Software Metrics

Any measurement which relates to a software system, process, or related documentation

Outline

• Properties of Metrics
• Analysis Metrics
• Design Metrics
• Implementation Metrics
• Documentation Metrics
Software metrics

• Any type of measurement which relates to a software system, process, or related documentation, e.g.
  – Lines of code in a program
  – The Fog index
  – Person-days required to develop a component

• Allow the software and the software process to be quantified

• Measures of the software process or product

• Should be captured automatically if possible

Metrics assumptions

• A software property can be measured

• The relationship exists between what we can measure and what we want to know

• This relationship has been formalized and validated

• It may be difficult to relate what can be measured to desirable quality attributes
Internal and external attributes

Properties of Metrics

- Simple and Computable
- Empirically and Intuitively Persuasive
- Consistent and Objective
- Programming Language Independent
Data accuracy

• Don’t collect unnecessary data. The questions to be answered should be decided in advance and the required data identified.
• Tell people why the data is being collected. It should not be part of personnel evaluation.
• Don’t rely on memory. Collect data when it is generated, not after a project has finished and, if possible, automatically.

Measurement analysis

• Not always obvious what data means. Analyzing collected data is very difficult.
• Professional statisticians should be consulted if available.
• Data analysis must take local circumstances into account.
Analysis Metrics

• Attempt to estimate costs early in the development process:
  – Cocomo
  – Function Points

Cocomo

• Effort (in person-months) as a function of lines-of-code where $b$ and $c$ are constants determined by historical data dependent on the project type:

  \[
  \text{Initial effort} = b \times KLOC^c
  \]
Cocomo Project Types

• **Organic** (b=2.4, c=1.05)
  - A relatively small team develops software in a known environment. Those involved generally have significant experience with projects of this type. These tend to be small projects.

• **Embedded** (b=3.0, c=1.12)
  - These projects involve the development where the intended environment poses significant constraints. The product is "embedded" in an environment which is inflexible.

• **Semidetached** (b=3.6, c=1.20)
  - The team may show a mixture of experience and inexperienced people. The project may be larger than that in the organic type.

Cocomo Cost Drivers

Factors used to adjust initial effort estimate:

<table>
<thead>
<tr>
<th>Cost Drivers</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>very low</td>
</tr>
<tr>
<td><strong>Product Attributes</strong></td>
<td></td>
</tr>
<tr>
<td>Reliability</td>
<td>.75</td>
</tr>
<tr>
<td>Data base</td>
<td>.94</td>
</tr>
<tr>
<td>Product Complexity</td>
<td>.70</td>
</tr>
<tr>
<td><strong>Computer Attributes</strong></td>
<td></td>
</tr>
<tr>
<td>Execution Time</td>
<td>.87</td>
</tr>
<tr>
<td>Main Storage</td>
<td>1.00</td>
</tr>
<tr>
<td>Virtual Machine Volatility</td>
<td>1.00</td>
</tr>
<tr>
<td>Computer Turnaround Time</td>
<td>.87</td>
</tr>
<tr>
<td><strong>Personnel Attributes</strong></td>
<td></td>
</tr>
<tr>
<td>Analyst Capabilities</td>
<td>1.46</td>
</tr>
<tr>
<td>Application Experience</td>
<td>1.29</td>
</tr>
<tr>
<td>Programmer Capability</td>
<td>1.42</td>
</tr>
<tr>
<td>Virtual Machine Experience</td>
<td>1.21</td>
</tr>
<tr>
<td>Programming Experience</td>
<td>1.14</td>
</tr>
<tr>
<td><strong>Project Attributes</strong></td>
<td></td>
</tr>
<tr>
<td>Use of modern prog. Tech.</td>
<td>1.24</td>
</tr>
<tr>
<td>Use of Software Tools</td>
<td>1.24</td>
</tr>
<tr>
<td>Development Schedule</td>
<td>1.23</td>
</tr>
</tbody>
</table>
Cocomo problems

- Empirical studies show high (e.g. 600%) estimation errors.
  - Breaking into subsystems may help
- How is initial size parameter obtained?
  - Why are cost driver multipliers given to three-digit accuracy?
- Does not address factors such as application domain, methodologies, CASE tools, code reuse, etc.
  - Probably requires calibration

---

Function/Feature Points (1)

Compute weighted function count, FC:

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Weighting Factors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input [I]</td>
<td>Use Input</td>
<td>Simple</td>
</tr>
<tr>
<td>Output [O]</td>
<td>Use Output</td>
<td>4</td>
</tr>
<tr>
<td>Inquiry [Q]</td>
<td>Use Inputs and Control</td>
<td>2</td>
</tr>
<tr>
<td>Logical</td>
<td>Internal Data maintained by the system</td>
<td>7</td>
</tr>
<tr>
<td>Interfaces [F]</td>
<td>Data input in another program</td>
<td>5</td>
</tr>
</tbody>
</table>
Function/Feature Points (2)

Compute *value adjustment factor*, VAF, as the sum of 14 characteristics on a six point scale (0 = no influence to 5 = strong influence).

