Software Measurement

Mark Micallef
mmica01@um.edu.mt
Brief Course Overview

- Introduction to Measurement Theory
- Measurement as applied Software
- Examples of Various Metrics, Measures and Indicators
Introduction to Measurement Theory
What is measurement?

Measurement is the process by which numbers or symbols are assigned to attributes of entities in the world according to clearly defined rules.
The importance of Measurement

Can software be measured?

Is it software measurement useful?

How do you measure software?
The importance of Measurement

- Measurement is **crucial** to the progress of all sciences, *even Computer Science*

- Scientific progress is made through
  - Observations and generalisations…
  - …based on data and measurements
  - Derivation of theories and…
  - …confirmation or refutation of these theories

- Measurement turns an art into a science
Uses of Measurement

- Measurement helps us to understand
  - Makes the current activity visible
  - Measures establish guidelines
- Measurement allows us to control
  - Predict outcomes and change processes
- Measurement encourages us to improve
  - When we hold our product up to a measuring stick, we can establish quality targets and aim to improve
Some propositions

- Developers who drink coffee in the morning produce better code than those who do drink orange juice.
- The more you test the system, the more reliable it will be in the field.
- If you add more people to a project, it will be completed faster.
Abstraction Hierarchy

Abstract World

Proposition

Theory

Hypothesis

Empirical World

Data Analysis

An assumption or set of assumptions which are taken to be true.

An proposed explanation of a phenomenon.

Take measurements, carry out studies, look at past data, etc.

Proof or refute the hypothesis.
Example: Proving a theory

Adding more developers to a late project will only make it later.

If a project misses one or more milestone deadlines, it is considered to be late.

The greater the difference between the current time and the missed milestone, the later the project is said to be.

Project $p$ has $n$ developers working on it
Project $p$ missed a deadline be $x$ days ($x > 0$)
If we add $m$ developers ($m > 0$), then when the milestone is reached, the project would be $y$ days late ($y > x$)

Carry out studies, analyse data from past projects,…Verify or disprove the theory
Definitions (1/2)

- **Theory** - A supposition which is supported by experience, observations and empirical data.

- **Proposition** – A claim or series of claims which are assumed to be true.

- **Hypothesis** – A proposed explanation for a phenomenon. Must be *testable* and based on previous observations or scientific principles.
**Definitions** (2/2)

- **Entities** – Objects in the real world. May be animate, inanimate or even events.

- **Attributes** – Characteristics / features / properties of an entity

**Example**

*Entity:* Program

*Attributes*

- Time to Develop
- Lines of code
- Number of Defects
Levels of Measurement

Various scales of measurements exist:

- Nominal Scale
- Ordinal Scale
- Interval Scale
- Ratio Scale
Example: A religion nominal scale

Joe    Michelle
Rachel  Christine
Michael James
Clyde   Wendy
The Nominal Scale (2/2)

- The most simple measurement scale
- Involves sorting elements into categories with regards to a certain attribute
- There is no form of ranking
- Categories must be:
  - Jointly exhaustive
  - Mutually exclusive
Example: *A degree-classification ordinal scale*

- 1st Class: Joe, Michelle
- 2nd Class: Rachel, Christine, Michael, James, Clyde, Wendy
- 3rd Class: Failed

---

The Ordinal Scale (1/2)
The Ordinal Scale (2/2)

- Elements classified into categories
- Categories are ranked
- Categories are transitive $A > B \& B > C \Rightarrow A > C$
- Elements in one category can be said to be better (or worse) than elements in another category
- Elements in the same category are not rankable in any way
- As with nominal scale, categories must be:
  - Jointly exhaustive
  - Mutually exclusive
Interval Scale

- Indicates exact differences between measurement points
- Addition and subtraction can be applied
- Multiplication and Division **CANNOT** be applied
- We can say that product D has 8 more crashes per month but we cannot say that it has 3 times as more crashes

<table>
<thead>
<tr>
<th>CPU A</th>
<th>CPU B</th>
<th>CPU C</th>
<th>Product D</th>
</tr>
</thead>
<tbody>
<tr>
<td>0°C</td>
<td>30°C</td>
<td>60°C</td>
<td>120°C</td>
</tr>
<tr>
<td>86°F</td>
<td>140°F</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Temperature of Different CPUs
Ratio Scale

- The highest level of measurement available
- When an absolute zero point can be located on an interval scale, it becomes a ratio scale
- Multiplication and division can be applied (product D crashes 4 times as much per month than product B)
- For all practical purposes almost all interval measurement scales are also ratio scales
Measurement Scales Hierarchy

