Software Metrics

“How do we measure the software?”

Lecture Objectives

- To understand the importance of measurement in software engineering
- To describe and compare the different metrics that can be used for measuring software
- To understand the important factors that affect the measurement of software

Engineering Method

- Scientific
- Precision
- Accuracy
- Repeatable
- Controllable
- Quality

Why Do We Measure?

- To indicate the quality of the product
- To assess the productivity of the people who produce the product
- To assess the benefits derived from new software engineering methods and tools
- To form a baseline for estimation
- To help justify requests for new tools or additional training

Definitions

- Measure - quantitative indication of the extent, amount, dimension, capacity, or size of some attribute of a product or process.
- Measurement - the act of determining a measure
- Metric - a quantitative measure of the degree to which a system, component, or process possesses a given attribute (IEEE)

What Can Be Measured?

- Direct measures
  - Lines of codes (LOC), speed, cost, memory size, errors, ...
- Indirect measures
  - Quality, functionality, complexity, reliability, efficiency, maintainability, ...

Example: length of a pipe is a direct measure. the quality of the pipes can only be measured indirectly by finding the ratio of the accepted pipes to the rejected.
**Size-Oriented Metrics**

- Based on the "size" of the software produced.
- Project data measured, including cost and effort, pages, defects...etc.
- Mainly uses the LOC as the normalization value.
- Advantages: easily counted, large body of literature and data based on LOC.
- Disadvantages: language dependent, programmer dependent.
- Useful for projects with similar environment.
- Therefore, size-oriented metrics are not universally accepted as the best way to measure the software process.

<table>
<thead>
<tr>
<th>Project</th>
<th>Effort (person-month)</th>
<th>Cost ($)</th>
<th>LOC</th>
<th>Doc. (pgs)</th>
<th>Errors</th>
<th>People</th>
</tr>
</thead>
<tbody>
<tr>
<td>aaa-01</td>
<td>24</td>
<td>168,000</td>
<td>12.1</td>
<td>365</td>
<td>29</td>
<td>3</td>
</tr>
<tr>
<td>ccc-03</td>
<td>62</td>
<td>440,000</td>
<td>27.2</td>
<td>1224</td>
<td>86</td>
<td>5</td>
</tr>
</tbody>
</table>

**Example of Size-Oriented Metrics**

- Productivity = Size / Effort
  - kLOC / person-month
- Quality = Errors / Size
  - Errors / kLOC
- Cost = $ / kLOC
- Documentation = pages / kLOC
- Other metrics can also be developed like: errors/kLOC, page/kLOC...etc.
- Or errors/person-month, LOC/person-month, cost/page.

**Function-Oriented Metrics**

- Based on "functionality" delivered by the software as the normalization value.
- Functionality is measured indirectly.
- Function points (FP) measure derived using an empirical relationship based on countable (direct) measures of software's information domain and assessments of software complexity.

**Software Information Domain Values**

- Number of user inputs: user inputs.
- Number of user outputs: reports, screens, error messages, etc.
- Number of user inquiries: an on-line input that generates an immediate on-line output response.
- Number of files: each logical (logical grouping of data) master file is counted.
- Number of external interfaces: all machine readable interfaces (e.g. data files on some storage media) that are used to transmit information to another system are counted.

**Steps In Calculating FP**

1. Count the information domain values.
2. Assess the complexity of the values.
3. Calculate the raw FP (see next table).
4. Rate the complexity factors to produce the complexity adjustment value (CAV)
   \[ CAV = \sum \alpha F_i \quad i = 1 \text{to} 14 \]
5. Calculate the adjusted FP as follows:
   \[ FP = \text{raw FP} \times (0.65 + 0.01 \times CAV) \]
**Function Point (FP) Metrics**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Count</th>
<th>Simple</th>
<th>Average</th>
<th>Complex</th>
<th>Weighting Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inputs</td>
<td>x 3</td>
<td>4</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Outputs</td>
<td>x 4</td>
<td>5</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Inquiries</td>
<td>x 3</td>
<td>4</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Files</td>
<td>x 7</td>
<td>10</td>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interfaces</td>
<td>x 5</td>
<td>7</td>
<td>10</td>
<td></td>
<td>Count-total (raw FP)</td>
</tr>
</tbody>
</table>

**Rate Complexity Factors**

For each complexity adjustment factor (F), give a rating on a scale of 0 to 5

0 - No influence
1 - Incidental
2 - Moderate
3 - Average
4 - Significant
5 - Essential

**Complexity Adjustment Factors (Continued)**

1. Does the system require reliable backup and recovery?
2. Are data communications required?
3. Are there distributed processing functions?
4. Is performance critical?
5. Will the system run in an existing, heavily utilized operational environment?
6. Does the system require on-line data entry?
7. Does the on-line data entry require the input transaction to be built over multiple screens or operations?
8. Are the master files updated on-line?
9. Are the inputs, outputs, files, or inquiries complex?
10. Is the internal processing complex?
11. Is the code designed to be reusable?
12. Are conversion and installation included in the design?
13. Is the system designed for multiple installations in different organizations?
14. Is the application designed to facilitate change and ease of use by the user?

**Complexity Adjustment Value**

The rating for all the factors, F₁ to F₁₄, are summed to produce the complexity adjustment value (CAV)

CAV is then used in the calculation of the function point (FP) of the software
**Example of Function-Oriented Metrics**

- **Productivity** = Functionality / Effort
  = FP / person-month
- **Quality** = Errors / Functionality
  = Errors / FP
- **Cost** = $ / FP
- **Documentation** = pages / FP

**FP Characteristics**

- **Advantages**: language independent, based on data known early in project, good for estimation
- **Disadvantages**: calculation complexity, subjective assessments, FP has no physical meaning (just a number)

**Extended Function Point Metrics**

- **Feature points**
  - Applied to systems and engineering software
  - Includes assessment for complex algorithms
- **3D Function points**
  - Applied to real-time systems and engineered products (by Boeing)
  - Integrates data dimension (normal FP) with functional and control dimensions

**Reconciling Metrics Approaches**

- A number of studies attempted to relate FP and LOC measures
- Relationship depends on programming language and quality of design
- Example: one LOC of Ada provides approximately 1.4 times as much “functionality” (on average) of one LOC of Fortran

**Comparison of LOC/FP**

<table>
<thead>
<tr>
<th>Programming Language</th>
<th>LOC/FP (average)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assembly Language</td>
<td>320</td>
</tr>
<tr>
<td>C</td>
<td>128</td>
</tr>
<tr>
<td>Cobol</td>
<td>105</td>
</tr>
<tr>
<td>Fortran</td>
<td>105</td>
</tr>
<tr>
<td>Pascal</td>
<td>90</td>
</tr>
<tr>
<td>Ada</td>
<td>70</td>
</tr>
<tr>
<td>Object-oriented language</td>
<td>30</td>
</tr>
<tr>
<td>Fourth-generation language</td>
<td>20</td>
</tr>
<tr>
<td>Code generators</td>
<td>15</td>
</tr>
</tbody>
</table>

**References**

- “Software Engineering” by Ian Sommerville, Addison-Wesley, 2001