Quotes

“Measure what is measurable, and make measurable what is not so”.

Galileo Galilei:

- *“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 meagre and unsatisfactory kind”.*

William Thomson (Lord Kelvin)

Quotes

- *"You can't control what you can't measure".*

Tom DeMarco

- *"We must be bold in our attempts at measurement. Just because no one has measured some attribute of interest does not mean that it cannot be measured satisfactorily".*

Fenton and Pfleeger

Vocabulary

- Measure
- Indirect measure
- Metric
- Composite metric
- Models for measurement
- Generic methods

Why Measure Software?

- Determine the quality of the current product or process

- Predict qualities of a product/process

- Improve quality of a product/process

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*.

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

Levels of Measurement

Various scales of measurements exist:

– Nominal Scale
– Ordinal Scale
– Interval Scale
– Ratio Scale

The Nominal Scale (1/2)

Example: ***A <u>religion</u> nominal scale***

Joe	Michelle
Rachel	Christine
Michael	James
Clyde	Wendy

Catholic

Muslim

Other

Jewish

The Nominal Scale (2/2)

- The most simple measurment 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

The Ordinal Scale (1/2)

Example: ***A degree-classification ordinal scale***

Joe	Michelle
Rachel	Christine
Michael	James
Clyde	Wendy

1st Class

2nd Class

3rd Class

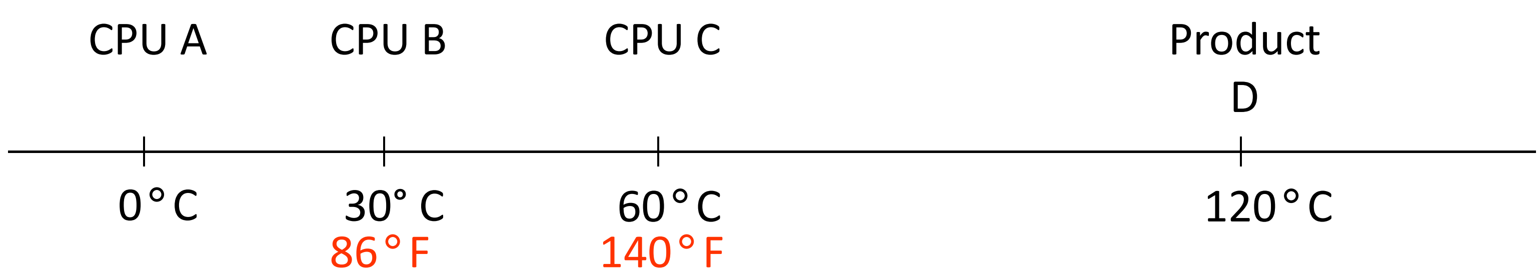
Failed

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



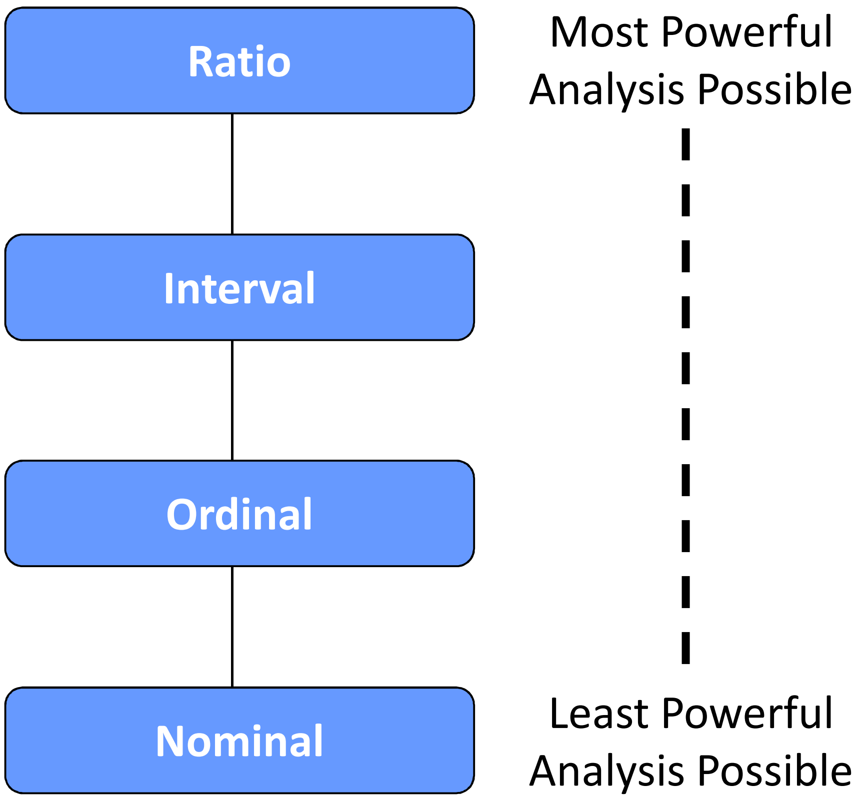
Temperature of Different CPUs

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



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 particalar 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

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.

$$\frac{\text{Number of satisfied customers}}{\text{Total number of customers}}$$

The 3 Ps of Software Measurment

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.

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 make data collection automatic
- Can expensive
  - Sometimes difficult to convince management

Collecting Software Engineering Data

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 (2/3)

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

Examples of Metrics Programmes (3/3)

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

Defect Density

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

$$\frac{\#defects}{system\_size}$$

Failure Rate

- Rate of defects over time
- May be represented by the λ (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

Example of Failure Rate (1/2)

Calculate the failure rate of system ***X*** based on a time interval of 60 days of testing. The probability of failure at time day 0 was calculated to be ***0.85*** and the probability of failure on day 60 was calculated to be ***0.2***.

Example of Failure Rate (2/2)

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

$$\lambda = \frac{0.85 - 0.2}{60 \times 0.85}$$

$$= \frac{0.65}{51}$$

$$= 0.013$$ **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

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

MTBF Example

Consider our previous example where we calculated the failure rate (λ) 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:

