Software Metrics

Software Engineering

Definitions

• **Measure** - quantitative indication of extent, amount, dimension, capacity, or size of some attribute of a product or process.
  – Number of errors

• **Metric** - quantitative measure of degree to which a system, component or process possesses a given attribute. "A handle or guess about a given attribute."
  – Number of errors found per person hours expended

Why Measure Software?

• Determine quality of the current product or process

• Predict qualities of a product/process

• Improve quality of a product/process

Example Metrics

• Defects rates
• Errors rates
• Measured by:
  – individual
  – module
  – during development
• Errors should be categorized by origin, type, cost
Metric Classification

• Products
  – Explicit results of software development activities.
  – Deliverables, documentation, by products
• Processes
  – Activities related to production of software
• Resources
  – Inputs into the software development activities
  – hardware, knowledge, people

Product vs. Process

• Process Metrics-
  – Insights of process paradigm, software engineering tasks, work product, or milestones.
  – Lead to long term process improvement.
• Product Metrics-
  – Assesses the state of the project
  – Track potential risks
  – Uncover problem areas
  – Adjust workflow or tasks
  – Evaluate teams ability to control quality

Types of Measures

• Direct Measures (internal attributes)
  – Cost, effort, LOC, speed, memory
• Indirect Measures (external attributes)
  – Functionality, quality, complexity, efficiency, reliability, maintainability

Size Oriented Metrics

• Size of the software produced
• Lines Of Code (LOC)
• 1000 Lines Of Code KLOC
• Effort measured in person months
• Errors/KLOC
• Defects/KLOC
• Cost/LOC
• Documentation Pages/KLOC
• LOC is programmer & language dependent
LOC Metrics

- Easy to use
- Easy to compute
- Can compute LOC of existing systems but cost and requirements traceability may be lost
- Language & programmer dependent

Function Oriented Metrics

- Function Point Analysis [Albrecht '79, '83]
- International Function Point Users Group (IFPUG)
- Indirect measure
- Derived using empirical relationships based on countable (direct) measures of the software system (domain and requirements)

Computing Functions Points

- Number of user inputs
  - Distinct input from user
- Number of user outputs
  - Reports, screens, error messages, etc
- Number of user inquiries
  - On line input that generates some result
- Number of files
  - Logical file (database)
- Number of external interfaces
  - Data files/connections as interface to other systems

Compute Function Points

- \( FP = \text{Total Count} \times [0.65 + .01\times\text{Sum}(F_i)] \)
- Total count is all the counts times a weighting factor that is determined for each organization via empirical data
- \( F_i \) (i=1 to 14) are complexity adjustment values
Complexity Adjustment

- Does the system require reliable backup and recovery?
- Are data communications required?
- Are there distributed processing functions?
- Is performance critical?
- Will the system run in an existing heavily utilized operational environment?
- Does the system require on-line data entry?
- Does the online data entry require the input transaction to be built over multiple screens or operations?

Complexity Adjustment (cont)

- Are the master files updated on line?
- Are the inputs, outputs, files, or inquiries complex?
- Is the internal processing complex?
- Is the code designed to be reusable?
- Are conversions and installations included in the design?
- Is the system designed for multiple installations in different organizations?
- Is the application designed to facilitate change and ease of use by the user?

Using FP

- Errors per FP
- Defects per FP
- Cost per FP
- Pages of documentation per FP
- FP per person month

FP and Languages

<table>
<thead>
<tr>
<th>Language</th>
<th>LOC/FP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assembly</td>
<td>320</td>
</tr>
<tr>
<td>C</td>
<td>128</td>
</tr>
<tr>
<td>COBOL</td>
<td>106</td>
</tr>
<tr>
<td>FORTRAN</td>
<td>106</td>
</tr>
<tr>
<td>Pascal</td>
<td>90</td>
</tr>
<tr>
<td>C++</td>
<td>64</td>
</tr>
<tr>
<td>Ada</td>
<td>53</td>
</tr>
<tr>
<td>VB</td>
<td>32</td>
</tr>
<tr>
<td>SQL</td>
<td>12</td>
</tr>
</tbody>
</table>
Using FP

- FP and LOC based metrics have been found to be relatively accurate predictors of effort and cost
- Need a baseline of historical information to use them properly
- Language dependent
- Productivity factors: People, problem, process, product, and resources
- FP can not be reverse engineered from existing systems easily

Complexity Metrics

- LOC - a function of complexity
- Language and programmer dependent
- Halstead’s Software Science (entropy measures)
  - $n_1$ - number of distinct operators
  - $n_2$ - number of distinct operands
  - $N_1$ - total number of operators
  - $N_2$ - total number of operands

Example

```java
if (k < 2)
{
    if (k > 3)
        x = x*k;
}
```

- Distinct operators: if ( ) { } > < = * ;
- Distinct operands: k 2 3 x
- $n_1 = 10$
- $n_2 = 4$
- $N_1 = 13$
- $N_2 = 7$

Halstead’s Metrics

- Amenable to experimental verification [1970s]
- Length: $N = N_1 + N_2$
- Vocabulary: $n = n_1 + n_2$
- Estimated length: $\hat{N} = n_1 \log_2 n_1 + n_2 \log_2 n_2$
  - Close estimate of length for well structured programs
- Purity ratio: $PR = \hat{N} / N$
Program Complexity

- **Volume**: $V = N \log_2 n$
  - Number of bits to provide a unique designator for each of the $n$ items in the program vocabulary.