- Scales are hierarchical
- Each higher-level scale possesses all the properties of the lower ones
- A higher-level of measurement can be reduced to a lower one but not vice-versa

Ratio

Interval

Ordinal

Nominal

Most Powerful Analysis Possible

Least Powerful Analysis Possible
Measures, Metrics and Indicators

- **Measure** – An appraisal or ascertainment by comparing to a standard. E.g. Joe’s body temperature is 99° Fahrenheit

- **Metric** – A quantitative measure of the degree to which an element (e.g. software system) given attribute.
  - E.g. 2 errors were discovered by customers in 18 months (more meaningful than saying that 2 errors were found)

- **Indicator** – A device, variable or metric can indicate whether a particular state or goal has been achieved. Usually used to draw someone’s attention to something.
  - E.g. A half-mast flag indicates that someone has died
Example of a Measure
Example of a Metric

- **X-axis**: Months since release
- **Y-axis**: Bugs

Graph shows an increasing trend in bugs over time since release.
Example of a Indicator

Indicator of maximum safe temperature

Time (Hours)
Some basic measures (1/2)

- **Ratio**
  - E.g. The ratio of testers to developers in our company is 1:5

- **Proportion**
  - Similar to ratio but the numerator is part of the denominator as well
  - E.g. \( \frac{\text{Number of satisfied customers}}{\text{Total number of customers}} \)
Some basic measures (2/2)

- **Percentage**
  - A proportion or ration express in terms of per hundred units
  - E.g. 75% of our customers are satisfied with our product

- **Rate**
  - Ratios, proportions and percentages are static measures
  - Rate provides a dynamic view of a system
  - Rate shows how one variable changes in relation to another (one of the variables is usually time)
  - E.g. Lines of Code per day, Bugs per Month, etc
Reliability and Validity of Measurements

- **Reliability** – Refers to the consistency of a number of measurements taken using the same measurement method.

- **Validity** – Refers to whether the measurement or metric really measures what we intend it to measure.
Reliability and Validity of Measurements

- Reliable but not valid
- Valid but not reliable
- Reliable and Valid
Measuring Software
What makes quality software?

- Cheap?
- Reliable?
- Testable?
- Secure?
- Maintainable?
- …
What makes quality software?

- There is not clear-cut answer
- It depends on:
  - Stakeholders
  - Type of system
  - Type of users
  - ...
- Quality is a multifaceted concept

Different ideas about a quality car
Different Quality Scenarios

- Online banking system
  - Security
  - Correctness
  - Reliability

- Air Traffic Control System
  - Robustness
  - Real Time Responses

- Educational Game for Children
  - Userfriendliness
The 3 Ps of Software Measurement

With regards to software, we can measure:

- **Product**
- **Process**
- **People**
Measuring the Product

- Product refers to the actual software system, documentation and other deliverables
- We examine the product and measure a number of aspects:
  - Size
  - Functionality offered
  - Cost
  - Various Quality Attributes
Measuring the Process

- Involves analysis of the way a product is developed
- What lifecycle do we use?
- What deliverables are produced?
- How are they analysed?
- How can the process help to produce products faster?
- How can the process help to produce better products?
Measuring the People

- Involves analysis of the people developing a product
- How fast do they work?
- How much bugs do they produce?
- How many sick-days do they take?
- Very controversial. People do not like being turned into numbers.
The Measuring Process

Measurement Programme

Non-intrusive Data Collection

Products

Processes

People

Results, Trends, Reports, etc

Modifications
Collecting Software Engineering Data

- **Challenge**: Make sure that collected data can provide useful information for project, process and quality management without being a burden on the development team.
- Try to be as unintrusive as possible
- Try to make data collection automatic
- Can expensive
  - Sometimes difficult to convince management
A possible collection methodology:
1. Establish the goal of data collection
2. Develop a list of questions of interest
3. Establish data categories
4. Design and test data collection forms/programs
5. Collect and validate data
6. Analyse data
Examples of Metrics Programmes

Motorola

7 Goals
- Improve Project Planning
- Increase defect containment
- Increase software reliability
- Decrease defect density
- Improve customer service
- Reduce the cost of non-conformance
- Increase software productivity

Various Measurement Areas
- Delivered defects, process effectiveness, software reliability, adherence to schedule, time that problems remain open, and more…
Examples of Metrics Programmes

**IBM**
- IBM have a Software Measurement Council
- A set of metrics called 5-Up are defined and deal with:
  - Customer Satisfaction
  - Postrelease Defect Rates
  - Customer problem calls
  - Fix response time
  - Number of defective fixes
Hewlett-Packard

- Heavily influenced by defect metrics
  - Average fixed defects/working day
  - Average engineering hours / fixed defect
  - Average reported defects/working day
  - Defects / testing time
  - ...
Product Metrics
What can we measure about a product?