- Data Communications
- Distributed Function
- Performance
- Heavily Used Configuration
- Transaction Rate
- On-line Data Entry
- End-user Efficiency
- On-line Update
- Complex Processing
- Re-usability
- Installation Ease
- Operational Ease
- Multiple Sites
- Facilitate Change

Function/Feature Points (3)

- **FP = FC * VAF**

- Counts and weights are subjective
  - Based on top level data flow diagrams
  - Studies show median difference between raters approximately 12%

- Studies show good correlation with FP’s
  - System size
  - Development effort

- But the studies are limited to MIS applications
  - No good studies on scientific, mathematical, communications, telecomm, real-time, etc.
Design Metrics

- Cohesion
- Coupling
- Complexity

Product Quality

- A quality metric should be a predictor of product quality.
- Design quality metrics concerned with measuring the cohesion, coupling, or complexity of a design are supposed to correlate with quality.
- The relationship between these metrics and quality as judged by a human may hold in some cases but it is not clear whether or not it is generally true.
Cohesion

- A measure of how strongly the elements within a module are related. The stronger the better.

Types of Cohesion

- Object
- Functional
- Sequential
- Communicational
- Procedural
- Temporal
- Logical
- Coincidental
Critique

- Difficult to apply in practice.
- Tedious to apply manually and impossible to automate.
- Informal

Coupling

- A measure of the degree of independence between modules. When there is little interaction between two modules, the modules are described as loosely coupled. When there is a high degree of interaction the modules are described as tightly coupled.
Types of Coupling

1. **Content Coupling**: One module refers to or changes the internals of another module.

2. **Common Coupling**: Two modules share the same global data areas.

3. **Control Coupling**: Data from one module is used to direct the order of instruction execution in another.

4. **Data Coupling**: Two modules communicate via a variable or array that is passed directly as a parameter between the two modules. The data is used in problem related processing, not for program control purposes.

Coupling metrics

- Associated with Yourdon's 'Structured Design'/Measures 'fan-in and fan-out' in a structure chart.

- High fan-in (number of calling functions) suggests high coupling because of module dependencies.

- High fan-out (number of calls) suggests high coupling because of control complexity.
Structural fan-in and fan-out

Informational fan-in/fan-out

• The approach based on the calls relationship is simplistic because it ignores data dependencies.

• Informational fan-in/fan-out takes into account:
  – Number of local data flows + number of global data structures updated.
  – Data-flow count subsumes calls relation. It includes updated procedure parameters and procedures called from within a module.

• Complexity = Length * (Fan-in * Fan-out)^2
  – Length is any measure of program size such as LOC.
Validation of Fan-in/fan-out

- Some studies with Unix found that informational fan-in/fan-out allowed complex and potentially faulty components to be identified.
- Some studies suggest that size and number of branches are as useful in predicting complexity than informational fan-in/fan-out.
- Fan-out on its own also seemed to be a good quality predictor.
- The whole area is still a research area rather than practically applicable.

CK Metrics for OO Designs

- Proposed by Chidamber and Kemerer
- Weighted Methods per Class (WMC)
  - A weighted sum of all the methods in a class.
  - How to weight the methods is a matter of debate. Some authors have used method length, cyclomatic complexity, or weighted all methods equally.
- Coupling Between Object classes (CBO)
  - A count of the number of other classes to which a given class is coupled. A class is coupled to another if it sends messages to objects or invokes constructors of the other class.
  - Inherited couplings are included in the count.
CK Metrics (2)

- Depth of Inheritance Tree (DIT)
  - Length of the longest path from a given class to the root class of the inheritance hierarchy.

- Number of Children (NOC)
  - The number of immediate subclasses.

- Response for a Class (RFC)
  - The count of the maximum number of methods that can be potentially invoked in response to a single message received by an object of a particular class.

