2IS55 Software Evolution

Software metrics (2)

Alexander Serebrenik

Technische Universiteit Eindhoven University of Technology

Where innovation starts

Assignments

- Assignment 5:
  - Deadline: Today
  - “Cathedral” / “bazaar”
- Assignment 6:
  - Deadline: May 10
  - Published on Peach
  - 3-4 students

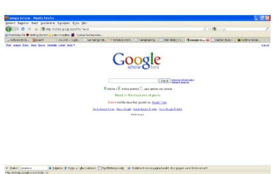
Assignment 6

- Determine which commits correspond to architecture improvement
  - How? Use modularity metrics
- Modify the log to include this information
- Integrate the log with the bug information from the bug tracker
  - FRASR
- Detect different developer roles: developers vs architects
  - ProM

Commercial break: Student assist. @ LaQuSo

- Extension of the MBE enviornment (Compat), 1x
- Modernization of a teaching tool AGORA (3TU), 1x
- Reduction voucher system (Daquilo;Sybren), 2x
- Cost and effort estimation (Ecosystems)
- Model checking PLC applications (Omron)
- Extension of a CRM system (two SMEs)
- You can also suggest a topic yourself!
- Contact Harold Weffers h.t.g.weffers@tue.nl

Sources



Recap: Software metrics

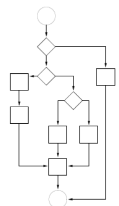
- Metrics and measurements
  - Scale: nominal, ordinal, interval, ratio, absolute
- Metrics:
  - Size
    - LOC/SLOC/LLOC
    - Amount of functionality
      - Function points
      - Size of the API
  - Complexity
    - Halstead’s volume, effort, time and #bugs

Structural complexity

- Structural complexity:
  - Control flow
  - Data flow

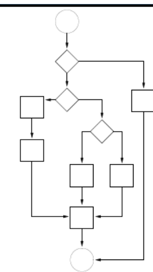
  Commonly represented as graphs → Graph-based metrics

  - 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) ≤ 15
- v(file) ≤ 100

McCabe complexity: Example

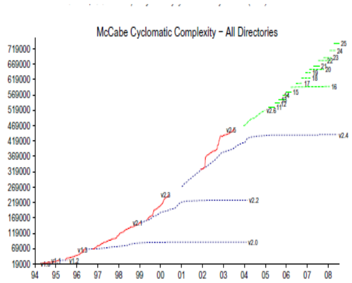
```
void sort ( int *a, int n ) {
int i, j, t;

  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?

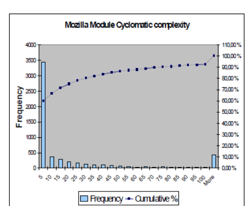
McCabe's complexity in Linux kernel

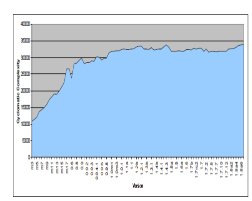


A. Israeli, D.G. Feitelson 2010

- 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

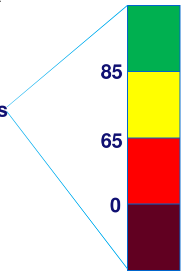
Summarizing: Maintainability index (MI)
[Coleman, Oman 1994]

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

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

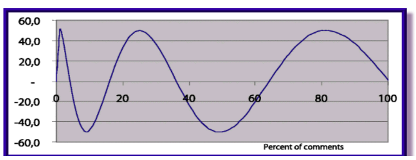
```
void sort ( int *a, int n ) {
int i, j, t;

