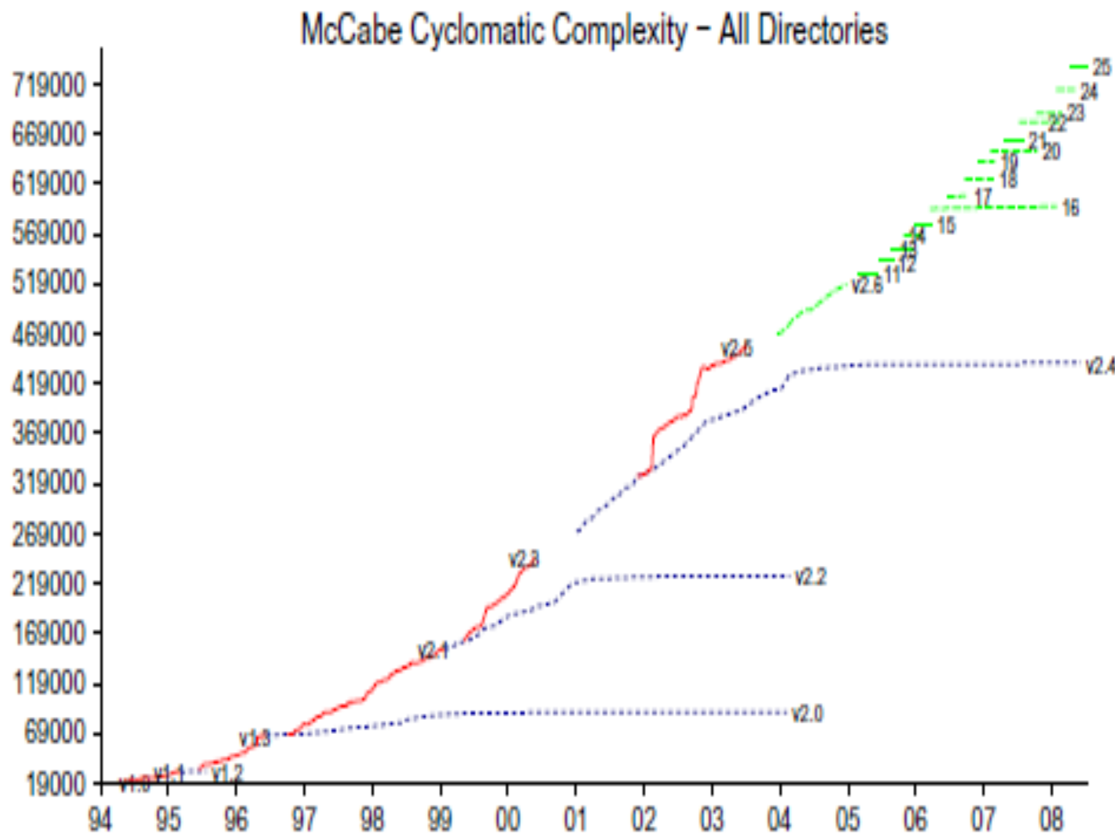
TU/e Technische Universiteit Eindhoven University of Technology

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

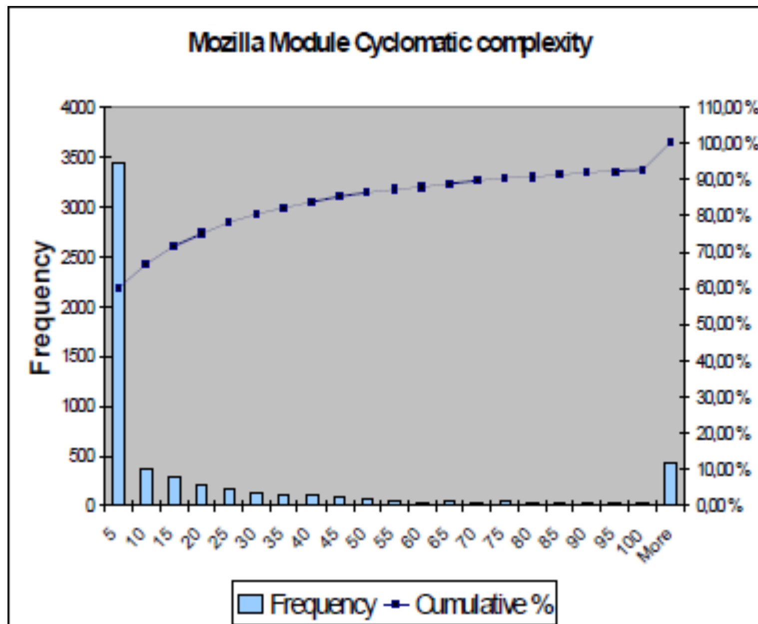


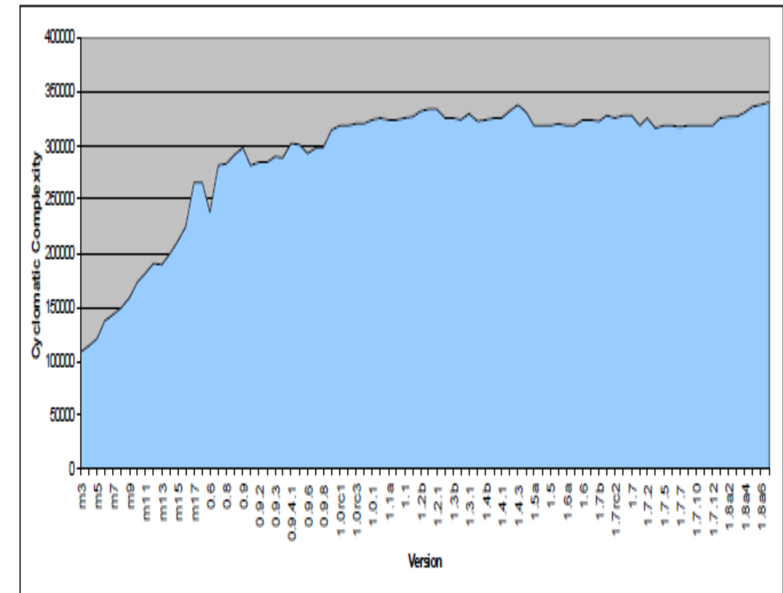
A. Israeli, D.G. Feitelson 2010

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

TU/e Technische Universiteit Eindhoven University of Technology

McCabe's complexity in Mozilla [Røsdal 2005]