CK Metrics (3)

- Lack of Cohesion of Methods (LCOM)
  - A count of the number of method-pairs whose similarity is zero minus the count of method pairs whose similarity is not zero.
  - Similarity refers to the number of instance variables by the methods.
CK Metrics Validation

- As of 2004, there have been about 10 empirical studies on the CK Metrics.
- Most studies have found correlations between one or more of the CK metrics and defects or maintenance cost.
- Studies are inconsistent as to which of the metrics are useful.
- One study found that size overwhelmed the effect of all metrics on defects.
- CK metrics do not take into account certain other factors (e.g. encapsulation, polymorphism) that affect complexity.

Implementation Metrics

- Design metrics are also applicable to programs.
- Other metrics include:
  - Length. The size of the program source code.
  - Cyclomatic complexity. The complexity of program control.
  - Length of identifiers.
  - Depth of conditional nesting.
- Anomalous metric values suggest a component may contain an above average number of defects or may be difficult to understand.
Lines of Code

• Lines of code (LOC or KLOC) is typically correlated with effort. Boehm, Walston-Felix, and Halstead all indicate Effort as a function of lines of code.

• Support
  – Easy to Determine
  – Easy to Automate
  – Easy to Enforce

LOC - Objections

• Some programmers write more verbose programs than others.

• Difficult to compare programs written in different languages.

• Some lines are more complex than others.

• What constitutes a line of code?
LOC - What is a line of code?

Cyclomatic Complexity

- Based on determining how complicated the control flow a program (procedure or function) is.
- The control flow is represented as directed graph.
- \( CC = \text{Number(edges)} - \text{Number(nodes)} + 1 \)
Cyclomatic Complexity (2)

Sequence: CC = 0

If-else: CC = 1

While loop: CC = 1

Program metrics utility

- Lines of code is simple, but experiments indicate it is a good predictor of problems.
- Cyclomatic complexity is a measure of control structure complexity, but has two drawbacks:
  - It is inaccurate for data-driven programs as it is only concerned with control constructs.
  - It places the same weight on nested and non-nested loops. Deeply nested structures, however, are usually harder to understand.
Reliability metrics

• Probability of failure-free operation for a specified time in a specified environment for a given purpose.

• This means quite different things depending on the system and the users of that system.

• Informally, reliability is a measure of how well system users think it provides the services they require.

Software reliability

• Cannot be defined objectively
  – Reliability measurements which are quoted out of context are not meaningful

• Requires operational profile for its definition
  – The operational profile defines the expected pattern of software usage

• Must consider fault consequences
  – Not all faults are equally serious. System is perceived as more unreliable if there are more serious faults
Failures and faults

- A failure corresponds to unexpected run-time behavior observed by a user of the software.

- A fault is a static software characteristic which causes a failure to occur.

- Faults need not necessarily cause failures. They only do so if the faulty part of the software is used.

- If a user does not notice a failure, is it a failure? Remember most users don’t know the software specification.

Reliability metrics

- Probability of failure on demand
  - This is a measure of the likelihood that the system will fail when a service request is made.
  - POFOD = 0.001 means 1 out of 1000 service requests result in failure.
  - Relevant for safety-critical or non-stop systems.

- Rate of failure occurrence (ROCOF)
  - ROCOF of 0.02 means 2 failures are likely in each 100 operational time units.
  - Relevant for operating systems, transaction processing systems.
Reliability metrics

• Mean time between failure
  – Measure of the time between observed failures.
  – MTBF is the reciprocal of ROCOF if the system is not being changed during operation.

• Availability
  – Measure of how likely the system is available for use.
    Takes repair/restart time into account.
  – Availability of 0.998 means software is available for 998 out of 1000 time units.
  – Relevant for continuously running systems e.g. telephone switching systems.

Time units

• Time units in reliability measurement must be carefully selected. Not the same for all systems.

• Raw execution time (for non-stop systems).

• Calendar time (for systems which have a regular usage pattern e.g. systems which are always run once per day).

• Number of transactions (for systems which are used on demand).
Failure consequences

• Reliability measurements do NOT take the consequences of failure into account

• Transient faults may have no real consequences but other faults may cause data loss or corruption and loss of system service

• May be necessary to identify different failure classes and use different measurements for each of these

Documentation quality metrics

• Readability of documentation is important.

• Gunnings Fog index is a simple measure of readability.
  – Based on length of sentences and number of syllables in a word

• However, this can be misleading when applied to technical documentation.
Key points

• Metrics gather information about both process and product.
• Quality metrics should be used to identify potentially problematical components.
• Lack of commonality across software process between organizations makes universal metrics difficult to develop.
• Metrics probably need to be calibrated to particular application domains and development organizations.

Key points

• Metrics still have a limited value and are not widely collected.
• Relationships between what we can measure and what we want to know are not well-understood.