Introduction to Software Metrics

Outline

• Today we begin looking at measurement of software quality using software metrics
• We’ll look at:
  • What are software quality metrics?
  • Some basic measurement theory
  • A framework for software measurement
• Well also focus on several examples of product metrics:
  • External product metrics – defect metrics
  • Internal product metrics – size metrics, complexity metrics
Software Quality Metrics

Applying Measurement to Software

- Software metrics are measurable properties of software systems, their development and use
- Includes wide range of different measures, of:
  - properties of the software product itself
  - the process of producing and maintaining it
  - its source code, design, tests, etc.
- Examples are:
  - number of failures
  - time to build
  - number of lines of code
  - number of failures per 1,000 lines of code
  - number of lines of code per programmer per month
  - number of decisions per 1,000 lines of code
What are Metrics Good for?

**Reliability and Quality Control**
- Metrics help us to predict and control the quality of our software
- **Example**: By measuring relative effectiveness of defect detection and removal of various testing or inspection methods, we can choose best one for our software products

**Cost Estimation and Productivity Improvement**
- Metrics help us predict effort to produce or maintain our software, and to improve our scheduling and productivity
- **Example**: By measuring code production using different languages or tools, we can choose those that give the best results

**Quality Improvement**
- Metrics help us to improve code quality and maintainability
- **Example**: By measuring complexity of our program code, we can identify sections of code most likely to fail or difficult to maintain
Kinds of Metrics

Three Basic Kinds

- There are three kinds of software quality metrics: product metrics, process metrics and project metrics

Product Metrics

- Product metrics are those that describe the internal and external characteristics of the product itself
- Examples: size, complexity, features, performance, reliability, quality level
- Most common software metrics are of this kind
Kinds of Metrics

Process Metrics
• Process metrics measure the process of software development and maintenance, in order to improve it
• Examples: effectiveness of defect removal during development, pattern of defect arrival during testing, response time for fix

Project Metrics
• Project metrics are those that describe the project characteristics
• Examples: number of developers, development cost, schedule, productivity
Measurement Basics

If You Want to Know, Measure...

- "When you can measure what you are speaking about, and 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 kind."

  - William Thomson (Lord Kelvin), Physicist

...But Make Sure You Know What You Are Measuring

- "In truth, a good case could be made that if your knowledge is meager and unsatisfactory, the last thing in the world you should do is make measurements. The chance is negligible that you will measure the right things accidentally."

  - George A. Miller, Psychologist
Measurement Basics

Definition of Measurement
• *Measurement* is the process of empirical *objective* assignment of numbers to *entities*, in order to characterize an *attribute*

What Does That Mean?
• An *entity* is an object or event, such as a source program
• An *attribute* is a feature or property of an entity, such as the size of the program
• *Objective* means measurement must be based on a well-defined *rule* whose results are *repeatable*, such as counting the number of source lines in the program

In Other Words ...
• Each *entity* is given a *number*, which tells you about its *attribute*
• *Example*: Each source program has a source line count, which tells you about its size
# Measurement Basics

## Example Measurements

<table>
<thead>
<tr>
<th>ENTITY</th>
<th>ATTRIBUTE</th>
<th>MEASURE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Person</td>
<td>Age</td>
<td>Years at last birthday</td>
</tr>
<tr>
<td>Person</td>
<td>Age</td>
<td>Months since birth</td>
</tr>
<tr>
<td>Source code</td>
<td>Length</td>
<td># Lines of Code (LOC)</td>
</tr>
<tr>
<td>Source code</td>
<td>Length</td>
<td># Executable statements</td>
</tr>
<tr>
<td>Testing process</td>
<td>duration</td>
<td>Time in hours from start to finish</td>
</tr>
<tr>
<td>Tester</td>
<td>efficiency</td>
<td>Number of faults found per KLOC</td>
</tr>
<tr>
<td>Testing process</td>
<td>fault</td>
<td>Number of faults found per KLOC</td>
</tr>
<tr>
<td>Source code</td>
<td>frequency</td>
<td>Number of faults found per KLOC</td>
</tr>
<tr>
<td>Operating system</td>
<td>reliability</td>
<td>Mean Time to failure rate of occurrence of failures</td>
</tr>
</tbody>
</table>
Measurement Basics

Common Mistakes in Software Measurement

- It’s easy to make mistakes in choosing what or how to measure software characteristics.
- To avoid mistakes, stick to the definition of measurement.