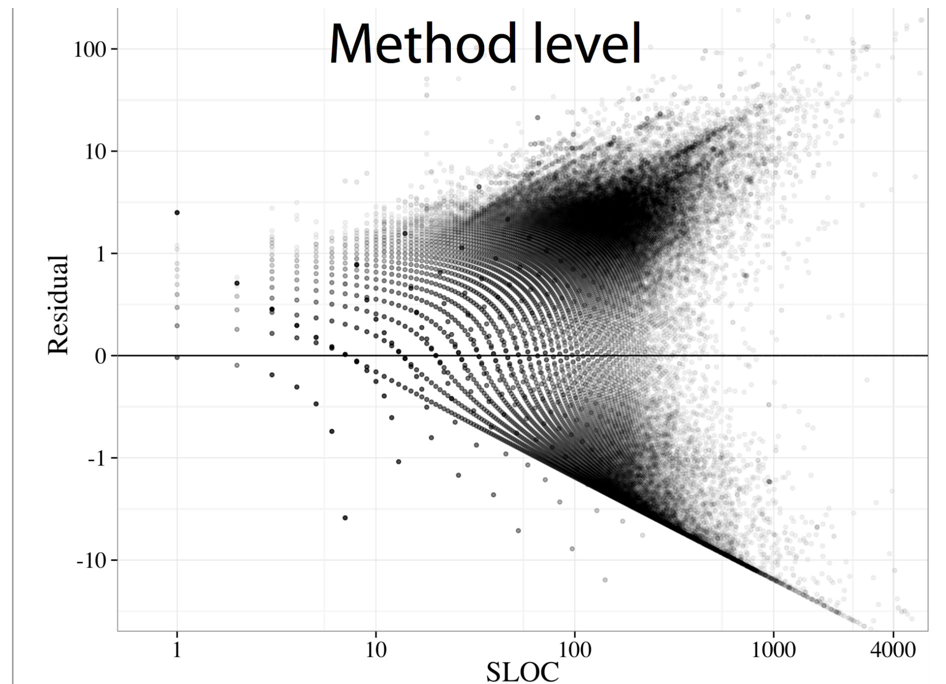


- Most of the modules have low cyclomatic complexity
- Complexity of the system seems to stabilize

TU/e Technische Universiteit Eindhoven University of Technology

CC vs SLOC

- **Strong correlation is often claimed**
- **Problematic** [Landman,Serebrenik,Vinju 2014]
  - **17M Java methods: R^2 ~ 0.43**
  - **Even less for larger methods**
  - **Huge variance**



TU/e Technische Universiteit Eindhoven University of Technology

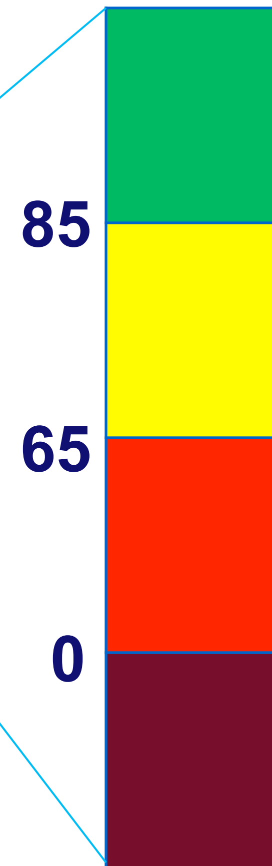
Summarizing: Maintainability index (MI) [Coleman, Oman 1994]

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

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 middle-sized systems!

