Assignment 6

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

Assignment 8:

• Assignment 8:
  • Open: June 1
  • Deadline: June 22
  • 1-2 students
  • ReqVis
    • Try it!
    • Give us feedback before June 1!
    • Mac-fans: Talk to Wiljan!
So far

- **Metrics**
  - Size
  - Length
  - (S)LOC

  - Number of files, classes

- **Structure**
  - Amount of functionality
  - Control flow
  - Data flow
  - Modularity

**Next Week**

**Today**
Complexity metrics: Halstead (1977)

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

Parts of a statement: Line: LOC, Units, files, Packages, SLOC, LLOC classes directories
operators and operands

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

• Four basic metrics of Halstead

<table>
<thead>
<tr>
<th></th>
<th>Total</th>
<th>Unique</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operators</td>
<td>N1</td>
<td>n1</td>
</tr>
<tr>
<td>Operands</td>
<td>N2</td>
<td>n2</td>
</tr>
</tbody>
</table>

• Length: \( N = N1 + N2 \)
• Vocabulary: \( n = n1 + n2 \)
• Volume: \( V = N \log_2 n \)
  • Insensitive to lay-out
  • VerifySoft:
    - \( 20 \leq \text{Volume(function)} \leq 1000 \)
    - \( 100 \leq \text{Volume(file)} \leq 8000 \)
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

\[
V = 80 \log_2(24) \approx 392
\]

Inside the boundaries [20;1000]

<table>
<thead>
<tr>
<th></th>
<th>Total</th>
<th>Unique</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operators</td>
<td>$N1 = 50$</td>
<td>$n1 = 17$</td>
</tr>
<tr>
<td>Operands</td>
<td>$N2 = 30$</td>
<td>$n2 = 7$</td>
</tr>
</tbody>
</table>
Further Halstead metrics

- **Volume**: \( V = N \log_2 n \)
- **Difficulty**: \( D = \left( \frac{n_1}{2} \right) \times \left( \frac{N_2}{n_2} \right) \)
  - Sources of difficulty: new operators and repeated operands
  - Example: \( 17/2 \times 30/7 \approx 36 \)
- **Effort**: \( E = V \times D \)
- **Time to understand/implement (sec)**: \( T = E/18 \)
  - Running example: 793 sec \( \approx 13 \text{ min} \)
  - Does this correspond to your experience?
- **Bugs delivered**: \( E^{2/3}/3000 \)
  - For C/C++: known to underapproximate
  - Running example: 0.19
Halstead metrics are sensitive to...

• What would be your answer?

• Syntactic sugar:

<table>
<thead>
<tr>
<th></th>
<th>Total</th>
<th>Unique</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operators</td>
<td>N1 = 2</td>
<td>n1 = 2</td>
</tr>
<tr>
<td>Operands</td>
<td>N2 = 3</td>
<td>n2 = 2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Total</th>
<th>Unique</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operators</td>
<td>N1 = 1</td>
<td>n1 = 1</td>
</tr>
<tr>
<td>Operands</td>
<td>N2 = 1</td>
<td>n2 = 1</td>
</tr>
</tbody>
</table>

• Solution: normalization (see the code duplication slides)
Structural complexity

- Structural complexity:
  - Control flow
  - Data flow
  - Modularity

Commonly represented as graphs

Graph-based metrics

- Number of vertices
- Number of edges
- Maximal length (depth)
McCabe complexity (1976)

In general
• $v(G) = \#\text{edges} - \#\text{vertices} + 2$

For control flow graphs
• $v(G) = \#\text{binaryDecisions} + 1$, or
• $v(G) = \#\text{IFs} + \#\text{LOOPs} + 1$

Number of paths in the control flow graph.
A.k.a. “cyclomatic complexity”

Each path should be tested!
$v(G)$ – a testability metrics

Boundaries
• $v(\text{function}) \leq 15$
• $v(\text{file}) \leq 100$
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;
            }
        }
    }
}
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

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

A. Israeli, D.G. Feitelson 2010
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]

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

\[ MI_2 = MI_1 + 50 \sin \sqrt{2.46 \text{perCM}} \]

- 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!
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 $V \approx 392$
• McCabe’s $v(G) = 5$
• LOC = 14
• $MI_1 \approx 96$
• Easy to maintain!
$50 \sin \sqrt{2.46 \text{ per CM}}$

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

Better:
- $0.12 \leq K \leq 0.2$

[1][Liso 2001]
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

- 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?

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

**Design A**

**Design B**

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

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cohesion</td>
<td>Lo</td>
<td>Hi</td>
</tr>
<tr>
<td>Coupling</td>
<td>Hi</td>
<td>Lo</td>
</tr>
</tbody>
</table>
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

\[ E_{i,j} = \frac{\varepsilon_{i,j}}{2N_iN_j} \]

• 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
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

\[
hk = sloc \times (\text{fanin} \times \text{fanout})^2
\]

- Shepperd

\[
s = (\text{fanin} \times \text{fanout})^2
\]
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
Evolution of the information flow complexity

- Mozilla
- Shepperd version
- Above: $\Sigma$ 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?

