2IS55 Software Evolution

Software metrics (2)

Alexander Serebrenik

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 environment (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
  - 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

McCabe complexity: Example

```c
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)

McCabe’s complexity in Mozilla [Resdal 2005]

- Most of the modules have low cyclomatic complexity
- Complexity of the system seems to stabilize
Summarizing: Maintainability index (MI)  
Coleman, Oman 1994

\[ M_{I_1} = 171 - 5.21 \ln(V) - 0.23V(g) - 16.2 \ln(LOC) \]

\[ M_{I_2} = M_{I_1} + 50 \sin \left( \sqrt{2.46 \text{perCM}} \right) \]

- MI 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

```c
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 \( v(G) = 5 \)
- \( \text{LOC} = 14 \)
- \( \text{MI} = 96 \)
- Easy to maintain!

Comments?

\[ 50 \sin \left( \sqrt{2.46 \text{perCM}} \right) \]

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

Better:
- \( 0.12 \leq K \leq 0.2 \)

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

Another alternative:

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

What about modularity?

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

- 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
  \[ hk = \text{sloc} \times (\text{fanin} \times \text{fanout})^2 \]
- 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: Σ the metrics over all modules
- Below: 3 largest modules
- What does this tell?

Summary so far...

- Complexity metrics
  - Halstead’s effort
  - McCabe (cycloomatic)
  - 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 \( \rightarrow \) 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
  - 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

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

<table>
<thead>
<tr>
<th>Metric</th>
<th>Our results</th>
<th>[2]</th>
<th>[22]</th>
<th>[23]</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOC</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>CBO</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>RFC</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>

Significance of CK metrics to predict the number of faults

First solution: LCOMN

- Defined similarly to LCOM but allows negative values

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

<table>
<thead>
<tr>
<th>LCOM</th>
<th>LCOMN</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>P</td>
</tr>
<tr>
<td>Q</td>
<td>Q</td>
</tr>
</tbody>
</table>

Modularity metrics: LCOM

- LCOM – lack of cohesion of methods

Chidamber Kemerer:

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

Discriminative ability is insufficient

where

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

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 \( V_i \)
- 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?

Hence

- LCOM is undefined for \( m = 1 \)
- LCOM \( \leq 2 \)

If some variables are not accessed at all, then

\[ m(V_i) = 0 \]

and

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

\[ \frac{1}{1-m} \]

\[ 1 - m = 1 + \frac{1}{m-1} \]
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

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

Experimental evaluation of LCOM variants

<table>
<thead>
<tr>
<th>Metric</th>
<th>Group 1 Correlation</th>
<th>Group 2 Correlation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chidamber Kemerer</td>
<td>-0.43 (p = 0.12)</td>
<td>-0.57 (p = 0.08)</td>
</tr>
<tr>
<td>Henderson-Sellers</td>
<td>-0.44 (p = 0.12)</td>
<td>-0.46 (p = 0.18)</td>
</tr>
<tr>
<td>Hitz, Montazeri</td>
<td>-0.47 (p = 0.06)</td>
<td>-0.53 (p = 0.08)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Metric</th>
<th>Group 1 Rating</th>
<th>Group 2 Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chidamber Kemerer</td>
<td>5/8</td>
<td>1.5/8</td>
</tr>
<tr>
<td>Henderson-Sellers</td>
<td>7/8</td>
<td>7/8</td>
</tr>
<tr>
<td>Hitz, Montazeri</td>
<td>2/8</td>
<td>5/8</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 = NDP/NP
- Loose class cohesion LCC = NIP/NP
- NB: Constructors and destructors are excluded
**Experimental evaluation of LCC/TCC**

<table>
<thead>
<tr>
<th>Methods</th>
<th>Correlation with expert assessment</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Group 1</td>
</tr>
<tr>
<td>Chidamber Kemerer</td>
<td>-0.46 (rating 5/8)</td>
</tr>
<tr>
<td>Henderson-Sellers</td>
<td>-0.44 (rating 7/8)</td>
</tr>
<tr>
<td>Hitz, Montazeri</td>
<td>-0.51 (rating 2/8)</td>
</tr>
<tr>
<td>TCC</td>
<td>-0.22 (rating 8/8)</td>
</tr>
<tr>
<td>LCC</td>
<td>-0.54 (rating 1/8)</td>
</tr>
</tbody>
</table>

**Metrics so far...**

<table>
<thead>
<tr>
<th>Level</th>
<th>Metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Method LOC, McCabe, Henry Kafura</td>
</tr>
<tr>
<td></td>
<td>Class 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
- Aggregation techniques