TU/e Technische Universiteit Eindhoven University of Technology

McCabe complexity: Example

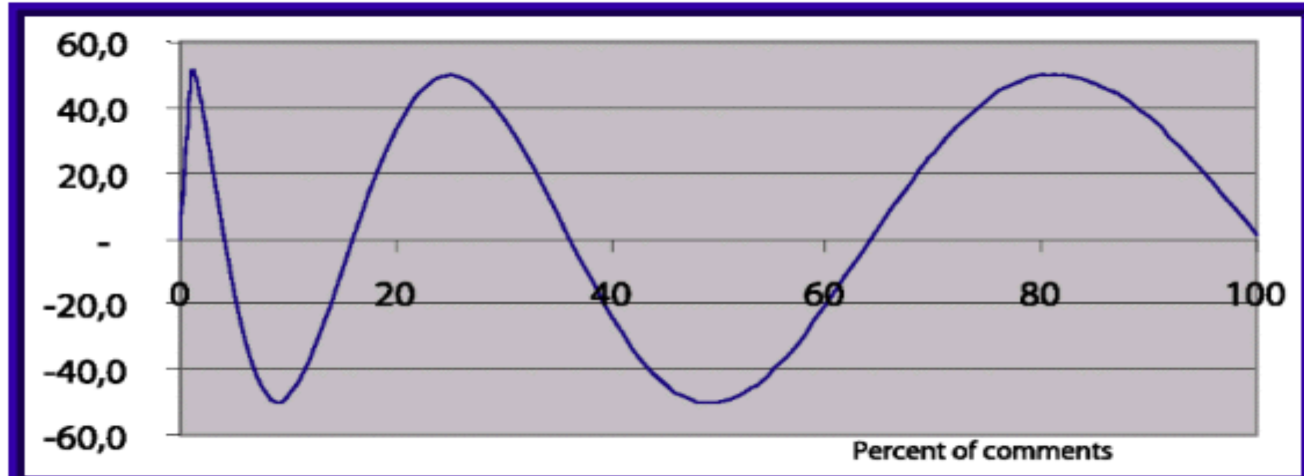
```
void sort ( int *a, int n ) {
int i, j, t;

  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 ≈ 96
- Easy to maintain!

Comments?

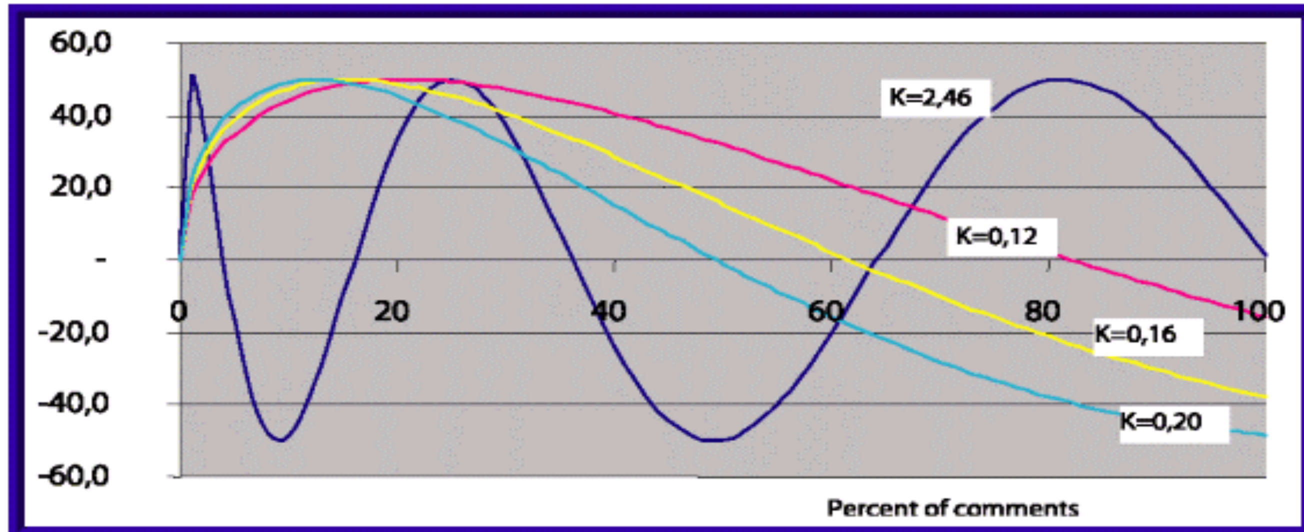
$$50\sin\sqrt{2.46\,perCM}$$

[Liso 2001]



Peaks:

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



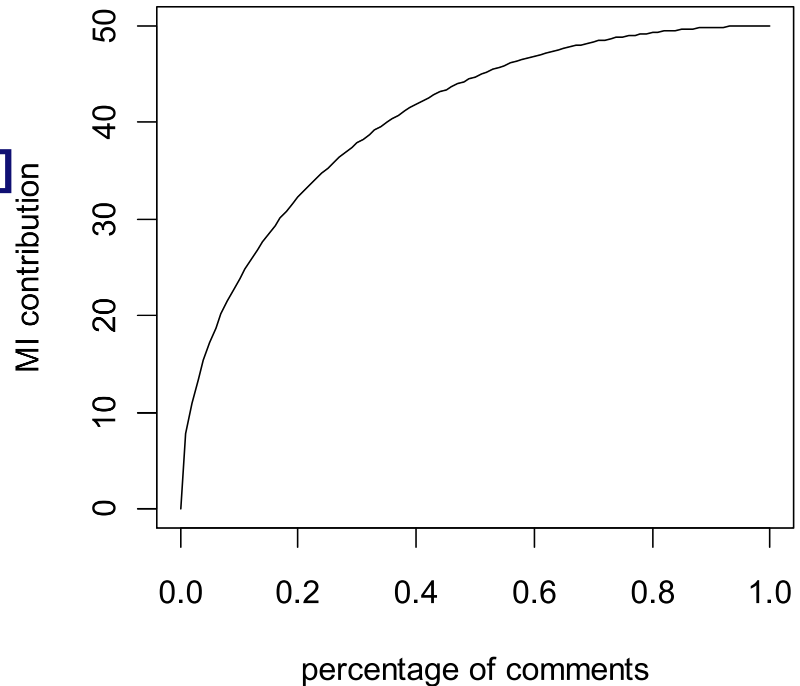
Better:

- $0.12 \leq K \leq 0.2$

TU/e Technische Universiteit Eindhoven University of Technology

Another alternative:

- **Percentage as a fraction [0;1] – [Thomas 2008, Ph.D. thesis]**
- **The more comments – the better?**



Evolution of the maintainability index in Linux

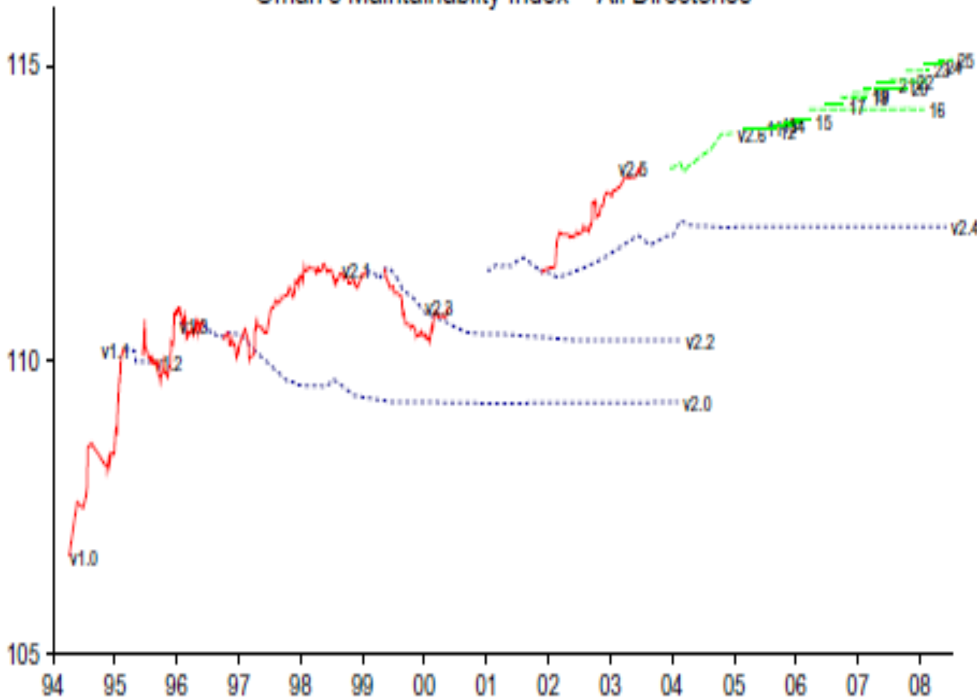


Fig. 9. Evolution of Oman's maintainability index.

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

A. Israeli, D.G. Feitelson 2010

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

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

Halstead **McCabe** **LOC**

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

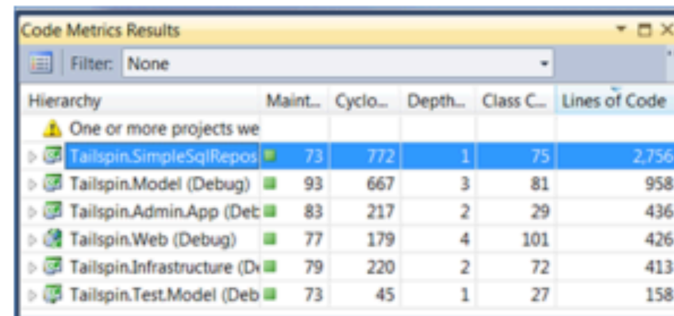
% comments

29 AUG 2014

Think Twice Before Using the "Maintainability Index"

