Software Process and Project Metrics

Software Engineering 5

Measurements

When you can measure what you are speaking about and can express it in numbers, you know something about it. But when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meager and unsatisfactory kind.

Lord Kelvin

If software development is to be viewed as an engineering discipline, it requires a measurement component that allows us to better understand, evaluate, predict and control the software process and product.

Victor Basili
University of Maryland
**Measurement & Metrics**

... collecting metrics is too hard ... it’s too time-consuming ... it’s too political ... it won’t prove anything ...

Anything that you need to quantify can be measured in some way that is superior to not measuring it at all..

Tom Gilb

---

### Why do we Measure?

- To characterize
- To evaluate
- To predict
- To improve
A Good Manager Measures

What do we use as a basis?
- size?
- function?

Measures, Metrics, and Indicators

Measure
Provides a quantitative indication of the extent, amount, dimensions, capacity, or size of some attribute of a product or process.

Metric
A quantitative measure of the degree to which a system, component, or process possesses a given attribute.

Indicator
A metric or combination of metrics that provide insight into the software process, a software project, or the product itself.
Software Measurement Classification

Software measurements can be classified in several ways:
- Objective/subjective
- Absolute/relative
- Explicit/derived
- Dynamic/static
- Predictive/explanatory

Process Metrics

- majorly focus on quality achieved as a consequence of a repeatable or managed process
- statistical SQA data
  - error categorization & analysis
- defect removal efficiency
  - propagation from phase to phase
- reuse data
**Project Metrics**

- Effort/time per SE task
- Errors uncovered per review hour
- Scheduled vs. actual milestone dates
- Changes (number) and their characteristics
- Distribution of effort on SE tasks

**Product Metrics**

- focus on the quality of deliverables
- measures of analysis model
- complexity of the design
  - internal algorithmic complexity
  - architectural complexity
  - data flow complexity
- code measures (e.g., Halstead)
- measures of process effectiveness
  - e.g., defect removal efficiency
Metrics Guidelines

- Use common sense and organizational sensitivity when interpreting metrics data.
- Provide regular feedback to the individuals and teams who have worked to collect measures and metrics.
- Don’t use metrics to appraise individuals.
- Work with practitioners and teams to set clear goals and metrics that will be used to achieve them.
- Never use metrics to threaten individuals or teams.
- Metrics data that indicate a problem area should not be considered “negative.” These data are merely an indicator for process improvement.
- Don’t obsess on a single metric to the exclusion of other important metrics.

Causes of Defects and Their Origin

<table>
<thead>
<tr>
<th>Specification/Requirements</th>
<th>Design</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic 20%</td>
<td>Specifications 25.5%</td>
<td>User Interface 11.7%</td>
</tr>
<tr>
<td>Data Handling 10.5%</td>
<td></td>
<td>Error Checking 10.9%</td>
</tr>
<tr>
<td>Standards 6.9%</td>
<td></td>
<td>Hardware Interface 7.7%</td>
</tr>
</tbody>
</table>

Relative Cost to Fix Defects
**Statistical Software Process Improvement**

**- Failure Analysis**

1. All errors and defects are categorized by origin (e.g. flaw in specification, flaw in logic, nonconformance to standards).
2. The cost to correct each error and defect is recorded.
3. The number of errors and defects in each category are counted and ordered in descending order.
4. The overall cost of errors and defects in each category is computed.
5. Resultant data are analyzed to uncover the categories that result in highest cost to the organization.
6. Plans are developed to modify the process with the intent of eliminating (or reducing the frequency of occurrence of) the class of errors and defects that is most costly.

**Problem Cause Analysis**

![Diagram of Problem Cause Analysis](image)
Normalization for Metrics

Normalized data are used to evaluate the process and the product (but never individual people)

size-oriented normalization—the line of code approach
function-oriented normalization—the function point approach

Typical Size-Oriented Metrics

- errors per KLOC (thousand lines of code)
- defects per KLOC
- $ per LOC
- page of documentation per KLOC
- errors / person-month
- LOC per person-month
- $ / page of documentation
Typical Function-Oriented Metrics

- errors per FP (thousand lines of code)
- defects per FP
- $ per FP
- pages of documentation per FP
- FP per person-month

The Controversy over Size-oriented Metrics

Size-oriented metrics are not universally accepted. The use of LOC as a key measure is the center of the conflict.

Proponents of the LOC measure claim:
- it is an artifact of all software engineering processes which can easily be counted
- many existing metrics exist which use LOC as an input
- a large body of literature and data exist which is predicated on LOC

Opponents of the LOC measure claim:
- that it is language dependent
- well designed short programs are penalized
- they do not work well with non-procedural languages
- their use in planning is difficult because the planner must estimate LOC before the design is completed
Function-Oriented Metrics

Function-oriented metrics are indirect measures of software which focus on functionality and utility. Functionality cannot be directly measured, so it must be derived indirectly using other direct measures.