(1) You must specify both an entity and an attribute, not just one or the other.
   - Example: Don’t just say you are measuring a program, say what property of the program you are measuring.
   - Example: Don’t just say you are measuring the size of the software, say what artifact of the software you are measuring the size of (e.g., source programs).

(2) You must define the entity precisely.
   - Example: Don’t just say program, say program source code.
Measurement Basics

Common Mistakes in Software Measurement (continued…)

(3) You must have a good intuitive understanding of the attribute before you propose a measure for it

• **Example**: We have good evidence that *size* is related to number of source lines

• It is a mistake to propose a *measure* if there is no clear consensus on what *attribute* it is characterizing
  • **Example**: Number of defects per KLOC (1000 lines of code) - characterizes quality of *code*, or quality of *testing*?

• It is a mistake to redefine an *attribute* to fit an existing *measure*
  • **Example**: If we’ve measured *#defects found this month*, don’t mistake that as an indicator of *code* quality
Kinds and Uses of Software Measurement

Kinds of Measurement
• Two distinct kinds of measurement,
  1. direct measurement, and
  2. indirect measurement

Uses of Measurement
• Two distinct uses for measurement,
  1. assessment (the way things are now), and
  2. prediction (the way things are likely to be in future)
• Measurement for prediction requires a prediction system
Direct Measurement

Some Direct Software Measures

- **Direct** measures are numbers that can be derived directly from the entity without other information

- **Examples:**
  - *Length* of source code
    (measured by number of lines)
  - *Duration* of testing process
    (measured in elapsed hours)
  - *Number of defects discovered* during the testing process
    (measured by counting defects)
  - *Effort* of a programmer on a project
    (measured by person-months worked)
Indirect Measurement

Some Indirect Software Measures

- **Indirect** measures are numbers that are derived by combining two or more direct measures to characterize an attribute.

- **Examples:**
  
  - *Programmer productivity* = \( \frac{\text{Lines of code produced}}{\text{Person-months of effort}} \)
  
  - *Program defect density* = \( \frac{\text{Number of defects}}{\text{Length of source code}} \)
  
  - *Requirements stability* = \( \frac{\text{Original number of requirements}}{\text{Total number of requirements}} \)
  
  - *Test effectiveness ratio* = \( \frac{\text{Number of items covered}}{\text{Total number of items}} \)
Predictive Measurement

Prediction Systems

- Measurement for prediction requires a prediction system
- A prediction system consists of:
  - A mathematical model
    Example: $E = a \cdot S^b$, where $E$ is the effort (to be predicted), $S$ is the estimated size in lines of code, and $a$ and $b$ are constant parameters
  - A procedure for determining the model parameters
    Example: Analyze past project data to determine $a$ and $b$
  - A procedure for interpreting the results
    Example: Use Bayesian probability analysis to determine the likelihood that our prediction is accurate within 10%
A Framework for Software Measurement

Products, Processes and Resources

Process
• A software-related activity or event (e.g., designing, coding, testing,...)

Product
• An item that results from a process (e.g., test plans, source code, design and specification documents, inspection reports, ...) 

Resource
• An item that is input to a process (e.g., people, hardware, software, ...)
A Framework for Software Measurement

**Internal and External Attributes**

- Let $X$ represent any product, process or resource
- The *external* attributes of $X$ are those attributes which can only be measured by how $X$ interacts with its environment
  - Example (product): Mean time to failure of a program
  - Example (product): Maintainability of source code
- The *internal* attributes of $X$ are those attributes which can be measured purely in terms of $X$ itself
  - Example (product): Length of source code
  - Example (product): Complexity of source code
- These are related to direct and indirect measurements (but are not the same - e.g., because *indirect* measurements can be made involving only *internal* attributes)
A Framework for Software Measurement

Applying the Framework

<table>
<thead>
<tr>
<th>ENTITIES</th>
<th>ATTRIBUTES</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Internal</td>
</tr>
<tr>
<td><strong>PRODUCTS</strong></td>
<td></td>
</tr>
<tr>
<td>Specification</td>
<td>Length, functionality</td>
</tr>
<tr>
<td>Source Code</td>
<td>modularity, structuredness,</td>
</tr>
<tr>
<td>Source</td>
<td>reuse</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>PROCESSES</strong></td>
<td>time, effort, #spec faults</td>
</tr>
<tr>
<td>Design</td>
<td>found</td>
</tr>
<tr>
<td>Test</td>
<td>time, effort, #failures</td>
</tr>
<tr>
<td></td>
<td>observed</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>RESOURCES</strong></td>
<td>age, price, CMM level</td>
</tr>
<tr>
<td>People</td>
<td>price, size</td>
</tr>
<tr>
<td>Tools</td>
<td></td>
</tr>
</tbody>
</table>

