Software metrics: Complexity

Alexander Serebrenik
Assignments

- Assignment 4: March 31, pairs
- Assignment 5: Apr 7, individual
  - Do software evolution laws hold in practice?
  - Use software metrics to instrument and verify the claim
  - Preprocessed datasets

COMETS - Code metrics time series dataset

COMETS (Code Metrics Time Series) is a dataset of source code metrics collected from several systems to support empirical studies on source code evolution. The dataset includes information on the evolution of the following Java-based systems:

- Eclipse JDT Core: compiler and type system
- Eclipse PDE UI: components
- Equinox: OSGi implementation
- Lucene: text search engine library
- Hibernate: persistence framework
- Spring: application development framework
- JabRef: bibliography reference manager
- PMD: a source code analyzer
- TV-Browser: electronic TV guide
- Pentaho Console: console for Pentaho Data Integration (PDI)

Download Helix - The Software Evolution Data Set

The following table contains the download links for each of the systems available as part of the Helix Data Set and includes:

- The **releases**: The JARs containing the class files for each release of the systems along with meta data.
- The **metrics**: A metric history derived from extraction of the releases. You can use this data...
Sources
So far

Metrics

- Size
- Length
- (S)LOC
- Number of files, classes

NEXT WEEK

Structure

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

TODAY

/ SET / W&I

18/03/15 PAGE 4
Complexity metrics: Halstead (1977)

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

Parts of a statement: operators and operands

Line: LOC, SLOC, LLOC
Units, files, classes
Packages, directories

- 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: \( N = N_1 + N_2 \)
- Vocabulary: \( n = n_1 + n_2 \)
- 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

```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;
            }
        }
    }
}
```

- Ignore the function definition
- Count operators and operands

<table>
<thead>
<tr>
<th>Operators</th>
<th>Total</th>
<th>Unique</th>
</tr>
</thead>
<tbody>
<tr>
<td>N1 = 50</td>
<td>n1 = 17</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operands</th>
<th>Total</th>
<th>Unique</th>
</tr>
</thead>
<tbody>
<tr>
<td>N2 = 30</td>
<td>n2 = 7</td>
<td></td>
</tr>
</tbody>
</table>

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

Inside the boundaries [20;1000]
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: \( \frac{17}{2} \times \frac{30}{7} \approx 36 \)
- **Effort**: \( E = V \times D \)
- **Time to understand/implement (sec)**: \( T = \frac{E}{18} \)
  - Running example: 793 sec \( \approx \) 13 min
- **Bugs delivered**: \( E^{2/3}/3000 \)
  - For C/C++: known to underapproximate
  - Running example: 0.19