posted in Research by Arie van Deursen

This is a quick note about the "Maintainability Index", a metric aimed at assessing software maintainability, as I recently run into developers and researchers who are (still) using it.



Code Metrics Results

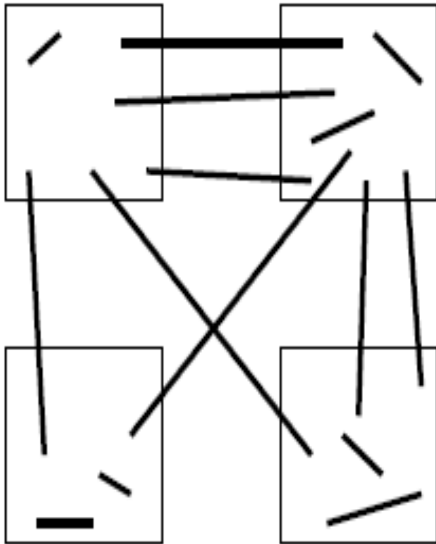
Filter: None

Hierarchy	Maint...	Cyclo...	Depth...	Class C...	Lines of Code
One or more projects we					
Tailspin.SimpleSqlRepos	73	772	1	75	2,756
Tailspin.Model (Debug)	93	667	3	81	958
Tailspin.AdminApp (Deb	83	217	2	29	436
Tailspin.Web (Debug)	77	179	4	101	426
Tailspin.Infrastructure (D	79	220	2	72	413
Tailspin.Test.Model (Deb	73	45	1	27	158

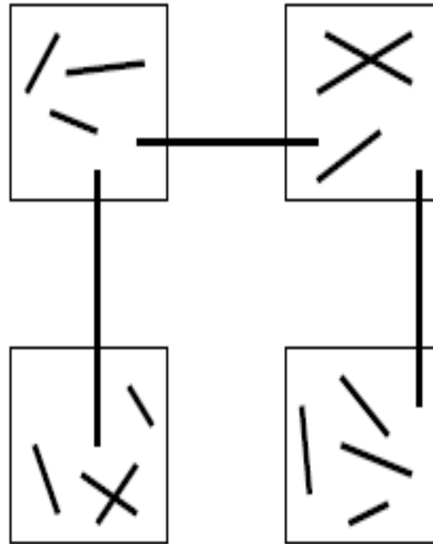
The Maintainability Index was introduced at the International Conference on Software Maintenance in 1992. To date, it is included in Visual Studio (since 2007), in the recent (2012) JSComplexity and Radon metrics reporters for Javascript and Python, and in older metric tool suites such as verifysoft.

What about modularity?

Design A



Design B



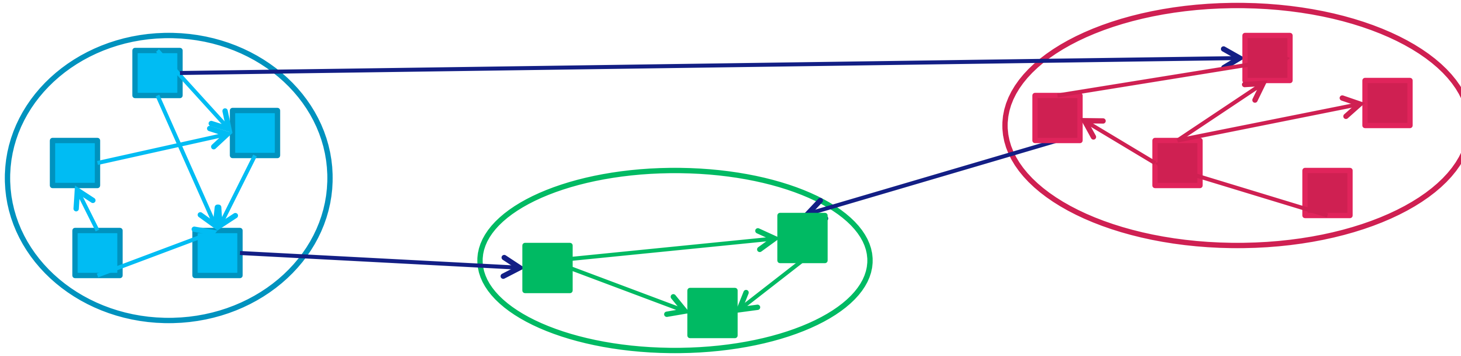
- **Cohesion: calls inside the module**
- **Coupling: calls between the modules**

	A	B
Cohesion	Lo	Hi
Coupling	Hi	Lo

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