The first function-oriented metrics was proposed by Albrecht who suggested a productivity measurement approach called the function point method. Function points (FPs) are derived using an empirical relationship from countable measures and assessments of software complexity.

Once calculated, FPs may be used in place of LOC as a measure of productivity, quality, cost, documentation, and other attributes.

Why Opt for FP Measures?

- independent of programming language
- uses readily countable characteristics of the "information domain" of the problem
- does not "penalize" inventive implementations that require fewer LOC than others
- makes it easier to accommodate reuse and the trend toward object-oriented approaches
**LOC to FP Ratio Estimates**

<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 languages</td>
<td>30</td>
</tr>
<tr>
<td>Fourth gen. languages (4GL)</td>
<td>20</td>
</tr>
<tr>
<td>Code generators</td>
<td>15</td>
</tr>
<tr>
<td>Spreadsheets</td>
<td>6</td>
</tr>
<tr>
<td>Graphical languages (icons)</td>
<td>4</td>
</tr>
</tbody>
</table>

**Computing Function Points**

1. **Analyze information domain of the application and develop counts**
2. **Weight each count by assessing complexity**
3. **Assess influence of global factors that affect the application**
4. **Establish count for input domain and system interfaces**
5. **Assign level of complexity or weight to each count**
6. **Grade significance of external factors, \( F_i \) such as reuse, concurrency, OS, ...**

\[
\text{function points} = \sum (\text{count} \times \text{weight}) \times C \\
\text{where:} \\
C = (0.65 + 0.01 \times N) \\
N = \sum F_i
\]
### Analyzing the Information Domain

<table>
<thead>
<tr>
<th>Measurement Parameter</th>
<th>Count</th>
<th>Weighting Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of user inputs</td>
<td>X 3</td>
<td>4, 6 =</td>
</tr>
<tr>
<td>Number of user outputs</td>
<td>X 4</td>
<td>5, 7 =</td>
</tr>
<tr>
<td>Number of user inquiries</td>
<td>X 3</td>
<td>4, 6 =</td>
</tr>
<tr>
<td>Number of files</td>
<td>X 7</td>
<td>10, 15 =</td>
</tr>
<tr>
<td>Number of ext. interfaces</td>
<td>X 5</td>
<td>7, 10 =</td>
</tr>
<tr>
<td>Complexity multiplier</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Function points</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Taking Complexity into Account

Factors are rated on a scale of 0 (not important) to 5 (very important):

- Data communications
- Distributed functions
- Heavily used configuration
- Transaction rate
- On-line data entry
- End user efficiency
- On-line update
- Complex processing
- Installation ease
- Operational ease
- Multiple sites
- Facilitate change
Measuring Quality

Correctness — the degree to which a program operates according to specification
Maintainability—the degree to which a program is amenable to change
Integrity—the degree to which a program is impervious to outside attack
Usability—the degree to which a program is easy to use

Defect Removal Efficiency

\[
\text{DRE} = \frac{\text{errors}}{\text{errors} + \text{defects}}
\]

where
errors = problems found before release
defects = problems found after release
Managing Variation

The mR Control Chart

Formulation Principles

- The objectives of measurement should be established before data collection begins;
- Each technical metric should be defined in an unambiguous manner;
- Metrics should be derived based on a theory that is valid for the domain of application (e.g., metrics for design should draw upon basic design concepts and principles and attempt to provide an indication of the presence of an attribute that is deemed desirable);
- Metrics should be tailored to best accommodate specific products and processes [BAS84]
Collection and Analysis Principles

- Whenever possible, data collection and analysis should be automated
- Valid statistical techniques should be applied to establish relationship between internal product attributes and external quality characteristics
- Interpretative guidelines and recommendations should be established for each metric

Attributes

simple and computable. It should be relatively easy to learn how to derive the metric, and its computation should not demand inordinate effort or time empirically and intuitively persuasive. The metric should satisfy the engineer’s intuitive notions about the product attribute under consideration consistent and objective. The metric should always yield results that are unambiguous.
consistent in its use of units and dimensions. The mathematical computation of the metric should use measures that do not lead to bizarre combinations of unit.
programming language independent. Metrics should be based on the analysis model, the design model, or the structure of the program itself.
an effective mechanism for quality feedback. That is, the metric should provide a software engineer with information that can lead to a higher quality end product
**Analysis Metrics**

- Function-based metrics: use the function point as a normalizing factor or as a measure of the “size” of the specification
- Bang metric: used to develop an indication of software “size” by measuring characteristics of the data, functional and behavioral models
- Specification metrics: used as an indication of quality by measuring number of requirements by type