<table>
<thead>
<tr>
<th></th>
<th>Total</th>
<th>Unique</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Operators</strong></td>
<td>( N_1 )</td>
<td>( n_1 )</td>
</tr>
<tr>
<td><strong>Operands</strong></td>
<td>( N_2 )</td>
<td>( n_2 )</td>
</tr>
</tbody>
</table>
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’s 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’s 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;
            }
        }
    }
}
```

- 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

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

A. Israeli, D.G. Feitelson 2010
Most of the modules have low cyclomatic complexity.
Complexity of the system seems to stabilize.
CC vs SLOC

- Strong correlation is often claimed
- Problematic [Landman, S, Vinju 2014]
  - 17M Java methods: $R^2 \approx 0.43$
  - Even less for larger methods
  - Huge variance
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 middle-sized 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 $v(G) = 5$
- LOC = 14
- $MI_1 \approx 96$
- Easy to maintain!
Comments?

\[ 50 \sin \sqrt{2.46} \text{perCM} \]

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

Better:
- \[ 0.12 \leq K \leq 0.2 \]

[Liso 2001]
Another alternative:


- 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?

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

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

- 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 from the three largest 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?
Where are we now?

Metrics

- Size
- Length
- (S)LOC

Structure

- Control flow
- Data flow
- Modularity

Amount of functionality

OO, churn, packages and

DONE
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 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
  - \( \text{WMC/unity} = \) number of methods
  - Statistically significant correlation with the number of defects

- **WMC/unity**
- **Gyimothy et al.**
- **Average**
• Variants: Where 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>
Software metrics: More metrics

Alexander Serebrenik
Assignments

- Assignment 4: March 31, pairs
- Assignment 5: Apr 7, individual
  - Do software evolution laws hold in practice?
  - Use software metrics to instrument and verify the claim
- Preprocessed datasets
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.
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

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

[BBM] 180 classes

Discriminative ability is insufficient

What about methods that use get/set instead of direct access?
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?
Henderson-Sellers, Constantine and Graham 1996

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

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

- 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 if no variables are accessed

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

Hence

LCOM is undefined for \( m = 1 \)

\[ \text{LCOM} \leq 2 \]
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) ?

\[
\left(\frac{1}{v} \sum_{i=1}^{v} m(V_i)\right) - m \quad \frac{1}{1 - m}
\]
Shortcomings of LCOM [Henderson-Sellers]

Due to [Fernández, Peña 2006]

- All LCOM values are the same: 0.67
  - \( m=4, m(V_i) = 2 \) for all \( i \)

- 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
    – except for getters/setters
  • Edge between a and b, if
    – a and b access the same variable

• LCOM values
  • 0, no methods
  • 1, cohesive component
  • 2 or more, lack of cohesion
Alternative [Hitz, Montazeri 1995]

- LCOM as the number of strongly connected components in the following graph
  - Vertices: methods
    - except for getters/setters
  - Edge between \( a \) and \( b \), if
    - \( a \) and \( b \) access the same variable

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

<table>
<thead>
<tr>
<th></th>
<th>Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>Methods</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>Methods</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Experimental evaluation of LCOM variants