TU/e Technische Universiteit Eindhoven University of Technology

Do you 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**

Fan-in and Fan-out Counter

# of components: 35

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

Component: file-package

Data filter: all / zero fan-in / zero fan-out / zero fan-in AND fan-out / NOT zero fan-in OR fan-out / zero fan-in AND NOT zero fan-out / NOT zero fan-in AND zero fan-out

Component name filter: any / begins with / contains / doesn't contain

Pattern:

Case sensitive

Save list to file...

Save metric file...

Save: fan-in / fan-out / fan-in and fan-out

OK

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

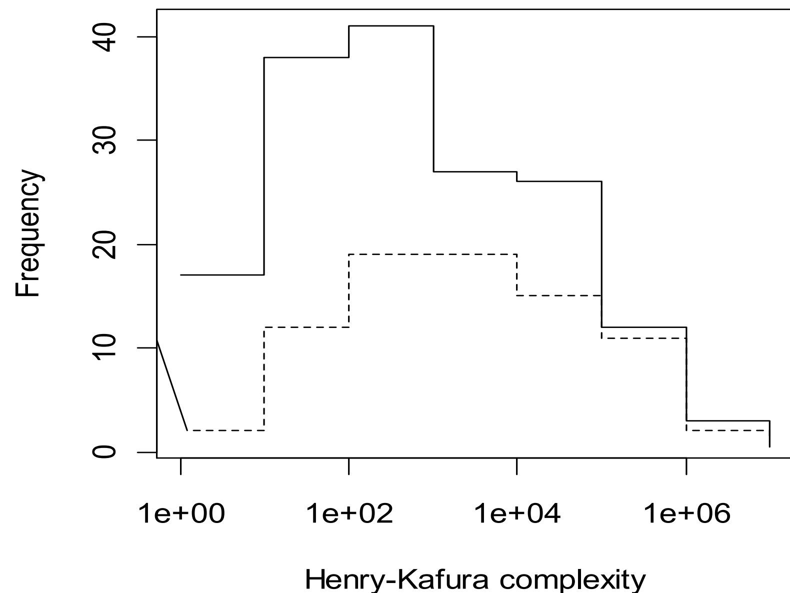
$$hk = sloc * (fanin * fanout)^2$$

- Shepperd

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

TU/e Technische Universiteit Eindhoven University of Technology

Information flow complexity of Unix procedures



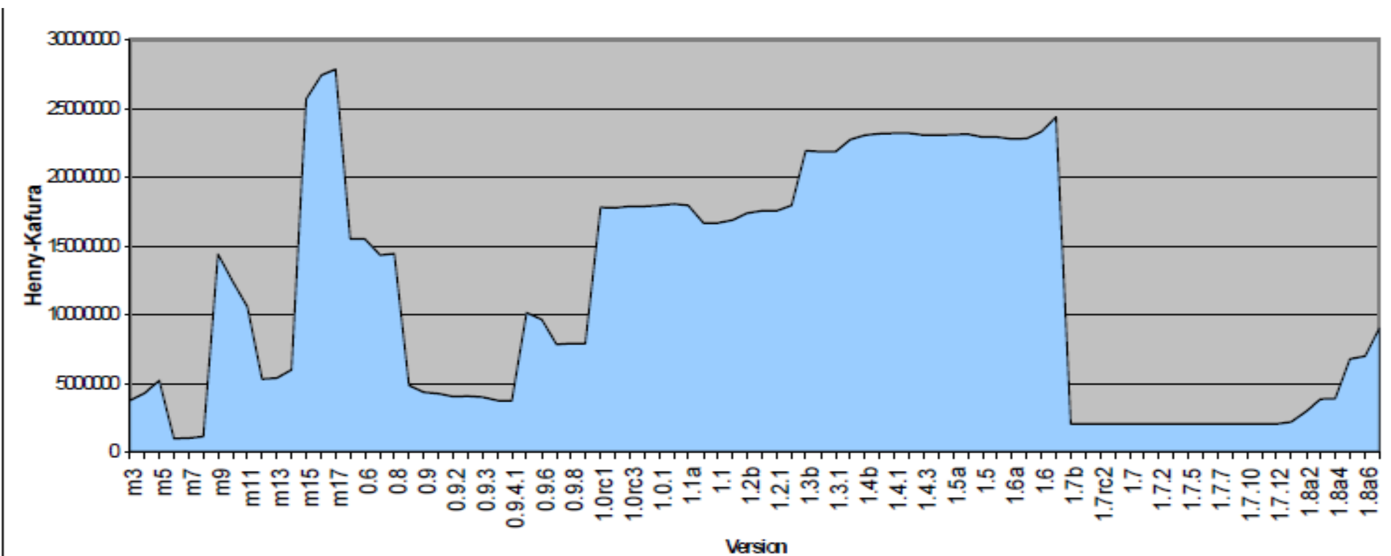
Module Name	Complexity	Procedure Complexity of Three Largest Procedures	Percent
buf	3541083	3468024	98
file	33062	29425	89
filesys	268807	254080	95
inode	13462921	12984995	96
kl11	3262	2120	65
lp11	855	829	97
mount	135503	135084	99
proc	436151	379693	87
text	24886	24831	99

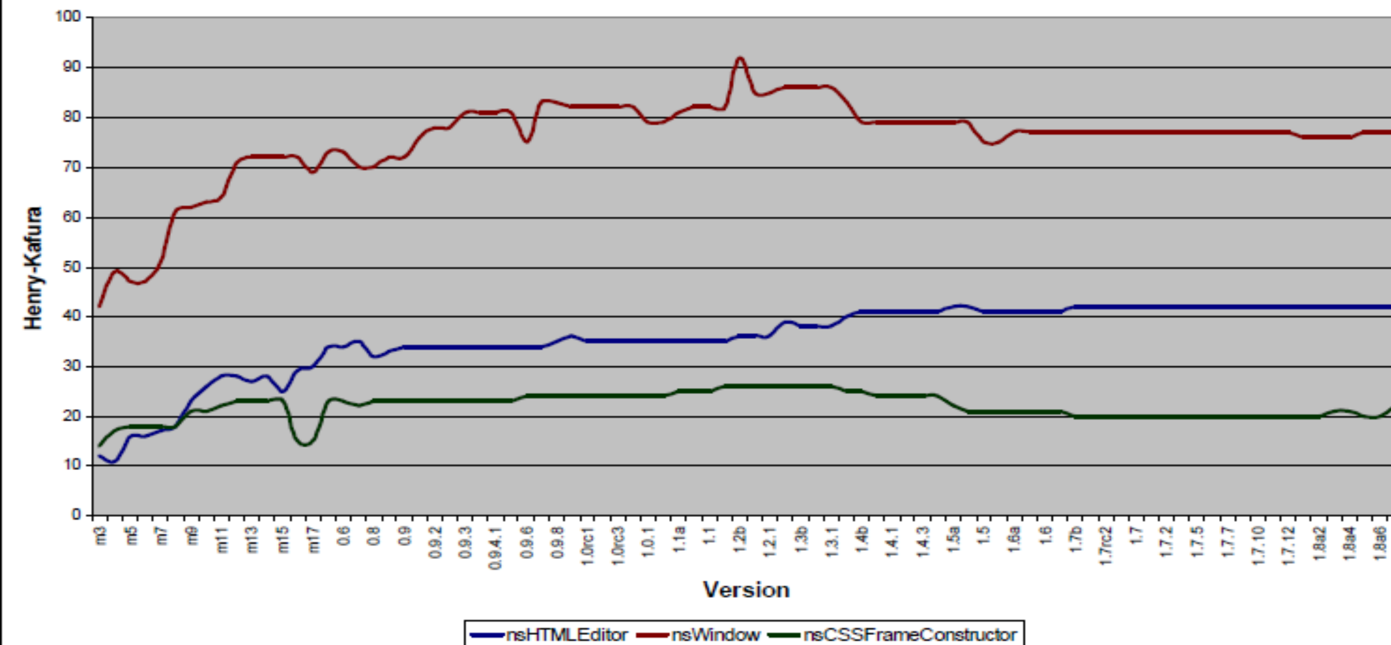