- **Program effort**: $E = V/L$
  - $L = V^*/V$
  - $V^*$ is the volume of most compact design implementation
  - This is a good measure of program understandability

McCabe’s Complexity Measures

- McCabe’s metrics are based on a control flow representation of the program.
- A program graph is used to depict control flow.
- Nodes represent processing tasks (one or more code statements)
- Edges represent control flow between nodes

Flow Graph Notation

- **Sequence**
- **If-then-else**
- **While**
- **Until**

Cyclomatic Complexity

- Set of independent paths through the graph (basis set)
- $V(G) = E - N + 2$
  - $E$ is the number of flow graph edges
  - $N$ is the number of nodes
- $V(G) = P + 1$
  - $P$ is the number of predicate nodes
Example

\begin{verbatim}
i = 0;
while (i<n-1) do
  j = i + 1;
  while (j<n) do
    if A[i]<A[j] then
      swap(A[i], A[j]);
    end do;
  i=i+1;
end do;
\end{verbatim}

Computing V(G)

- \( V(G) = 9 - 7 + 2 = 4 \)
- \( V(G) = 3 + 1 = 4 \)
- Basis Set
  - 1, 7
  - 1, 2, 6, 1, 7
  - 1, 2, 3, 4, 5, 2, 6, 1, 7
  - 1, 2, 3, 5, 2, 6, 1, 7

Another Example

\begin{verbatim}
\end{verbatim}
Meaning

- \( V(G) \) is the number of (enclosed) regions/areas of the planar graph
- Number of regions increases with the number of decision paths and loops.
- A quantitative measure of testing difficulty and an indication of ultimate reliability
- Experimental data shows value of \( V(G) \) should be no more than 10. Testing is very difficult above this value.

McClure’s Complexity Metric

- Complexity = \( C + V \)
  - \( C \) is the number of comparisons in a module
  - \( V \) is the number of control variables referenced in the module
- Similar to McCabe’s but with regard to control variables.

Metrics and Software Quality

**FURPS**

- Functionality - features of system
- Usability - aesthetics, documentation
- Reliability - frequency of failure, security
- Performance - speed, throughput
- Supportability - maintainability

Measures of Software Quality

- Correctness
  - Defects/KLOC
  - Defect is a verified lack of conformance to requirements
  - Failures/hours of operation
- Maintainability
  - Mean time to change
  - Change request to new version (Analyze, design etc)
  - Cost to correct
- Integrity
  - Fault tolerance, security & threats
- Usability
  - Training time, skill level necessary to use. Increase in productivity, subjective questionnaire or controlled experiment
Quality Model

High level Design Metrics

- Structural Complexity
- Data Complexity
- System Complexity
- Card & Glass ‘80

- Structural Complexity $S(i)$ of a module $i$.
  - $S(i) = f_{out}(i)$
  - Fan out is the number of modules immediately subordinate (directly invoked).

Design Metrics

- Data Complexity $D(i)$
  - $D(i) = v(i)/(f_{out}(i)+1)$
  - $v(i)$ is the number of inputs and outputs passed to and from $i$.
- System Complexity $C(i)$
  - $C(i) = S(i) + D(i)$
  - As each increases the overall complexity of the architecture increases.

System Complexity Metric

- Another metric:
  - $\text{length}(i) \times [f_{in}(i) + f_{out}(i)]^2$
  - Length is LOC
  - Fan in is the number of modules that invoke $i$.
- Graph based:
  - Nodes + edges
  - Modules + lines of control
  - Depth of tree, arc to node ratio
Coupling

- Data and control flow
  - $d_i$ – input data parameters
  - $c_i$ – input control parameters
  - $d_o$ – output data parameters
  - $c_o$ – output control parameters
- Global
  - $g_d$ – global variables for data
  - $g_c$ – global variables for control
- Environmental
  - $w$ – fan in number of modules called
  - $r$ – fan out number modules that call module

Metrics for Coupling

- $M_c = k/m$, $k=1$
  - $m = d_i + ac_i + d_o + bc_o + g_d + g_c + w + r$
  - $a, b, c, k$ can be adjusted based on actual data

Component Level Metrics

- Cohesion (internal interaction)
- Coupling (external interaction)
- Complexity of program flow
- Cohesion – difficult to measure
  - Bieman ’94, TSE 20(8)
  - Data slice – from a program slice

Using Metrics

- The Process
  - Select appropriate metrics for problem
  - Utilized metrics on problem
  - Assessment and feedback
- Formulate
- Collect
- Analysis
- Interpretation
- Feedback
### Metrics for the Object Oriented
- Chidamber & Kemerer ’94 TSE 20(6)
- Metrics specifically designed to address object oriented software
- Class oriented metrics
- Direct measures

### Weighted Methods per Class
$$WMC = \sum c_i$$
- $c_i$ is the complexity (e.g., volume, cyclomatic complexity, etc.) of each method
- Must normalize
- What about inherited methods?
  - Be consistent

### Depth of Inheritance Tree
- DIT is the maximum length from a node to the root (base class)
- Lower level subclasses inherit a number of methods making behavior harder to predict
- However, more methods are reused in higher DIT trees.