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

<table>
<thead>
<tr>
<th>Etzkorn, Gholston, Fortune, Stein, Utley, Farrington, Cox</th>
<th>Correlation with expert assessment</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Group 1</td>
<td></td>
<td>Group 2</td>
</tr>
<tr>
<td>Chidamber Kemerer</td>
<td>-0.46 (rating 5/8)</td>
<td></td>
<td>-0.73 (rating 1.5/8)</td>
</tr>
<tr>
<td>Henderson-Sellers</td>
<td>-0.44 (rating 7/8)</td>
<td></td>
<td>-0.45 (rating 7/8)</td>
</tr>
<tr>
<td>Hitz, Montazeri</td>
<td>-0.51 (rating 2/8)</td>
<td></td>
<td>-0.54 (rating 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** \( TCC = \frac{NDP}{NP} \)

**Loose class cohesion** \( LCC = \frac{NIP}{NP} \)

- NB: Constructors and destructors are excluded
### Experimental evaluation of LCC/TCC

<table>
<thead>
<tr>
<th></th>
<th>Correlation with expert assessment</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Group 1</td>
</tr>
<tr>
<td>**Etzkorn, Gholston, **</td>
<td></td>
</tr>
<tr>
<td><strong>Fortune, Stein, Utley,</strong></td>
<td>-0.46 (rating 5/8)</td>
</tr>
<tr>
<td><strong>Farrington, Cox</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Chidamber Kemerer</strong></td>
<td>-0.46 (rating 5/8)</td>
</tr>
<tr>
<td><strong>Henderson-Sellers</strong></td>
<td>-0.44 (rating 7/8)</td>
</tr>
<tr>
<td><strong>Hitz, Montazeri</strong></td>
<td>-0.51 (rating 2/8)</td>
</tr>
<tr>
<td><strong>TCC</strong></td>
<td>-0.22 (rating 8/8)</td>
</tr>
<tr>
<td><strong>LCC</strong></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>Method</td>
<td>LOC, McCabe</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>
Metrics for higher-level objects as aggregation of metrics for low-level objects
Aggregation techniques

• **Metrics-independent**
  • Applicable for any metrics to be aggregated
    – Traditional: mean, median...
    – “By no means”
    – Econometric: inequality indices

• **Metrics-dependent**
  • Produce more precise results
  • BUT: need to be redone for any new metrics
  • Based on fitting probability distributions
Metrics independent: Coefficient of variation

- Coefficient of variation: $C = \frac{\sigma}{\mu}$
- Allows to compare distributions with different means
- Sometimes used to assess stability of the metrics
  - Metrics is stable for $C < 0.3$
  - Unreliable for small samples
  - Evolution should be studied…
Metrics are like money

- prog. lang.
- domain
- ...
- region
- education
- gender
- ...

poor econometric values

rich

software metrics

How far?
Popular technique: Gini coefficient

- Gini coefficient measure of economic inequality
- Ranges on $[0; 1 - 1/n]$
- High values indicate high inequality
Gini coefficient: Formally

- Lorenz curve:
  - % of income shared by the lower % of the population

- Gini = A/(A+B)
- Since A+B = 0.5
  Gini = 2A
Gini and software metrics [Vasa et al. 2009]

- For most of the metrics on the benchmark systems: $0.45 \leq \text{Gini} \leq 0.75$
- Higher Gini/WMC: presence of generated code or code, structured in a way similar to the generated code (parsers)
## Gini and metrics: Exceptions

<table>
<thead>
<tr>
<th>System</th>
<th>Metrics</th>
<th>Increase</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>JabRef</td>
<td>WMC</td>
<td>0.75</td>
<td>Machine generated parser introduced</td>
</tr>
<tr>
<td>Checkstyle</td>
<td>Fan-in (classes)</td>
<td>0.44</td>
<td>Plug-in based architecture introduced.</td>
</tr>
<tr>
<td>Jasper-Reports</td>
<td>#Public methods</td>
<td>0.58</td>
<td>Introduction of a set of new base classes.</td>
</tr>
<tr>
<td>WebWork</td>
<td>Fan-out</td>
<td>0.51</td>
<td>A large utility class and multiple cases of copy-and paste introduced.</td>
</tr>
</tbody>
</table>
Rather stable: programmers accumulate competence and tend to solve similar problems by similar means.

Similar for other econometric techniques: Theil, Hoover, Atkinson, ...
Aggregation techniques

- **Metrics-independent**
  - Applicable for any metrics to be aggregated