<table>
<thead>
<tr>
<th>HK</th>
<th>McCabe</th>
<th>Bugs</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.95</td>
<td>0.89</td>
<td>0.96</td>
</tr>
<tr>
<td>0.36</td>
<td>0.84</td>
<td></td>
</tr>
<tr>
<td>0.38</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
From imperative to OO

• All metrics so far were designed for imperative languages
  • Applicable for OO
    − On the method level
    − Also
      − Number of files $\rightarrow$ number of classes/packages
      − Fan-in $\rightarrow$ afferent coupling ($C_a$)
      − Fan-out $\rightarrow$ 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
  - \( \text{CBO}(A) = | \{ B | A \text{ is coupled to } B \} | \)
- **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
  - \( \text{WMC}/\text{unity} = \text{number of methods} \)
  - 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
  - \( \text{WMC/unity} = \text{number of methods} \)
  - Statistically significant correlation with the number of defects

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

Significance of CK metrics to predict the number of faults

<table>
<thead>
<tr>
<th>Metric</th>
<th>Our results</th>
<th>[1]</th>
<th>[22]</th>
<th>[21]</th>
</tr>
</thead>
<tbody>
<tr>
<td>WMC</td>
<td>++</td>
<td>+</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>DIT</td>
<td>+</td>
<td>++</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>RFC</td>
<td>++</td>
<td>++</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>NOC</td>
<td>0</td>
<td>++</td>
<td>--</td>
<td></td>
</tr>
<tr>
<td>CBO</td>
<td>++</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
</tbody>
</table>
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

\[ LCOMN(C) = P - Q \]
Still...

- Method * method tables
  - Light blue: Q, dark blue: P
- Calculate the LCOMs
- Does this correspond to your intuition?
• $m$ – number of methods
• $v$ – number of variables (attrs)
• $m(V_i)$ - #methods that access $V_i$

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

• Cohesion is maximal: all methods access all variables
  $$m(V_i) = m \text{ and } LCOM = 0$$

• No cohesion: every method accesses a unique variable
  $$m(V_i) = 1 \text{ and } LCOM = 1$$

• Can LCOM exceed 1?
• If some variables are not accessed at all, then

\[ m(V_i) = 0 \]

and

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

Hence

\[ \text{LCOM is undefined for } m = 1 \]

\[ \text{LCOM} \leq 2 \]
Evolution of LCOM [Henderson-Sellers et al.]

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

Sato, Goldman, Kon 2007
Shortcomings of LCOM [Henderson-Sellers]

- Due to [Fernández, Peña 2006]
- Method-variable diagrams: dark spot = access
- $\text{LCOM}(\ ) = \text{LCOM}(\ ) = \text{LCOM}(\ ) = 0.67$
  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
## Experimental evaluation of LCOM variants

<table>
<thead>
<tr>
<th></th>
<th>Correlation with expert assessment</th>
<th>Group 1</th>
<th>Group 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cox, Etzkorn and Hughes 2006</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chidamber Kemerer</td>
<td>-0.43 (p = 0.12)</td>
<td>-0.57 (p = 0.08)</td>
<td></td>
</tr>
<tr>
<td>Henderson-Sellers</td>
<td>-0.44 (p = 0.12)</td>
<td>-0.46 (p = 0.18)</td>
<td></td>
</tr>
<tr>
<td>Hitz, Montazeri</td>
<td>-0.47 (p = 0.06)</td>
<td>-0.53 (p = 0.08)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Correlation with expert assessment</th>
<th>Group 1</th>
<th>Group 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Etzkorn, Gholston, Fortune, Stein, Utley, Farrington, Cox</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chidamber Kemerer</td>
<td>-0.46 (rating 5/8)</td>
<td>-0.73 (rating 1.5/8)</td>
<td></td>
</tr>
<tr>
<td>Henderson-Sellers</td>
<td>-0.44 (rating 7/8)</td>
<td>-0.45 (rating 7/8)</td>
<td></td>
</tr>
<tr>
<td>Hitz, Montazeri</td>
<td>-0.51 (rating 2/8)</td>
<td>-0.54 (rating 5/8)</td>
<td></td>
</tr>
</tbody>
</table>
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 \( TCC = \frac{NDP}{NP} \)
- Loose class cohesion \( LCC = \frac{NIP}{NP} \)
- NB: Constructors and destructors are excluded
## Experimental evaluation of LCC/TCC

<table>
<thead>
<tr>
<th>Method</th>
<th>Correlation with expert assessment</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Etzkorn, Gholston, Fortune, Stein, Utley, Farrington, Cox</td>
<td>Group 1</td>
<td>Group 2</td>
</tr>
<tr>
<td>Chidamber Kemerer</td>
<td>-0.46 (rating 5/8)</td>
<td>-0.73 (rating 1.5/8)</td>
</tr>
<tr>
<td>Henderson-Sellers</td>
<td>-0.44 (rating 7/8)</td>
<td>-0.45 (rating 7/8)</td>
</tr>
<tr>
<td>Hitz, Montazeri</td>
<td>-0.51 (rating 2/8)</td>
<td>-0.54 (rating 5/8)</td>
</tr>
<tr>
<td>TCC</td>
<td>-0.22 (rating 8/8)</td>
<td>-0.057 (rating 8/8)</td>
</tr>
<tr>
<td>LCC</td>
<td>-0.54 (rating 1/8)</td>
<td>-0.73 (rating 1.5/8)</td>
</tr>
</tbody>
</table>
Conclusions: Metrics so far…

<table>
<thead>
<tr>
<th>Level</th>
<th>Metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method</td>
<td>LOC, McCabe, Henry Kafura</td>
</tr>
<tr>
<td>Class</td>
<td>WMC, NOC, DIT, LCOM (and variants), LCC/TCC</td>
</tr>
<tr>
<td>Packages</td>
<td>???</td>
</tr>
</tbody>
</table>

Next time:
- Package-level metrics (Martin)
- Metrics of change