### Number of Children
- NOC is the number of subclasses immediately subordinate to a class
- As NOC grows, reuse increases
- But the abstraction may be diluted
Coupling between Classes

- CBO is the number of collaborations between two classes
- As collaboration increases reuse decreases
- CRC – lists the number of collaborations
  - Classes, Responsibilities, and Collaborations

Response for a Class

- RFC is the number of methods that could be called in response to a message to a class
- Testing effort increases as RFC increases

Lack of Cohesion in Methods

- LCOM – poorly described in Pressman
- Class Cₙ with n methods M₁,…Mₙ
- Iᵢ is the set of instance variables used by Mᵢ

LCOM

- There are n such sets I₁,…, Iₙ
  - P = { (Iᵩ, Iᵢ) | (Iᵩ ∩ Iᵢ) = ∅ }  
  - Q = { (Iᵩ, Iᵢ) | (Iᵩ ∩ Iᵢ) ≠ ∅ }  
- If all n sets Iᵢ are Ø then P = Ø
- LCOM = |P| - |Q|, if |P| > |Q|
- LCOM = 0 otherwise
Example LCOM

- Take class C with M1, M2, M3
- \( I_1 = \{a, b, c, d, e\} \)
- \( I_2 = \{a, b, e\} \)
- \( I_3 = \{x, y, z\} \)
- \( P = \{(I_1, I_3), (I_2, I_3)\} \)
- \( Q = \{(I_1, I_2)\} \)
- Thus LCOM = 1

Explanation

- LCOM is the number of empty intersections minus the number of non-empty intersections
- This is a notion of degree of similarity of methods.
- If two methods use common instance variables then they are similar
- LCOM of zero is not maximally cohesive
- \(|P| = |Q|\) or \(|P| < |Q|\)

Class Size

- CS
  - Total number of operations (inherited, private, public)
  - Number of attributes (inherited, private, public)
- May be an indication of too much responsibility for a class

Number of Operations Overridden

- NOO
- A large number for NOO indicates possible problems with the design
- Poor abstraction in inheritance hierarchy
Number of Operations Added

- NOA

- The number of operations added by a subclass
- As operations are added it is farther away from super class
- As depth increases NOA should decrease

Specialization Index

\[ SI = \frac{[NOO \times L]}{M_{total}} \]

- L is the level in class hierarchy
- \( M_{total} \) is the total number of methods
- Higher values indicate class in hierarchy that does not conform to the abstraction

Method Inheritance Factor

\[ MIF = \frac{\sum_{i} M_i(C_i)}{\sum_{i} M_j(C_i)} \]

- \( M_i(C_i) \) is the number of methods inherited and not overridden in \( C_i \)
- \( M_j(C_i) \) is the number of methods that can be invoked with \( C_i \)
- \( M_d(C_i) \) is the number of methods declared in \( C_i \)

MIF

- \( M_a(C_i) = M_d(C_i) + M_i(C_i) \)
- All that can be invoked = new or overloaded + things inherited

- MIF is [0,1]
- MIF near 1 means little specialization
- MIF near 0 means large change
Coupling Factor

\[ CF = \frac{\sum \sum \text{is\_client}(C_i, C_j)}{(TC^2 - TC)} \]

- \( \text{is\_client}(x, y) = 1 \) iff a relationship exists between the client class and the server class. 0 otherwise.
- \((TC^2 - TC)\) is the total number of relationships possible (Total Classes\(^2 \) – diagonal)
- CF is \([0,1]\) with 1 meaning high coupling

Polymorphism Factor

\[ PF = \frac{\sum M_n(C_i)}{\sum [M_n(C_i) \times DC(C_i)]} \]

- \( M_n() \) is the number of new methods
- \( M_o() \) is the number of overriding methods
- DC() number of descendent classes of a base class
- The number of methods that redefines inherited methods, divided by maximum number of possible distinct polymorphic situations

Operational Oriented Metrics

- Average operation size (LOC, volume)
- Number of messages sent by an operator
- Operation complexity – cyclomatic
- Average number of parameters/operation
  - Larger the number the more complex the collaboration

Encapsulation

- Lack of cohesion
- Percent public and protected
- Public access to data members
Inheritance

• Number of root classes

• Fan in – multiple inheritance

• NOC, DIT, etc.