CSC 440/540: Software Engineering

Software Metrics
Topics

1. Software Metrics
2. Product Metrics
   1. Size metrics
   2. Defect density and compound metrics
   3. Complexity metrics
   4. CK object-oriented design metrics
   5. Testing metrics
   6. Metrics tools
3. Process Metrics
A **software metric** is a “quantitative measure of the degree to which a system, component, or process possesses a given attribute.”

— IEEE Glossary of Software Engineering Terminology
Why Software Metrics

- Help software engineers to gain insight into the design and construction of the software.
- Focus on specific attributes of software engineering work products resulting from analysis, design, coding, and testing.
- Provide a systematic way to assess quality based on a set of clearly defined rules.
- Provide an “on-the-spot” rather than “after-the-fact” insight into the software development.
Characteristics of Good Metrics

- **Intuitive:** mathematical values should have natural values with an intuitive scale. For example, a metric to measure quality should be high for good quality software and low for bad quality software.

- **Consistent and objective:** the metric should have the same values no matter who computes it.

- ** Easily obtainable:** should be automatically computable in a reasonable amount of time.

- **Valid:** the metric should measure what it is intended to measure.

- **Robust:** relatively insensitive to insignificant changes in the process or product.
Uses of Metrics

- Minimize a schedule.
- Stay on schedule.
- Reduce the number of bugs.
- Predict the number of bugs that will arise.
- Make bug fixing easier.
- Assess ongoing quality.
- Improve finished results.
- Improve maintenance.
- Detect risks such as schedule slips and adjust staffing and work effort to address them.
How to Use Metrics

- Identify that a project is healthy
- Document project trends
- Provide regular feedback to management.
  - Management and team want different sets of metrics.
- Don’t use to evaluate performance of devs or teams
  - Or you won’t get accurate metric data in the future.
Product Metrics

- Help software engineers assess quality of software and better understand design and construction of software.
- Measure specific attributes of software engineering
  - Analysis
  - Design
  - Implementation
  - Testing
- Assess project as it is being developed.
Source Lines Of Code (SLOC)

Definitions
- All lines of code
- Non-blank non-comment lines
- Lines with executable statements only

Applications
- Effort estimation
- Used to normalize other software metrics

Problems
- Difference in languages
- Developer experience impacts
- IDE boilerplate code
Function Points (FP)

Measure size based on features (functions) from a user point of view in a way independent of technology used.

Inputs to FP computation:

- number of external inputs (EIs)
- number of external outputs (EOs)
- number of external inquiries (EQs)
- number of internal logical files (ILFs)
- number of external interface files (EIFs)
FP Inputs

**External Inputs (EI)**
Create Invoice

**External Outputs (EO)**
Generate Balance Sheet (with calculated data)

**External Inquiries (EQ)**
Validate Login

---

<table>
<thead>
<tr>
<th>Function</th>
<th>Low</th>
<th>Average</th>
<th>High</th>
</tr>
</thead>
<tbody>
<tr>
<td>ILF</td>
<td>7</td>
<td>10</td>
<td>15</td>
</tr>
<tr>
<td>EIF</td>
<td>5</td>
<td>7</td>
<td>10</td>
</tr>
<tr>
<td>EI</td>
<td>3</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>EO</td>
<td>4</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>EQ</td>
<td>3</td>
<td>4</td>
<td>6</td>
</tr>
</tbody>
</table>

Data shown is illustrative only.
## Weighting FPs

<table>
<thead>
<tr>
<th>Information Domain Value</th>
<th>Count</th>
<th>Weighting factor</th>
<th>Simple</th>
<th>Avg.</th>
<th>Complex</th>
</tr>
</thead>
<tbody>
<tr>
<td>External Inputs (EIs)</td>
<td>X</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>External Outputs (EOs)</td>
<td>X</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>External Inquiries (EQs)</td>
<td>X</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Internal Logic Files (ILFs)</td>
<td>X</td>
<td>7</td>
<td>10</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>External Interface Files (EIFs)</td>
<td>X</td>
<td>5</td>
<td>7</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Count Total</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Relating SLOC and FPs

<table>
<thead>
<tr>
<th>Language</th>
<th>Nominal Level</th>
<th>Low</th>
<th>Mean</th>
<th>High</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic Assembly</td>
<td>1.00</td>
<td>200</td>
<td>320</td>
<td>450</td>
</tr>
<tr>
<td>C</td>
<td>2.5</td>
<td>60</td>
<td>128</td>
<td>170</td>
</tr>
<tr>
<td>C++</td>
<td>6</td>
<td>30</td>
<td>53</td>
<td>125</td>
</tr>
<tr>
<td>Visual Basic</td>
<td>10</td>
<td>20</td>
<td>32</td>
<td>37</td>
</tr>
</tbody>
</table>