  - Are the results also metrics-independent?
  - Based on econometrics

- **Metrics-dependent**
  - Produces more precise results
  - BUT: needs to be redone for any new metrics
  - Based on fitting probability distributions
Metrics-dependent aggregation: Statistical fitting

1. Collect the metrics values for the lower-level elements

2. Present a histogram

3. Fit a (theoretical) probability distribution to describe the sample distribution
   a) Select a family of theoretical distributions
   b) Fit the parameters of the probability distribution
   c) Assess the goodness of fit

4. If a theoretical distribution can be fitted, use the fitted parameters as the aggregated value
Step 1: Histograms

- We have seen quite a number of them already!

Robles et al. 2006: LOC in Debian 2.0 (left) and 3.0 (right)
Histograms are not without problems

- Data: 50 birth weights of children with a severe idiopathic respiratory syndrome

- The same data leads to four different “distributions”

- What can affect the way histogram looks like?
  - Bin width
  - Position of the bin’s edges
Kernel density estimators

- **Advantages**
  - Statistically more sound (no dependency on the end-points of the bins)
  - Produces smooth curves

- **Disadvantages**
  - Statistically more complex
  - Parameter tuning might be a challenge
Kernel density estimates: Intuition

- Data: -2.1, -1.3, -0.4, 1.9, 5.1, 6.2

What if each value will be a “bump” that can be added together to create a smooth curve?

Histogram: every value is a rectangle.
Shape is a “sum” of the rectangles.
Kernel density estimation: Formally

\[ f(x) = \frac{1}{nh} \sum_{i=1}^{n} K \left( \frac{x - x_i}{h} \right) \]

Where

- \( n \) – number of observations
- \( h \) – a smoothing parameter, the “bandwidth”
- \( K \) – a weighting function, the “kernel”

Histogram can be obtained using \( \text{Rectangular function} \) as \( K \)

Once \( K \) is chosen one can determine the optimal \( h \).
Histogram as a kernel density estimate

- $h$ is the bin width
- $x$ and $x_i$ are in the same bin if $-1 \leq \frac{x - x_i}{h} \leq 1$
Histogram vs. Kernel density estimate

Histogram

Kernel density estimate

N = 425   Bandwidth = 0.03203
Step 2: fitting a distribution

Tamai, Nakatani. Negative binomial distribution

- Family of distributions is chosen based on shape
- If the parameters fitting is not good enough try a different one!

S, Roubtsov, vd Brand Exponential distribution
Sometimes well-known distributions do not really seem to match

- **Exponential distribution:**
  \[ f(x) = \lambda e^{-\lambda x} \]

- **However, support is [0;1] rather than [0;\infty)!**
  - Since \( \int_0^1 f(x)\,dx = 1 - e^{-\lambda} \)
  - we normalize: \( g(x) = \frac{f(x)}{\int_0^1 f(x)\,dx} \)

- **And use max-likelihood fitting to find \( \lambda \)**
Step 3c. Goodness of fit: Pearson $\chi^2$ test

- The test statistic
  
  \[ X^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i} \]
  
  where
  - $O$ – observed frequency of the result $i$
  - $E$ – expected frequency of the result $i$

- Compare $X^2$ with the theoretical $\chi^2$ distribution for the given number of degrees of freedom: $P(\chi^2 > X^2)$
  - Degrees of freedom = number of observations – number of fitted parameters
  - Comparison is done based on table values
  - If the $P(\chi^2 > X^2) < \text{threshold}$ – the fit is good
  - Common thresholds are 0.1, 0.05 and 0.01
Recapitulation: Statistical fitting

1. Collect the metrics values for the lower-level elements
2. Present a histogram
3. Fit a (theoretical) probability distribution to describe the sample distribution
   a) Select a family of theoretical distributions
   b) Fit the parameters of the probability distribution
   c) Assess the goodness of fit
4. If a theoretical distribution can be fitted, use the fitted parameters as the aggregated value
What about the evolution of the aggregated values?

- Geometry library: Jun, subsystem “Geometry”
- Tamai, Nakatani: Negative binomial distribution
  \[ f(x) = \binom{x-1}{k-1} p^k (1 - p)^{x-k} \]
- \( p, k \) – distribution parameters