  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;
              }
       }
  }
}
```

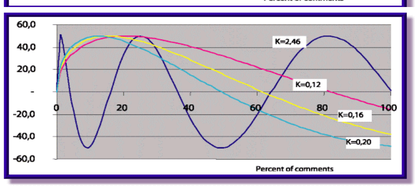
- Halstead's V ≈ 392
- McCabe's v(G) = 5
- LOC = 14
- $MI_1 \approx 96$
- Easy to maintain!

Comments?

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

[Liso 2001]



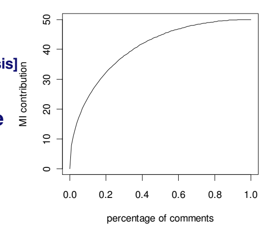


Peaks:
- 25% (OK),
- 1% and 81% - ???

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?



Evolution of the maintainability index in Linux

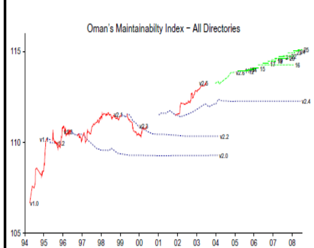


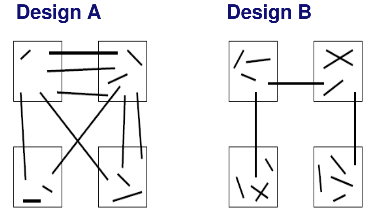
Fig. 9. Evolution of Oman's maintainability index.

- 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

What about modularity?

Design A Design B



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

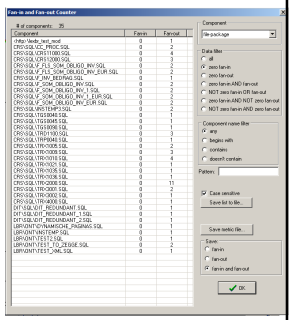
	A	B
Cohesion	Lo	Hi
Coupling	Hi	Lo

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

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



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

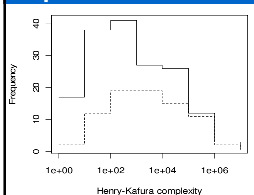
- Henry and Kafura: procedure as HW component connecting inputs to outputs

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

- Shepperd

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

Information flow complexity of Unix procedures

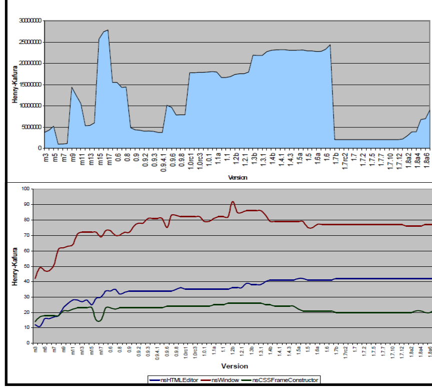


Module Name	Complexity	Procedure Complexity of Three Largest Procedures	Percent
buf	3541083	3468024	98
file	33062	29425	89
filesys	268807	254080	95
inode	13462921	12984995	96
kl11	3262	2120	65
lp11	855	829	97
mount	135503	135084	99
proc	436151	379693	87
text	24886	24831	99

- 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 the “most complex” procedures

Evolution of the information flow complexity



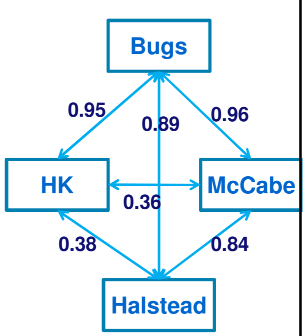
- Mozilla
- Shepperd version
- Above: Σ the metrics over all modules
- Below: 3 largest modules
- What does this tell?

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

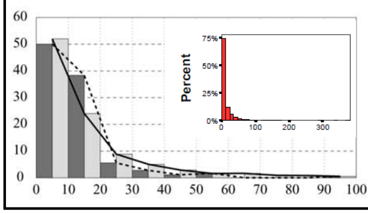
- All metrics so far were designed for imperative languages
  - Applicable for OO
    - On the method level
    - Also
      - Number of files → number of classes/packages
      - Fan-in → afferent coupling (C_a)
      - Fan-out → efferent coupling (C_e)
  - But do not reflect OO-specific complexity
    - 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