- Size metrics
- Defects-based metrics
- Cost-metrics
- Time metrics
- Quality Attribute metrics
Size Metrics

- Knowing the size of a system was important for comparing different systems together
- Software measured in lines of code (LOC)
- As systems grew larger KLOC (thousands of lines of code) was also used
The problems with LOC

Same system developed with different programming languages will give different LOC readings

- FoxPro 2 KLOC
- Pascal 5 KLOC
- Assembly 15 KLOC
The problems with LOC

- Same system developed by different developers using the same language will give different LOC readings

  Video Rental System

  - Developer A: 2 KLOC
  - Developer B: 1.2 KLOC
  - Developer C: 2.5 KLOC
The problems with LOC

- To calculate LOC you have to wait until the system is implemented.
- This is not adequate when management requires prediction of cost and effort.
- A different approach is sometimes necessary...
Function Points

- Instead of measuring size, function points measure the *functionality* offered by a system.
- Invented by Albrecht at IBM in 1979
- Still use today: [http://www.ifpug.org](http://www.ifpug.org)
Overview of Function Points

- Function Points gauge the functionality offered by a system.
- A *Function* can be defined as a collection of executable statements that performs a certain task.
- Function points can be calculated before a system is developed.
- They are language and developer independent.
Overview of Function Points

- A function point count is calculated as a weighted total of five major components that comprise an application...
  - External Inputs
  - External Outputs (e.g. reports)
  - Logical Internal Files
  - External Interface Files – *files accessed by the application but not maintained by it*
  - External Inquiries – *types of online inquiries supported*
The simplest way to calculate a function point count is calculated as follows:

\[(\text{No. of external inputs } \times 4) + (\text{No. of external outputs } \times 5) + (\text{No. of logical internal files } \times 10) + (\text{No. of external interface files } \times 7) + (\text{No. of external enquiries } \times 4)\]
Consider the following system specs:

Develop a system which allows customers to report bugs in a product. These reports will be stored in a file and developers will receive a daily report with new bugs which they need to solve. Customers will also receive a daily status report for bugs which they submitted. Management can query the system for a summary info of particular months.

1 External Inputs
1 Logical Internal Files
2 External Outputs
1 External Enquiries
Function Points Example

External Inputs: 1
External Outputs: 2
Logical Internal Files: 1
External Interface Files: 0
External Enquiries: 1

Total Functionality is \((1 \times 4) + (2 \times 5) + (1 \times 10) + (0 \times 7) + (1 \times 4) = 28\)
Function Point Extensions

- The original function points were sufficient but various people extended them to make them more expressive for particular domains.

Examples

- General System Characteristics (GSC) Extension
- 3D Function Points for real time systems
- Object Points
- Feature Points
The **GSC** Function Points Extension (1/3)

- **Reasoning:** Original Function Points do not address certain functionality which systems can offer
  - E.g. Distributed functionality, performance optimisation, etc
  - The GSC extension involves answering 14 questions about the system and modifying the original function point count accordingly
The GSC Function Points Extension (2/3)

1. Data communications
2. Distributed Functions
3. Performance
4. Heavily used configuration
5. Transaction rate
6. Online Data Entry
7. End-user Efficiency
8. On-line update
9. Complex Processing
10. Reusability
11. Installation ease
12. Operational Ease
13. Multiple sites
14. Facilitation of Change
The analyst/software engineer assigns a value between 0 and 5 to each question

- \( 0 = \text{not applicable} \) and \( 5 = \text{essential} \)

The Value-Adjustment Factor (VAF) is then calculated as:

\[
VAF = 0.65 + 0.01 \sum_{i=1}^{14} Ci
\]

You then adjust the original function point count as follows:

\[
FP = FC \times VAF
\]
Consider the bug-reporting system for which we already looked at and suppose the analyst involved answers the GSC questions as follows…