- \( \binom{x-1}{k-1} \) - binomial coefficient extended to the reals
- Increase – functionality enhancement
- Decrease – refactoring
In general, how do we study evolution?

- Visual inspection
- Is this a real “trend” or just noise?
In general, how do we study evolution?

- **Time-series analysis**
- **Simplest form: linear regression with time**

**Linear trend**

**Significant**

**Strong**

More advanced techniques:

2DD23 - Time series analysis and forecasting
Conclusions: Aggregation

• Aggregation:
  • Metrics-independent
    – Applicable for any metrics to be aggregated
      – Traditional: mean, median...
      – “By no means”
    – Econometric: inequality indices

• Metrics-dependent
  – Produce more precise results
  – BUT: need to be redone for any new metrics
  – Based on fitting probability distributions
## Metrics so far...

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

• Size:
  • number of classes/interfaces
  • number of classes in the subpackages

• Dependencies
  • visualization
  • à la fan-in and fan-out
    − Marchesi’s UML metrics
    − Martin’s $D_n$: abstractness-instability balance or “the normalized distance from the main sequence”
    − PASTA

• Do you still remember aggregations of class metrics?
“Fan-out”

\[ \text{PK}_1 \text{ or R: } 5 \]

\[ C_e: 1 \]

[Marchesi 1998] [Martin 2000]

[Marchin 1994] [JDepend] [Martin 2000]
Fan-in

• “Fan-in” similarly to the “Fan-out”
  • Afferent coupling (Martin)
  • PK₂ (Marchesi)

- Dark: TDD, light: no-TDD
- Test-driven development positively affects Cₐ
  • The lower Cₐ - the better.
- Exception: JUnit vs. Jericho
  • But Jericho is extremely small (2 packages)

[Hilton 2009]
More fan-in and fan-out

- “Fan-in” similarly to the “Fan-out”
- Afferent coupling (Martin)
- PK$_2$ (Marchesi)

<table>
<thead>
<tr>
<th>Marchesi</th>
<th>Man-months</th>
<th>#Pack</th>
<th>avg(PK$_1$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Railway simulator</td>
<td>13</td>
<td>6</td>
<td>8.7</td>
</tr>
<tr>
<td>Warehouse management</td>
<td>7</td>
<td>5</td>
<td>5.0</td>
</tr>
<tr>
<td>CASE tool</td>
<td>13</td>
<td>5</td>
<td>8.1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Correlation post-release defects</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAP (Herzig)</td>
<td>0.091</td>
</tr>
<tr>
<td>Afferent</td>
<td>0.091</td>
</tr>
<tr>
<td>Efferent [Martin 2000]</td>
<td>0.157</td>
</tr>
<tr>
<td>Class-in</td>
<td>0.084</td>
</tr>
<tr>
<td>Class-out</td>
<td>0.086</td>
</tr>
<tr>
<td>Fan-in</td>
<td>0.287</td>
</tr>
<tr>
<td>Fan-out</td>
<td>0.148</td>
</tr>
</tbody>
</table>
Evolution of afferent and efferent coupling

- Almost all systems show an increasing trend (Lehman’s growing complexity)
- Project 7 (workflow system) is almost stable but very high!
  - Outsourced development
  - No automated tests
  - Severe maintainability problems

Sato, Goldman, Kon 2007
Package metrics: Stability

Stability is related to the amount of work required to make a change [Martin, 2000].

- **Stable** packages
  - Do not depend upon classes outside
  - Many dependents
  - Should be extensible via inheritance (*abstract*)

- **Instable** packages
  - Depend upon many classes outside
  - No dependents
  - Should not be extensible via inheritance (*concrete*)
What does balance mean?

A good real-life package must be **instable** enough in order to be easily modified.

It must be **generic** enough to be adaptable to evolving requirements, either without or with only minimal modifications.

Hence: contradictory criteria.
$D_n$ – Distance from the main sequence

Abstractness = \#AbstrClasses/\#Classes

Instability = $\frac{C_e}{C_e + C_a}$

$D_n = | \text{Abstractness} + \text{Instability} - 1 |$

[R.Martin 1994]
Normalized distance from the main sequence

- Dark: TDD, light: no-TDD
- Test-driven development positively affects $D_n$
  - The lower $D_n$ - the better.
- The same exception (Jericho vs. JUnit)

[Hilton 2009]
Distribution and evolution

Exponential distribution

For all benchmark systems studied, here Vuze 4.0.0.4