$$\mathbf{M} \quad = \mathbf{V(G) = e - n + 2p}$$

where

V(G) = cyclomatic number of Graph G

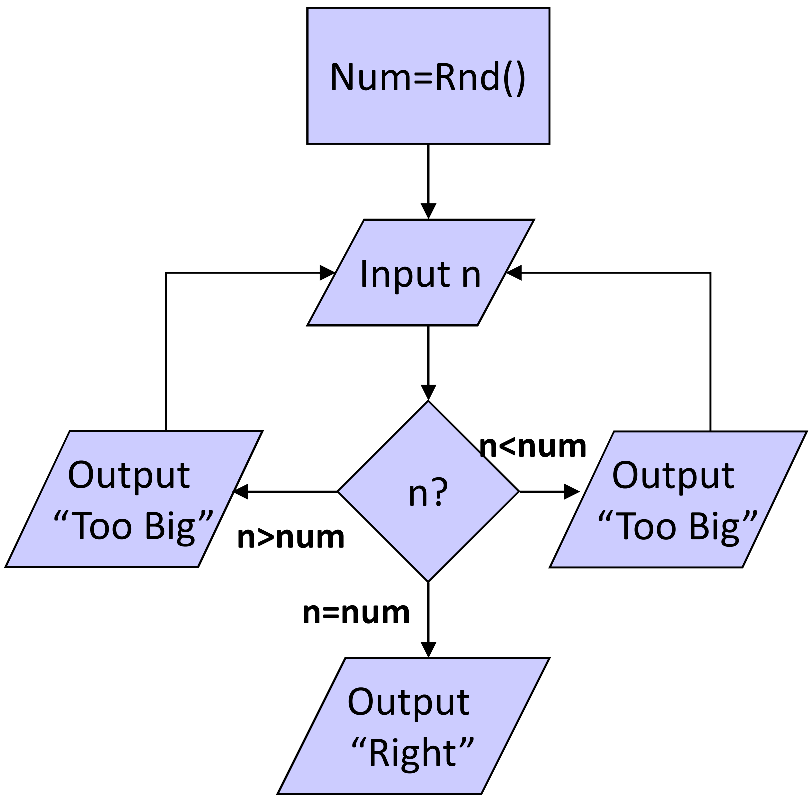
e = number of edges

n = number of nodes

p = number of unconnected parts of the graph

Example: Cyclomatic Complexity

Consider the following flowchart…



<u>Calculating cyclomatic complexity</u>

e = 7, n=6, p=1

M = 7 - 6 + (2x1) = **3**

Halstead’s Software Science (1/3)

- Halstead (1979) distinguished software science from computer science
- **Premise:** Any programming task consists of selecting and arranging a finite number of progam “tokens”
- Tokens are basic syntactic units distinguishable by a compiler
- Computer Program: A collection of tokens that can be classified as either operators or operands

Halstead's Software Science (2/3)

- Halstead (1979) distinguished software science from computer science
- Primitives:

  n_1 = # of distinct operators appearing in a program

  n_2 = # of distinct operands appearing in a program

  N_1 = total # of operator occurences

  N_2 = total # of operand occurences
- Based on these primitive measures, Halstead defined a series of equations

Halstead's Software Science (3/3)

Vocabulary (n) $n = n_1+n_2$

Length (N) $N = N_1 + N_2$

Volume (V) $V = N \log_2(n)$ ← #bits required to represent a program

Level (L) $L = V^* / V$ ← Measure of abstraction and therefore complexity

Difficulty (D) $D = N/N^*$

Effort (E) $E = V/L$

Faults (B) $B = V/S^*$

Where:

$V^* = 2 + n_2 \times \log_2(2 + n_2)$

M* = average number of decisions between errors (3000 according to Halstead)

Example

```
if (k < 2)
{
  if (k > 3)
    x = x*k;
}
```

- Distinct operators: if ( ) { } > < = * ;
- Distinct operands: k 2 3 x
- $n_1 = 10$
- $n_2 = 4$
- $N_1 = 13$
- $N_2 = 7$

Halstead's Metrics

- Amenable to experimental verification [1970s]

- Program length: $N = N_1 + N_2$
- Program vocabulary: $n = n_1 + n_2$

- Estimated length: $\hat{N} = n_1 \log_2 n_1 + n_2 \log_2 n_2$
  - Close estimate of length for well structured programs

- Purity ratio: $PR = \hat{N}/N$

Program Complexity

- Volume: $V = N \log_2 n$
  - Number of bits to provide a unique designator for each of the n items in the program vocabulary.

- Difficulty

$$D = \frac{n_1}{2} \times \frac{N_2}{n_2}.$$

- Program effort: $E=D*V$
  - This is a good measure of program understandability

McCabe’s Complexity Measures

- McCabe’s metrics are based on a control flow representation of the program.
- A program graph is used to depict control flow.
- Nodes represent processing tasks (one or more code statements)
- Edges represent control flow between nodes

Process Metrics

Why measure the process?

- The process creates the product
- If we can improve the process, we indirectly improve the product
- Through measurement, we can ***understand***, ***control*** and ***improve*** the process
- This will lead to us engineering quality into the process rather than simply taking product quality measurements when the product is done
- We will look briefly at a number of process metrics