<table>
<thead>
<tr>
<th>Question</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Data communications</td>
<td>5</td>
</tr>
<tr>
<td>2. Distributed Functions</td>
<td>0</td>
</tr>
<tr>
<td>3. Performance</td>
<td>1</td>
</tr>
<tr>
<td>4. Heavily used configuration</td>
<td>0</td>
</tr>
<tr>
<td>5. Transaction rate</td>
<td>1</td>
</tr>
<tr>
<td>6. Online Data Entry</td>
<td>5</td>
</tr>
<tr>
<td>7. End-user Efficiency</td>
<td>0</td>
</tr>
<tr>
<td>8. On-line update</td>
<td>3</td>
</tr>
<tr>
<td>9. Complex Processing</td>
<td>1</td>
</tr>
<tr>
<td>10. Reusability</td>
<td>0</td>
</tr>
<tr>
<td>11. Installation ease</td>
<td>2</td>
</tr>
<tr>
<td>12. Operational Ease</td>
<td>3</td>
</tr>
<tr>
<td>13. Multiple sites</td>
<td>4</td>
</tr>
<tr>
<td>14. Facilitation of Change</td>
<td>0</td>
</tr>
</tbody>
</table>

**Total GSC Score = 25**
As you may remember, when we calculated the function point count for this system, we got a result of 28.

If we apply the GSC extension, this count will be modified as follows.

\[
\text{VAF} = 0.65 + (0.01 \times 25) = 0.9 \\
\text{FC} = 28 \times 0.9 = 25.2
\]

Note that the GSC extension can increase or decrease the original count.

In larger systems, the GSC extension will have a much more significant influence on the Function Point Count.
Defect Density

- A metric which describes how many defects occur for each size/functionality unit of a system
- Can be based on LOC or Function Points

\[
\frac{\#\text{defects}}{\text{system\_size}}
\]
Failure Rate

- Rate of defects over time
- May be represented by the $\lambda$ (lambda) symbol

\[
\lambda = \frac{R(t_1) - R(t_2)}{(t_2 - t_1) \times R(t_1)}
\]

where,

- $t_1$ and $t_2$ are the beginning and ending of a specified interval of time
- $R(t)$ is the reliability function, i.e. probability of no failure before time $t$
Calculate the failure rate of system \( X \) based on a time interval of 60 days. The probability of no failure at time day 0 was calculated to be 0.85 and the probability of no failure on day 5 was calculated to be 0.20.
Example of Failure Rate \((2/2)\)

\[
\lambda = \frac{R(t_1) - R(t_2)}{(t_2 - t_1) \times R(t_1)}
\]

\[
\lambda = \frac{0.85 - 0.2}{60 \times 0.85}
\]

\[
= \frac{0.65}{51}
\]

\[
= 0.013 \text{  Failures per day}
\]
Mean Time Between Failure (MTBF)

- MTBF is useful in safety-critical applications (e.g. avionics, air traffic control, weapons, etc)
- The US government mandates that new air traffic control systems must not be unavailable for more than 30 seconds per year

\[ MTBF = \frac{1}{\lambda} \]
Consider our previous example where we calculated the failure rate ($\lambda$) of a system to be 0.013. Calculate the MTBF for that system.

$$MTBF = \frac{1}{\lambda}$$

$$= 76.9 \text{ days}$$

This system is expected to fail every 76.9 days.
McCabe’s **Cyclomatic Complexity Metric**

- Complexity is an important attribute to measure
- Measuring Complexity helps us
  - Predict testing effort
  - Predict defects
  - Predict maintenance costs
  - Etc
- Cyclomatic Complexity Metric was designed by McCabe in 1976
- Aimed at indicating a program’s testability and understandability
- It is based on graph theory
- Measures the number of linearly independent paths comprising the program
McCabe’s *Cyclomatic Complexity* Metric

The formula of cyclomatic complexity is:

\[ M = V(G) = e - n + 2p \]

where

\[ V(G) = \text{cyclomatic number of Graph G} \]
\[ e = \text{number of edges} \]
\[ n = \text{number of nodes} \]
\[ p = \text{number of unconnected parts of the graph} \]
Example: Cyclomatic Complexity

Consider the following flowchart...

Calculating cyclomatic complexity

\[ e = 7, \ n=6, \ p=1 \]

\[ M = 7 - 6 + (2 \times 1) = 3 \]
McCabe’s Cyclomatic Complexity

- Note that the number delivered by the cyclomatic complexity is equal to the number of different paths which the program can take.
- Cyclomatic Complexity is additive. i.e. $M(G_1 \text{ and } G_2) = M(G_1) + M(G_2)$
- To have good testibility and maintainability, McCabe recommends that no module have a value greater than 10.
- This metric is widely used and accepted in industry.