Peak: many feature requests (average Dn)

JBoss

- Diagram showing frequency distribution and exponential distribution with peak at many feature requests (average Dn)
PASTA [Hautus 2002]

- **PASTA – Package structure analysis tool**

- **Metrics**
  - Similarly “fan-in/fan-out”: based on dependencies between packages
  - Go beyond calculating numbers of dependencies

- **Focus on dependencies between the subpackages**
- Some dependencies are worse than others
  - What are the “bad dependencies”?
  - Cyclic dependencies, layering violations
PASTA [Hautus]

Idea: remove bad (cycle-causing) dependencies

- **Weight** – number of references from one subpackage to another one.
- Dependencies to be removed are such that
  - The result is acyclic
  - The total weight of the dependencies removed is minimal
- Minimal effort required to resolve all the cycles

Upwards dependencies should be removed
From dependencies to metrics

- PASTA(P) = Total weight of the dependencies to be removed / total weight of the dependencies

- No empirical validation of the metrics

- No studies of the metrics evolution

<table>
<thead>
<tr>
<th>Package</th>
<th>PASTA Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>junit</td>
<td>0%</td>
</tr>
<tr>
<td>org.apache.batik</td>
<td>0%</td>
</tr>
<tr>
<td>org.apache.tools.ant</td>
<td>1%</td>
</tr>
<tr>
<td>java</td>
<td>5%</td>
</tr>
<tr>
<td>org.apache.jmeter</td>
<td>6%</td>
</tr>
<tr>
<td>javax.swing</td>
<td>10%</td>
</tr>
<tr>
<td>org.jboss</td>
<td>11%</td>
</tr>
<tr>
<td>org.gjt.sp.jedit</td>
<td>18%</td>
</tr>
<tr>
<td>java.awt</td>
<td>20%</td>
</tr>
</tbody>
</table>
One metric is good, more metrics are better (?)

- Recall...

\[ M1 = 171 - 5.2 \ln(V) - 0.23V(g) - 16.2 \ln(LOC) \]

- [Kaur, Singh 2011] propose an adaptation...

\[ MIP = 171 - 5.2CC - 0.23 \ln(S) - 16.2 \ln(NC) \]
Summary: package metrics

- **Size**: number of classes

- **Dependencies à la fan-in and fan-out**
  - Marchesi’s UML metrics
  - Martin’s $D_n$: abstractness-instability balance or “the normalized distance from the main sequence”
  - PASTA

- **Aggregations of class metrics**: reminder
  - Metrics independent: average, sum, Gini/Theil coefficients
  - Metrics dependent: Distribution fitting
Measuring change: Churn metrics

• Why? Past evolution to predict future evolution

• Code Churn [Lehman, Belady 1985]:
  • Amount of code change taking place within a software unit over time

• Code Churn metrics [Nagappan, Bell 2005]:

Absolute:
Churned LOC, Deleted LOC, File Count, Weeks of Churn, Churn Count, Files Churned

Relative:

- $M_1$: Churned LOC / Total LOC
- $M_2$: Deleted LOC / Total LOC
- $M_3$: Files churned / File count
- $M_4$: Churn count / Files churned
- $M_5$: Weeks of churn / File count
- $M_6$: Lines worked on / Weeks of churn
- $M_7$: Churned LOC / Deleted LOC
- $M_8$: Lines worked on / Churn count
Case Study: Windows Server 2003

• Analyze Code Churn between WS2003 and WS2003-SP1 to predict defect density in WS2003-SP1
  • 40 million LOC, 2000 binaries
  • Use absolute and relative churn measures

• Conclusion 1: Absolute measures are no good
  • $R^2 < 0.05$

• Conclusion 2: Relative measures are good!
  • An increase in relative code churn measures is accompanied by an increase in system defect density
  • $R^2 \approx 0.8$
Case Study: Windows Server 2003

- Construct a statistical model
  - Training set: 2/3 of the Windows Set binaries
- Check the quality of the prediction
  - Test set: remaining binaries
- Three models
  - Right: all relative churn metrics are taken into account
Open issues

• To predict bugs from history, but we need a history filled with bugs to do so
  • Ideally, we don’t have such a history

• We would like to learn from previous projects:
  • Can we make predictions without history?
  • How can we leverage knowledge between projects?
  • Are there universal properties?

• Not just code properties but also properties of the entire software process
Conclusions

• Package metrics
  • Directly defined: $D_n$, Marchesi metrics, PASTA
  • Results of aggregation

• Churn metrics