Source: Fenton, Agena Corp. 2000
Product Metrics
External Product Metrics

Measures of the Software in its Environment

- **External** metrics are those we can apply only by observing the software product in its environment (e.g., by running it)
- Includes many measures, but particularly:
  - failure rate (\# of failures/unit of time)
  - availability rate (% of time system is "up")
  - defect rate (number of defects/size of code)
  - ...
Definition of Reliability

- Reliability is the probability that a system will execute without failure in a given environment over a given period of time
- **Implications:**
  - no single number for a given program - depends on how the program is used (its environment)
  - use probability to model our uncertainty
  - time dependent
Reliability

Definition of Reliability
• Reliability is the probability that a system will execute without failure in a given environment over a given period of time
• Implications:
  • no single number for a given program - depends on how the program is used (its environment)
  • use probability to model our uncertainty
  • time dependent

Definition of Failure
• Formal view:
  Any deviation from specified behaviour
• Engineering view:
  Any deviation from required, specified or expected behaviour
  (by environment) (by user)
Errors, Faults and Failures

Definitions

• A (human) error is a mistake or oversight on the part of a designer or programmer, which may cause ...
• A fault, which is a mistake in the software, which in turn may cause ...
• A failure when the program is run in certain situations

Defects

• A defect is usually defined as a fault or a failure:

    Defects = Faults + Failures

    (or sometimes just Faults or just Failures)
Defect Density Metric

**Defect Density**

- **Defect density** is a standard reliability metric:

\[
\text{Defect Density} = \frac{\text{Number of defect found}}{\text{System Size}}
\]

- Size is normally measured in **KLOC** (1000’s of lines of code), so units of defect density are defects found per 1000 lines.
- Widely used as an indicator of software **quality**
However ...

• Unfortunately, faults are not a good predictor of failures, and vice versa (Adams 1984)

• 35% of faults cause only about 1% of failures, and 35% of failures are caused by only about 2% of faults

• This finding makes historical defect density look like not such a good predictor of quality
(ASIDE) A Process Metric Using Defects

Effectiveness of Defect Detection & Defect Removal

- Defect statistics can also be used for evaluating and improving software process

<table>
<thead>
<tr>
<th>Testing Type</th>
<th>Defects found per hour</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular use</td>
<td>0.21</td>
</tr>
<tr>
<td>Black box</td>
<td>0.282</td>
</tr>
<tr>
<td>White box</td>
<td>0.322</td>
</tr>
<tr>
<td>Reading/Inspections</td>
<td>1.057</td>
</tr>
</tbody>
</table>

- Grady (1992) used defect metrics to show effectiveness of inspection vs. testing

Source: Fenton, Agena Corp. 2000
Internal Product Metrics

Measures of the Product Itself

• The vast majority of metrics in practice are internal product metrics, measures of the software code, design or functionality itself, independent of its environment

• The U.S. military lists literally hundreds of measures of code alone

• These measures are easy to make and easy to automate, but it’s not always clear which attributes of the program they characterize (if any)
Code Metrics

Software Size

- The simplest and most enduring product metric is the size of the product, measured using a count of the number of lines of source code (LOC), most often quoted in 1000’s (KLOC).
- It is used in a number of other indirect measures, such as:
  - productivity (LOC / effort)
  - effort / cost estimation (Effort = f(LOC))
  - quality assessment / estimation (defects / LOC)
- Many similar measures are also used:
  - KDSI (1000’s of delivered source instructions)
  - NLOC (non-comment lines of code)
  - number of characters of source or bytes of object code
Using Code Metrics

Example: Toshiba Productivity

Instructions per programmer month

Source: Fenton, Agena Corp. 2000
© J.S. Bradbury, J.R. Cordy
Problems with LOC Measures

What Attribute is Measured?