- **Solid – #procedures within the complexity range**
- **Dashed - #changed procedures within the complexity range**

- **Highly complex procedures are difficult to change but they are changed often!**
- **Complexity comes from the three largest procedures**

TU/e Technische Universiteit Eindhoven University of Technology

Evolution of the information flow complexity



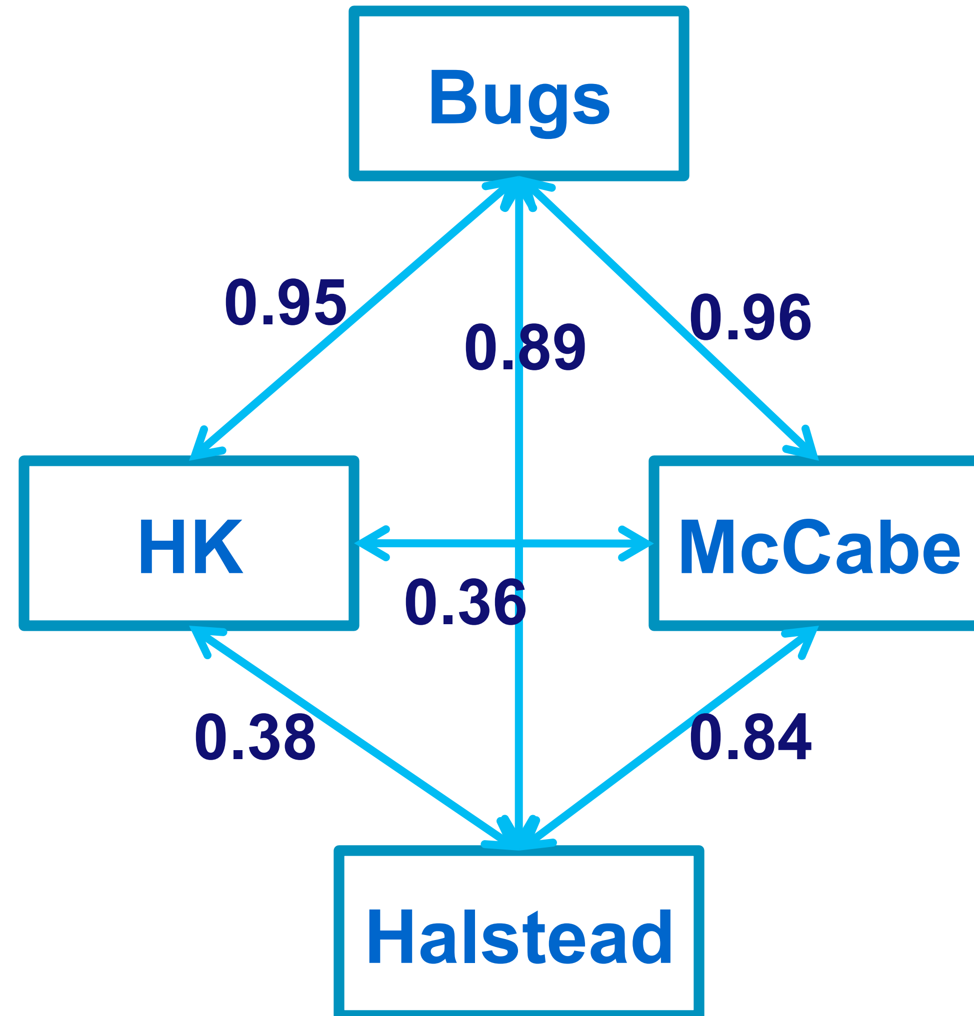


- **Mozilla**
- **Shepperd version**
- **Above: Σ the metrics over all modules**
- **Below: 3 largest modules**
- **What does this tell?**

Technische Universiteit Eindhoven University of Technology

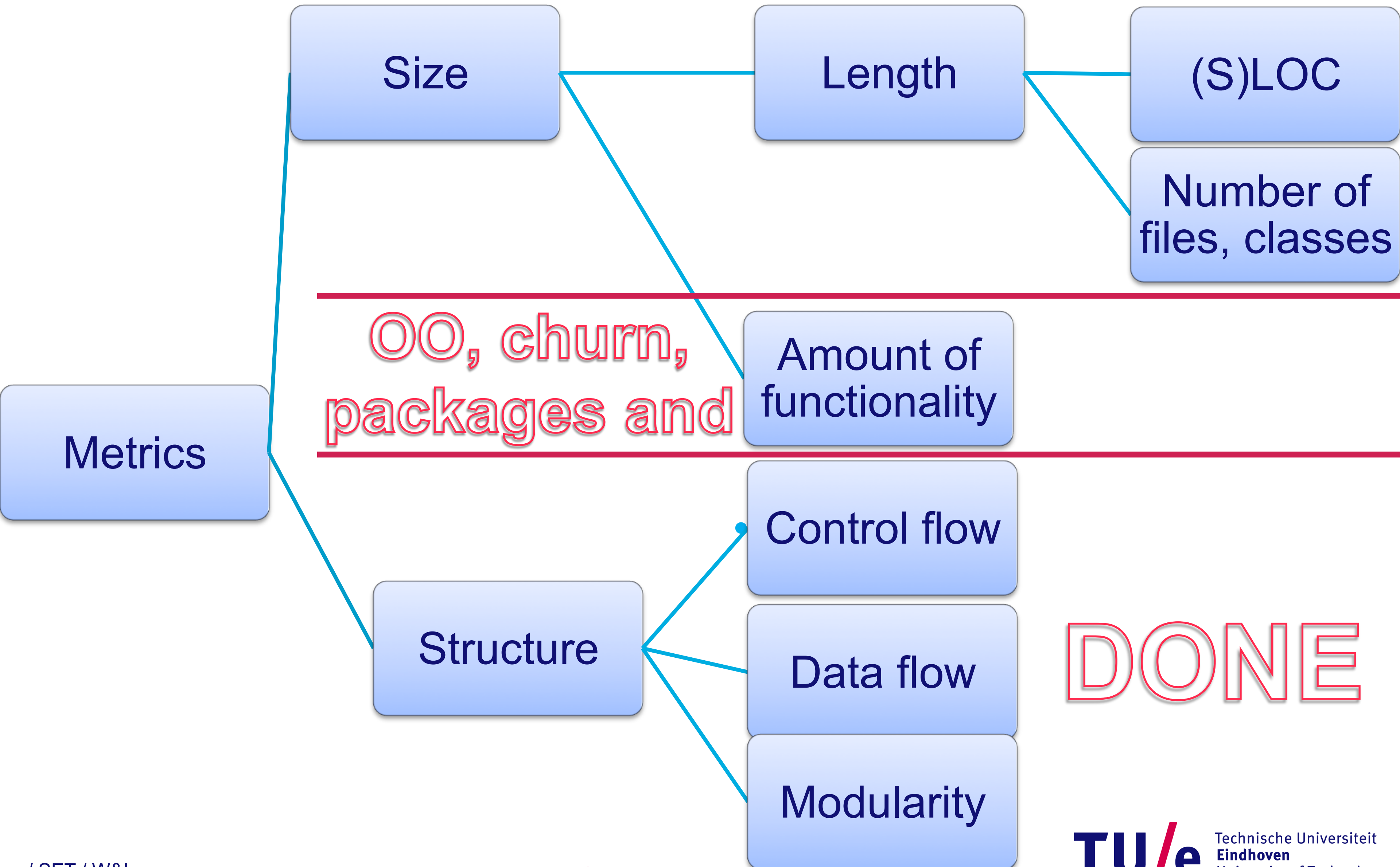
Summary so far…

- 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?



TU/e Technische Universiteit Eindhoven University of Technology

Where are we now?



TU/e Technische Universiteit Eindhoven University of Technology

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

TU/e Technische Universiteit Eindhoven University of Technology

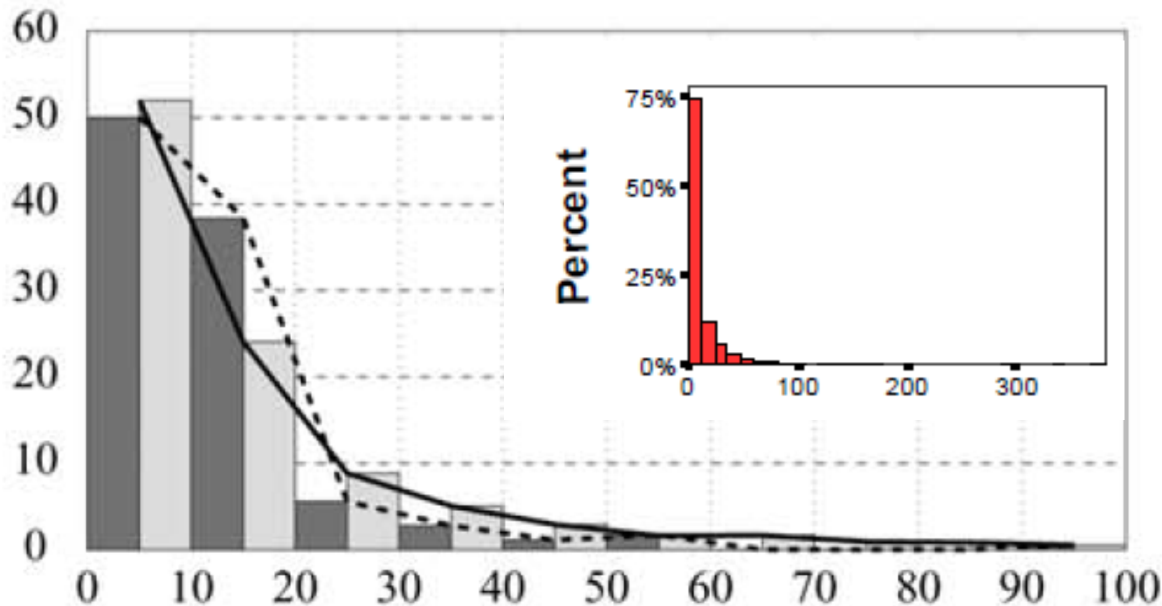
Chidamber and Kemerer

- **WMC – weighted methods per class**
  - **Sum of metrics(m) for all methods m in class C**
- **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.**