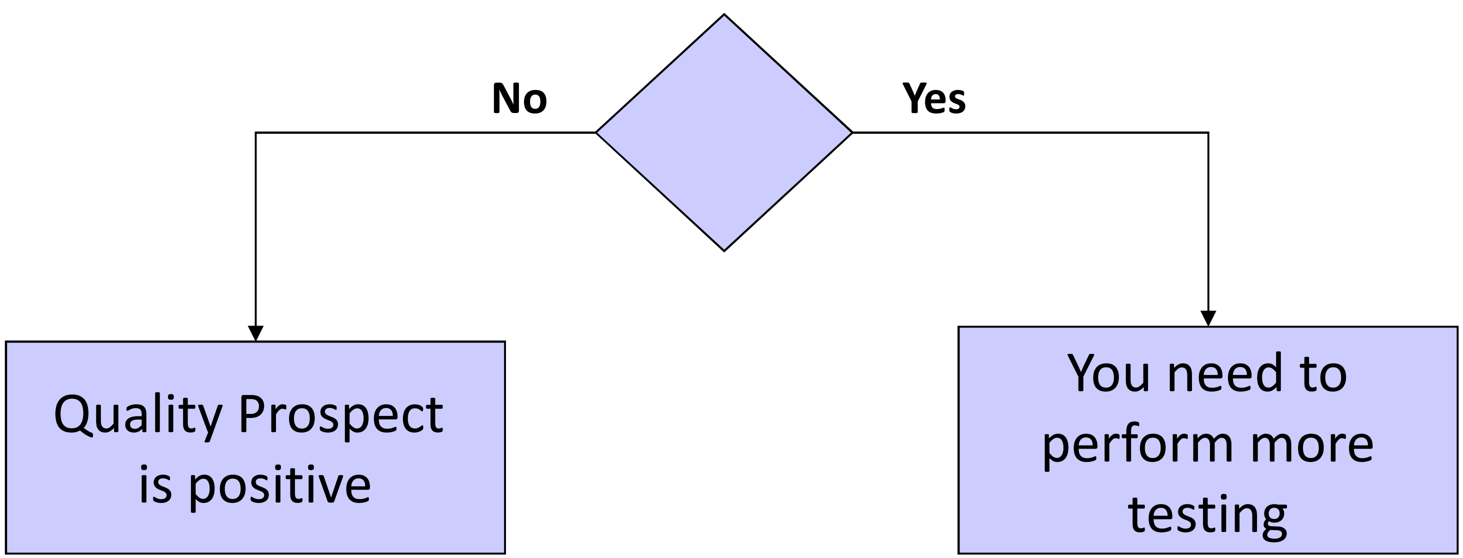
Defect Density During Machine Testing

- Defect rate during formal testingis usually positively correlated with the defect rate experienced in the field
- Higher defect rates found during testing is an indicator that higher defect rates will be experienced in the field
- **Exception:** In the case of exceptional testing effort or more effective testing methods being employed
- It is useful to monitor defect density metrics of subsequent releases of the same product
- In order to appraise product quality, consider the following scenarios

Defect Density During Machine Testing

Scenario 1: Defect rate during testing is the same or lower than previous release.

Reasoning: Does the testing for the current release deteriorate?



People Metrics

Some people metrics…

For individual developers or teams:

- Cost per Function Point
- Mean Time required to develop a Function Point
- Defects produced per hour
- Defects produced per function point

Object Oriented Design Metrics

Unique OO Characteristics (1/2)

- **Encapsulation**
  - Binding together of a collection of items
    - State information
    - Algorithms
    - Constants
    - Exceptions
    - …
- **Abstraction and Information Hiding**
  - Suppressing or hiding of details
  - One can use an object’s advertised methods without knowing exactly how it does its work

Unique OO Characteristics (2/2)