Source: Int'l Function Point Users Group Rules, Version 4.1
Additional Size Metrics

Implementation Size
- Number of functions
- Number of web pages
- Number of internal page links
- Number of persistent data objects
- ABC metric counts assignments, branches, + conditionals

Design Size
- Number of classes
- Number of files

Project Size (Requirements)
- Number of use cases
- Number of domain classes
Defect Density

Definitions

- # defects / kLOC
- # defects / class
- # defects / file
- # defects / FP

Problems

- Treats defects equally, but defects have different impacts.
Additional Compound Metrics

Quality metrics
- Defect density
- Defects introduced per person-month

Cost metrics
- Cost($) / size unit

Work metrics
- Documentation / size unit
- Tests / size unit
- LOC per person-month
- Defects removed per person-month
Complexity Metrics

Nesting Complexity
- Maximum or average depth of nesting loops or if’s.

Cyclomatic Complexity
- Measures independent paths through program.

Halstead Complexity Metrics
- Counts statement parts (operators and operands).
- Combines into metrics in different ways.
Control Flow Graph

a) If/then/else statement.
b) While loop
c) Loop with two exits.
d) Loop with multiple entry points via goto or similar features.
Cyclomatic Complexity

Count of linearly independent paths through source code. Defined based on program Control Flow Graph

\[ M = E - N + 2P, \]

where

- \( E \) = number of graph edges
- \( N \) = number of graph nodes
- \( P \) = \# connected components

\[ CC = E - N + 2P \]
\[ = 3 - 3 + 2*1 \]
Halstead Complexity Metrics

Based on counts of

- Operators: \( N_1 = \text{total}, \ n_1 = \text{unique} \)
- Operands: \( N_2 = \text{total}, \ n_2 = \text{unique} \)

Metrics

- Halstead Length: \( N = N_1 + N_2 \)
- Program vocabulary: \( n = n_1 + n_2 \)
- Halstead Volume: \( V = N \log_2 n \)
- Difficulty: \( D = \frac{n_1}{2} \cdot \frac{N_2}{n_2} \)
- Effort: \( E = VD \)
Structural and Data Complexity

Based on two simpler metrics:

- **Fan-In**: the number of other components called by this component.
- **Fan-Out**: the number of other components that call this component.

**Metrics**

- Structural Complexity: \( S = (\text{fan-out})^2 \)
- Data Complexity: \( D = (\text{number of input vars})/(\text{fan-out} + 1) \)
- System Complexity: \( C = S + D \)
CK OO Design Metrics

- **Weighted Methods per Class (WMC):** Sum of complexity metric values for all methods in class.
- **Depth of Inheritance Tree (DIT):** Maximum number of classes from superclass to subclass in class hierarchy.
- **Number of Children (NOC):** Number of immediate subclasses of a class.
- **Coupling Between Object classes (CBO):** Number of collaborations for a class listed on CRC design card.
- **Response For a Class (RFC):** Number of methods that can potentially be executed in response to a message received by an object of that class.
- **Lack of Cohesion in Methods (LCOM):** Number of methods that access one or more of the same attributes.
Testing Metrics

- Requirement coverage
  - Percentage of requirements tested.

- Code coverage
  - Percentage of lines of code tested.
  - Note that 100% coverage does not mean that you have sufficient tests--the same line of code can produce different results based on different program states.

- Defect open and close rates
  - How fast are testing defects reported and repaired.

- Percent of test cases passed
  - Changes with addition of test cases and code fixes.
Metrics Tools

- Ndepend for .NET code
- SonarQube for Java and many other languages
- Scitools Understand for C/C++ and other languages
- Gems for Rails
  - Coco, Rcov, and SimpleCov: unit test code coverage
  - Flog: ABC-like metric that counts language-specific features like metaprogramming.
  - metric_abc: ABC metric
  - Rubocop: checks compliance w/ ruby style guide
  - Saikuro: cyclomatic complexity
- rails stats counts LOC, classes, methods.
Process Metrics

- Designed to be recorded over a long period of time over all projects to drive software process improvement.

- Measure
  - Defects uncovered pre/post release
  - Effort expended (person-months)
  - Calendar time required
  - Conformance to schedule
  - Productivity and efficiency
  - Traceability between requirements, design, code, test.
References