TU/e Technische Universiteit Eindhoven University of Technology

Chidamber and Kemerer

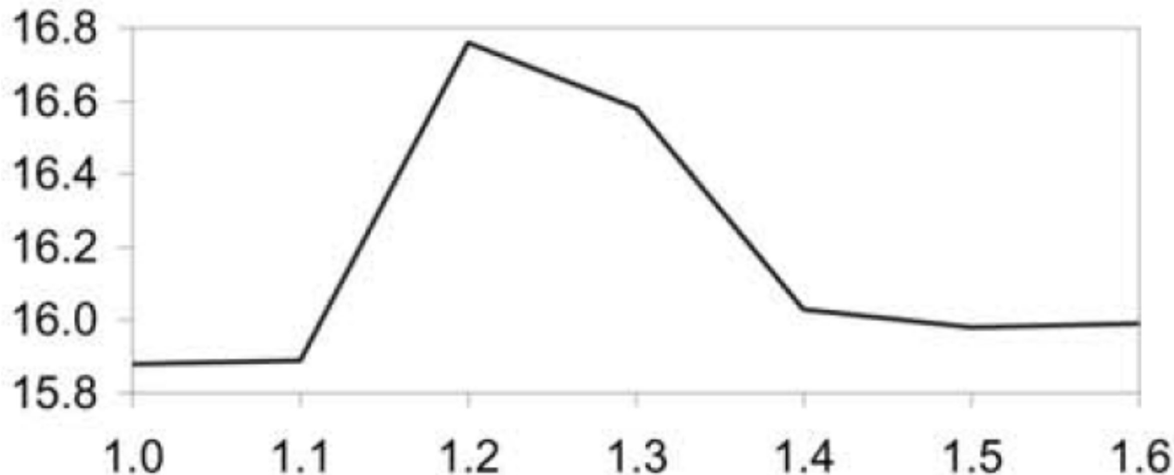
- **WMC – weighted methods per class**
  - **Sum of metrics(m) for all methods m in class C**
  - **Popular metrics: McCabe's complexity and unity**
  - **WMC/unity = number of methods**
  - **Statistically significant correlation with the number of defects**



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

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



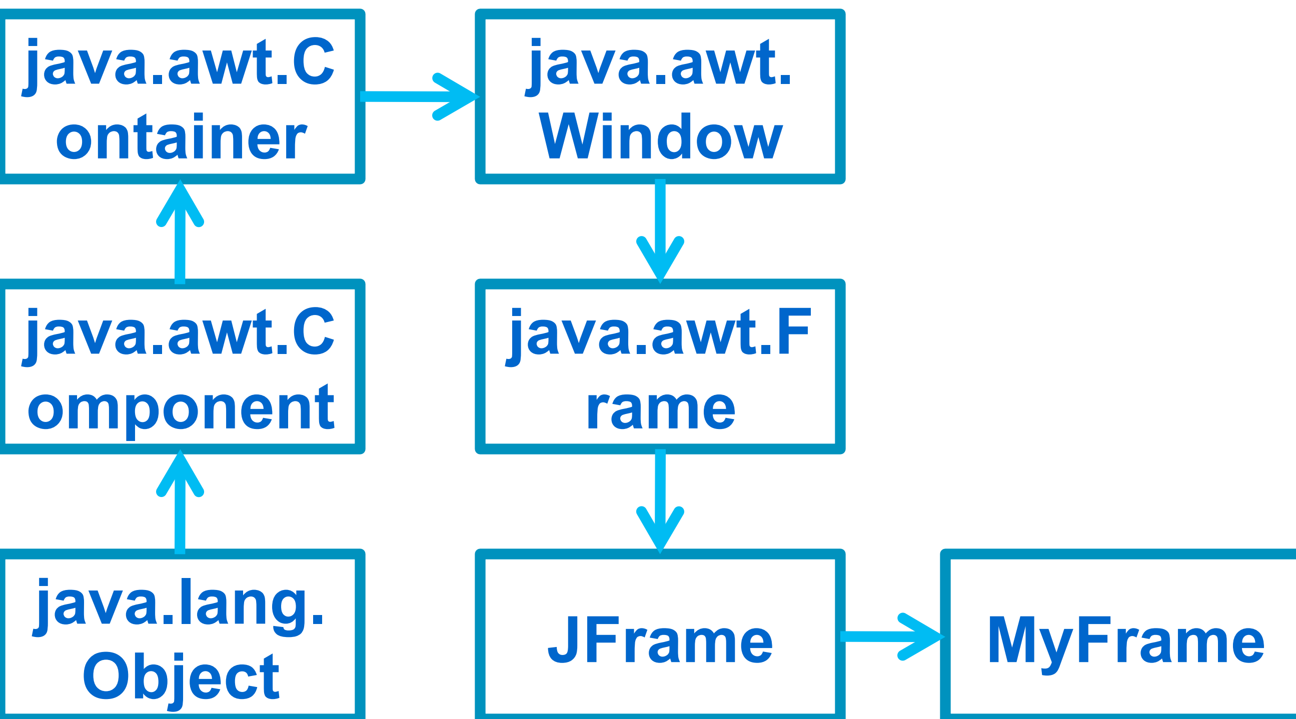
- WMC/unity
- Gyimothy et al.
- Average

- What do you think about this plot?

TU/e Technische Universiteit Eindhoven University of Technology

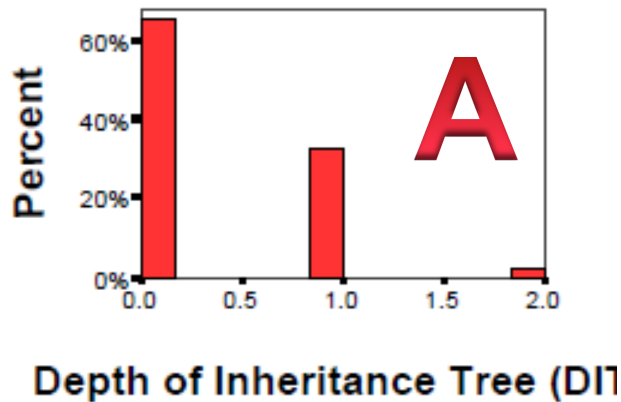
Depth of inheritance - DIT

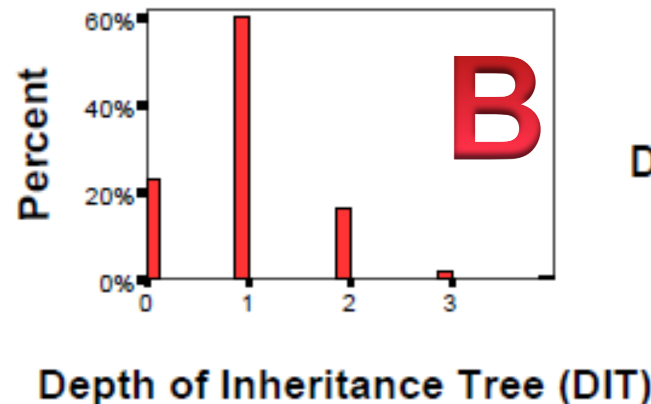
- **Variants: Were to start and what classes to include?**
  - **1, JFrame is a library class, excluded**
  - **2, JFrame is a library class, included**
  - **7**

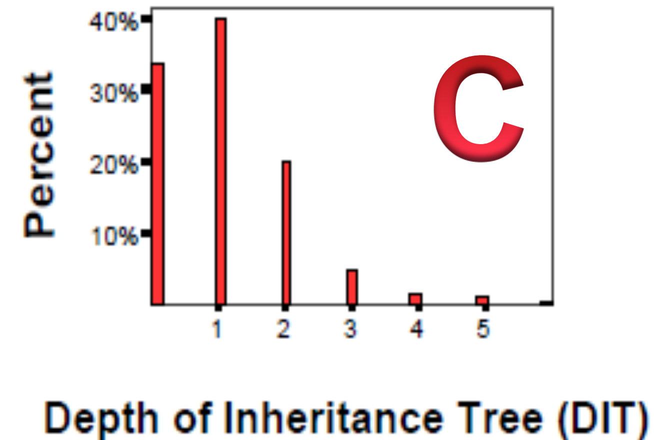


TU/e Technische Universiteit Eindhoven University of Technology

DIT – what is good and what is bad?



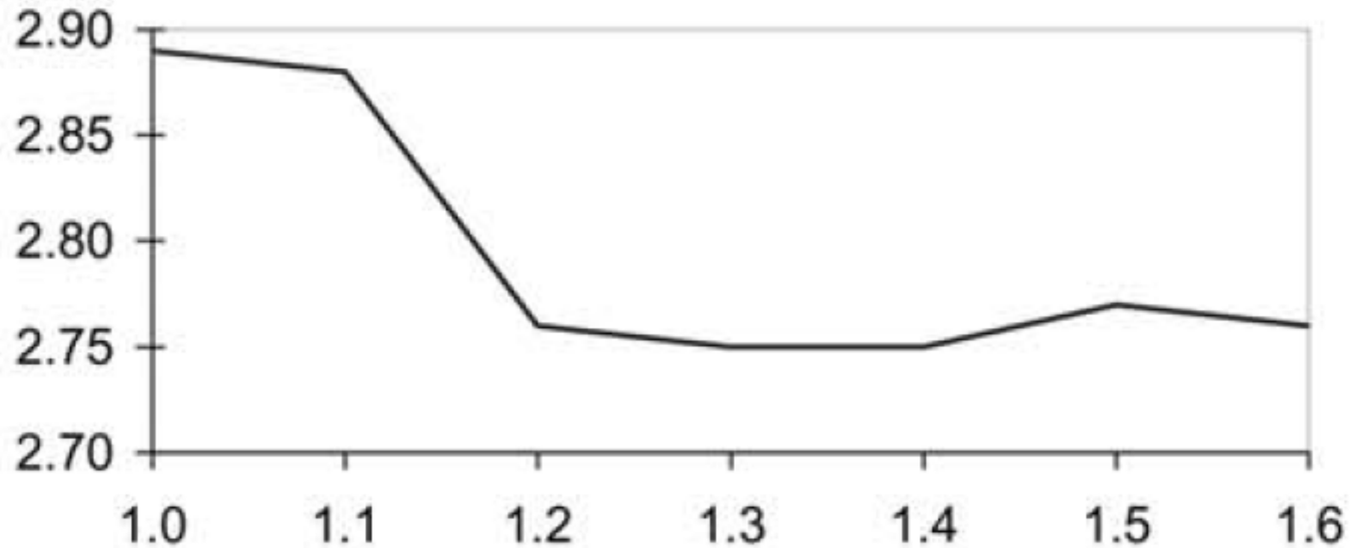




- Three NASA systems
- What can you say about the use of inheritance in systems A, B and C?