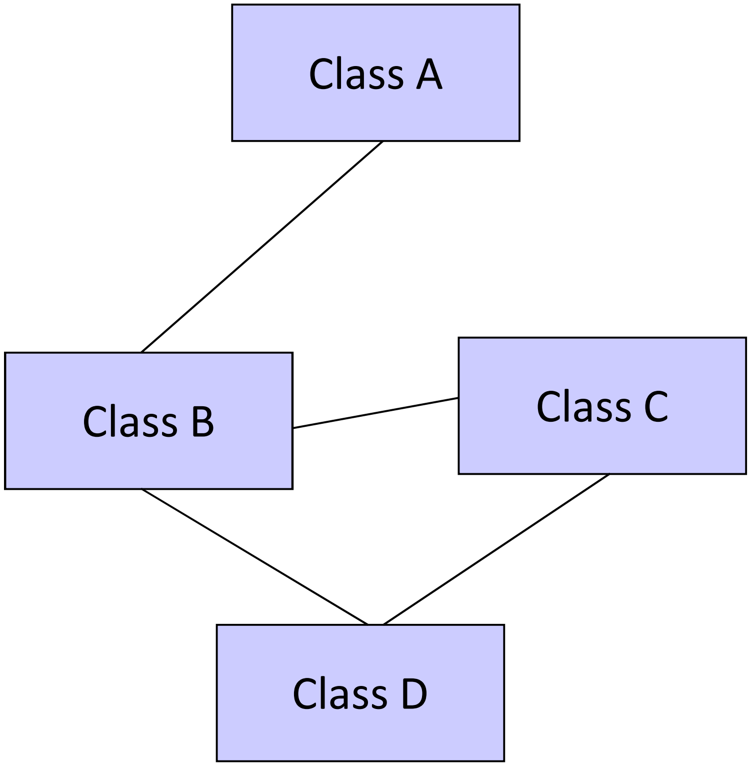
- **Inheritance**
  - Objects may acquire characteristics of one or more other objects
  - The way inheritance is used will affect the overall quality of a system
- **Localisation**
  - Placing related items in close physical proximity to each other
  - In the case of OO, we group related items into objects, packages, ets

Measurable Structures in OO (1/5)

- **Class**
  - Template from which objects are created
  - Class design affects overall:
    - Understandability
    - Maintainability
    - Testability
  - Reusability is also affected by class design
    - E.g. Classes with a large number of methods tend to be more application specific and less reusable

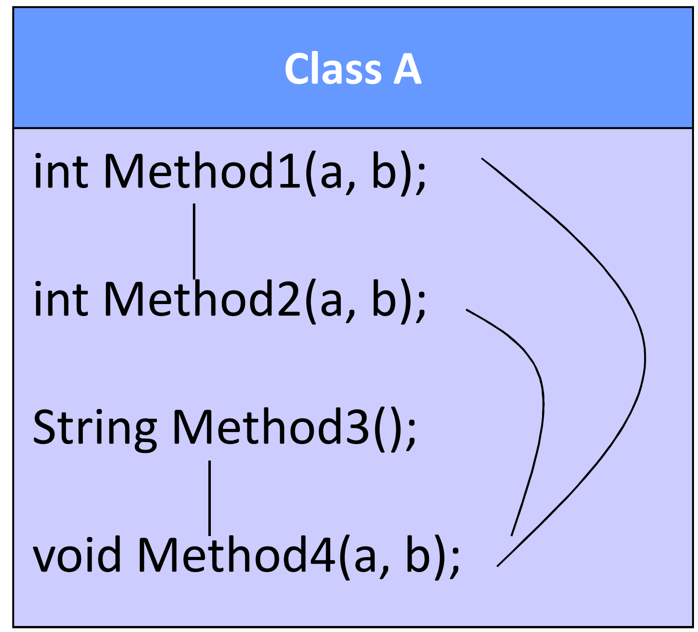
Measurable Structures in OO (3/5)

- **Coupling**
  - A measure of the strength of association established by connections between different entities
  - Occurs through:
    - Use of an object's methods
    - Inheritance



Measurable Structures in OO (4/5)

- **Cohesion**
  - The degree to which methods in a class are related to each other
  - Effective OO designs maximise cohesion because they promote encapsulation
  - A high degree of cohesion indicates:
    - Classes are self contained
    - Fewer messages need to be passed (more efficiency)



Measurable Structures in OO (5/5)

- **Inheritance**
  - A mechanism which allows an object to acquire the characteristics of one or more other objects
  - Inheritance can reduce complexity by reducing the number of methods and attributes in child classes
  - Too much inheritance can make the system difficult to maintain