• LOC really measures length of program (a physical characteristic), not size (a logical characteristic)
• Mistakenly used as a surrogate for measures of what we’re really interested in - effort, complexity, functionality
• Does not take into account redundancy, reuse (e.g., XXXX Bank - 500 MLOC, only about 100 MLOC unique)
• Cannot be compared across different programming languages
• Can only be measured at end of development cycle
Better Size Measures

Fundamental Size Attributes of Software

- **Length** - the physical size of the software (LOC will do as measure)
- **Functionality** - the capabilities provided to the user by the software (how big / rich is the set of functions provided)
- **Complexity** - how complex is this software?
  - **Problem complexity** - measures the complexity of the underlying problem
  - **Algorithmic complexity** - measures the complexity / efficiency of the solution implemented by the software
  - **Structural complexity** - measures the structure of the program used to implement the algorithm (includes control structure, modular structure, data flow structure and architectural structure)
  - **Cognitive complexity** - measures the effort to understand the software
Code Complexity Metrics

Better Measures of Source Code

• Realization that we need something better approaching cognitive complexity led to work on complexity metrics for code

• Early explorations measured characteristics such as:
  • number / density of decision (if) statements
  • number / depth of blocks / loops
  • number / average length of methods / classes
  • and many more…

• Best known and accepted source code complexity measures are
  • Halstead’s “Software Science” metrics
  • McCabe’s “Cyclomatic Complexity” and “Data Complexity” metrics
Halstead’s “Software Science” Metric

Operators and Operands

• Program source code considered as a sequence of tokens, each of which is either an operator or an operand

\[ n_1 = \text{number of unique (different) operators} \]
\[ n_2 = \text{number of unique (different) operands} \]
\[ N_1 = \text{total number of operator uses} \]
\[ N_2 = \text{total number of operand uses} \]

Length of program \[ N = N_1 + N_1 \]
Vocabulary of program \[ n = n_1 + n_2 \]
Halstead’s “Software Science” Metric

The Software Science Predictive Theory

- Using \( n_1, n_2, N_1 \) and \( N_2 \) as a basis, Halstead formulated a theory of software complexity and effort

  - **Theory 1**: An estimate of \( N \) is \( N = n_1 \log n_1 + n_2 \log n_2 \)
  
  - **Theory 2**: Effort \( E \) required to create \( P \) is
    \[
    E = \frac{n_1 N_2 N \log N}{2 n_2}
    \]
  
  - **Theory 3**: Time \( T \) required to program \( P \) is
    \[
    T = \frac{E}{18} \text{ seconds}
    \]

- Where: \( n_1 = \) unique operators, \( N_1 = \) total operators, \( n_2 = \) unique operands, \( N_2 = \) total operands
McCabe’s “Cyclomatic Complexity: Metric

**Flow Graphs Again**
- If the control flow graph $G$ of program $P$ has $e$ edges and $n$ nodes, then the cyclomatic complexity $v$ of $P$ is
  $$v(P) = e - n + 2$$
- $v(P)$ is the number of linearly independent paths in $G$
- **Example**
  
  $e = 16 \quad n = 13$
  $$v(P) = 16 - 13 + 2 = 5$$
- More simply, if $d$ is the number of decision nodes in $G$ then
  $$v(P) = d + 1$$
- McCabe proposed that for each module $P$
  $$v(P) < 10$$
Other Flowgraph Metrics

Flowgraph Complexity of Software

- McCabe is just one of many flowgraph-based complexity metrics, all with the advantage that they are independent of language.
- Others measure things like:
  - maximum path length
  - number / interaction of cycles
  - maximum number of alternative paths (width or “fan out”)
  - lots of others
- Can all be automatically computed once flowgraph is known (and it can be automatically computed too).
- Flowgraph decomposition, which partitions flowgraphs into independent one-node-in one-node-out subgraphs, provides a rigorous general theory of structured programming.
- And it can also be automatically computed.
Introduction to Software Metrics

Summary

- Measurement is about characterizing the **attributes** of entities
  - can be **direct** or **indirect**
  - can be for either **assessment** or **prediction**
- Framework for software measurement is based on:
  - classifying entities as **products**, **processes** and **resources**
  - classifying attributes as **internal** and **external**
  - determining whether we want **assessment** or **prediction**
- Product metrics include both **external** metrics (e.g., defect) and **internal** metrics (e.g., size, complexity)
Metrics II - Process Metrics

Outline

• Today’s topic is process metrics, the measurements we can make related to the software development and maintenance process

• We’ll look at:
  • Predictive process metrics - effort and cost
  • Specification metrics