- Observation: quality assessment depends not just on one class but on the entire distribution

TU/e Technische Universiteit Eindhoven University of Technology

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

TU/e Technische Universiteit Eindhoven University of Technology

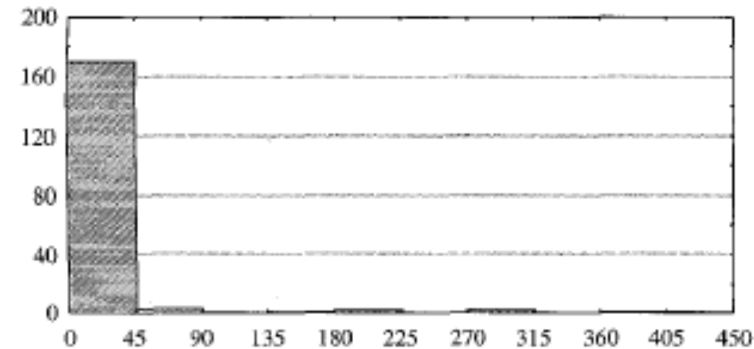
Modularity metrics: LCOM

- **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 instance variables
  - Q = #pairs of distinct methods in C that share instance variables
- Do you remember what is an **instance variable**?



[BBM] 180 classes

Discriminative ability is insufficient

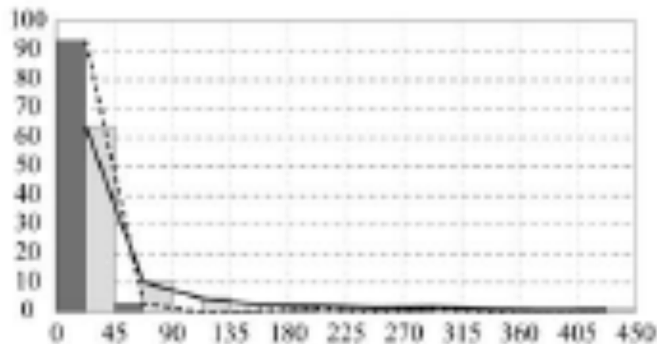
What about methods that use get/set instead of direct access?

TU/e Technische Universiteit Eindhoven University of Technology

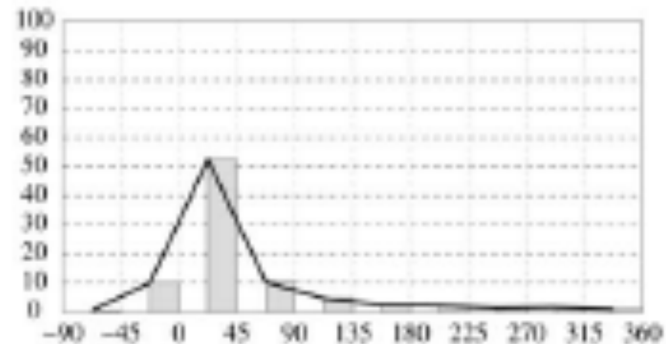
First solution: LCOMN

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

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



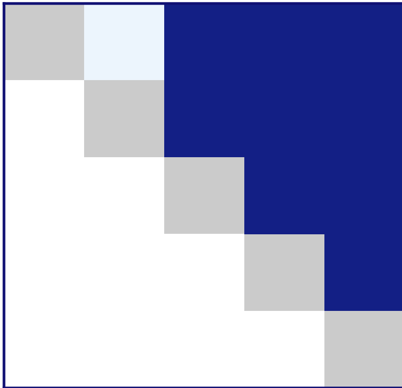
LCOM

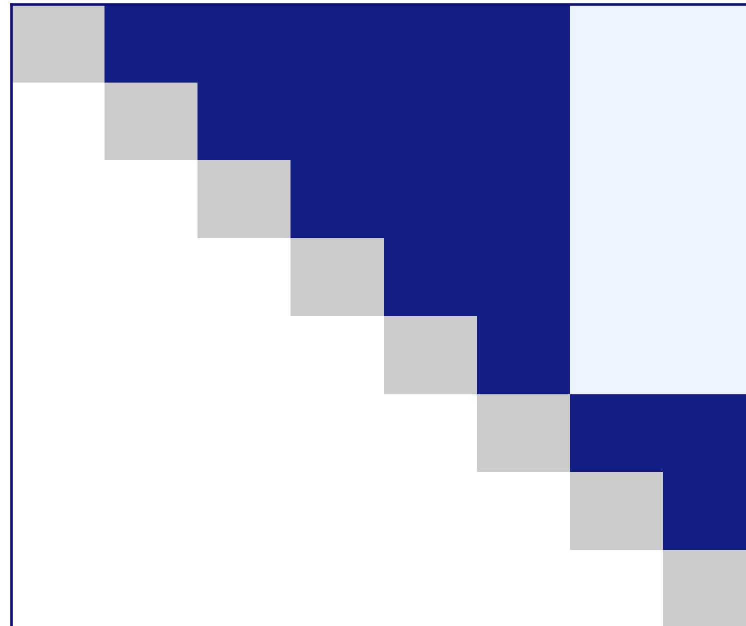


LCOMN

TU/e Technische Universiteit Eindhoven University of Technology

Still…





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

Next time

- **More about LCOM and other metrics**

TU/e Technische Universiteit Eindhoven University of Technology