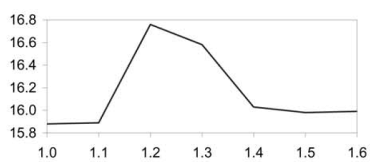
- WMC – weighted methods per class
  - Sum of metrics(m) for all methods m in class C
  - Popular metrics: McCabe's complexity and unity
  - WMC/unity = number of methods
  - Statistically significant correlation with the number of defects



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

Chidamber and Kemerer

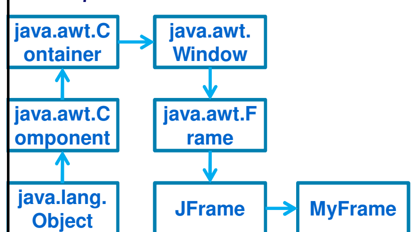
- WMC – weighted methods per class
  - Sum of metrics(m) for all methods m in class C
  - Popular metrics: McCabe's complexity and unity
  - WMC/unity = number of methods
  - Statistically significant correlation with the number of defects



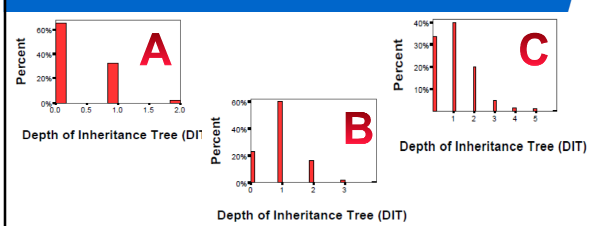
- WMC/unity
- Gyimothy et al.
- Average

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

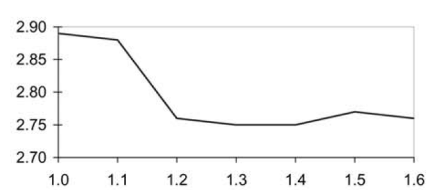


DIT – what is good and what is bad?



- 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 in DIT?

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	++	+	+	+

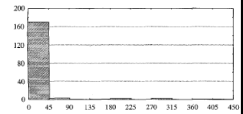
Significance of CK metrics to predict the number of faults

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

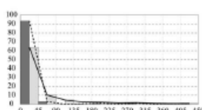
Discriminative ability is insufficient

What about get/set?

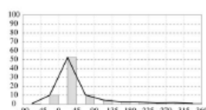
First solution: LCOMN

- Defined similarly to LCOM but allows negative values

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

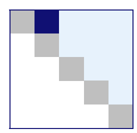


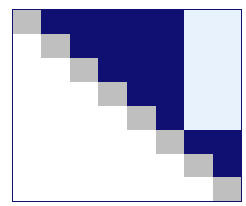
LCOM



LCOMN

Still…





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

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

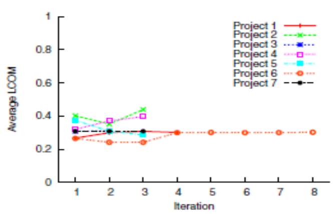
$$m(V_i) = 0$$

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

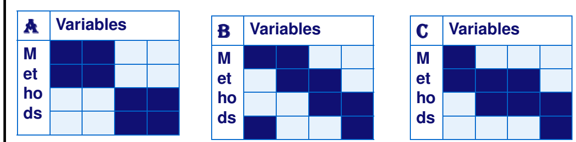
Evolution of LCOM [Henderson-Sellers et al.]



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]

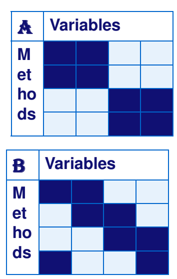


- Due to [Fernández, Peña 2006]
- Method-variable diagrams: dark spot = access
- LCOM(A) = LCOM(B) = LCOM(C) = 0.67
- A seems to be less cohesive than B and C!

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?



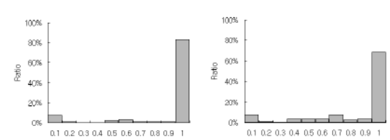
Experimental evaluation of LCOM variants

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)

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
• Aggregation techniques