Weighted Methods Per Class (WMC)

- Consider the class C with methods m_1, m_2, … $m_{n.}$
- Let c_1, c_2 … c_n be the complexity of these methods.

$$WMC = \sum_{i=1}^{n} c_i$$

Weighted Methods Per Class (WMC)

- Refers to the complexity of an object
- The number of methods involved in an object is an indicator of how much time and effort is required to develop
- Complex classes also make their child classes complex
- Objects with large number of methods are likely to be more application-specific and less reusable
- Guidelines: WMC of 20 for a class is good but do not exceed 40.
- Affects:
  - Understandability, Maintainability, Reusability

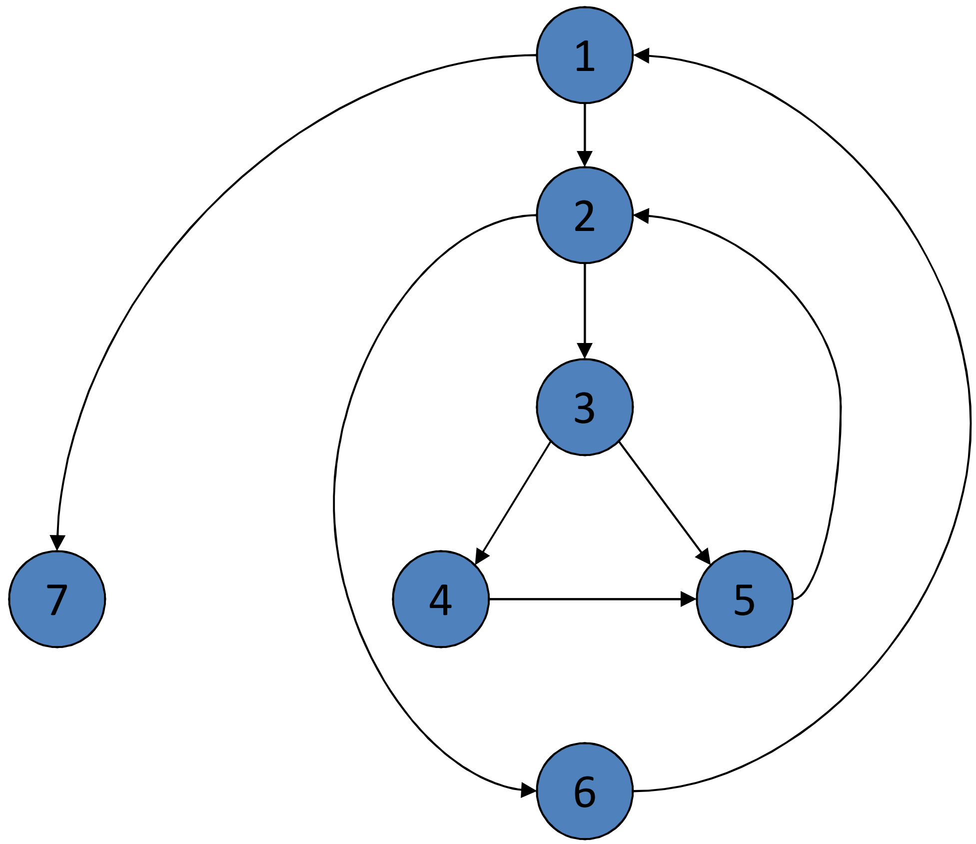
Cyclomatic Complexity

- Set of independent paths through the graph (basis set)
- $V(G) = E - N + 2$
  - E is the number of flow graph edges
  - N is the number of nodes
- $V(G) = P + 1$
  - P is the number of predicate nodes

Example

```
i = 0;
while (i<n-1) do
   j = i + 1;
   while (j<n) do
      if A[i]<A[j] then
         swap(A[i], A[j]);
   end do;
   i=i+1;
end do;
```

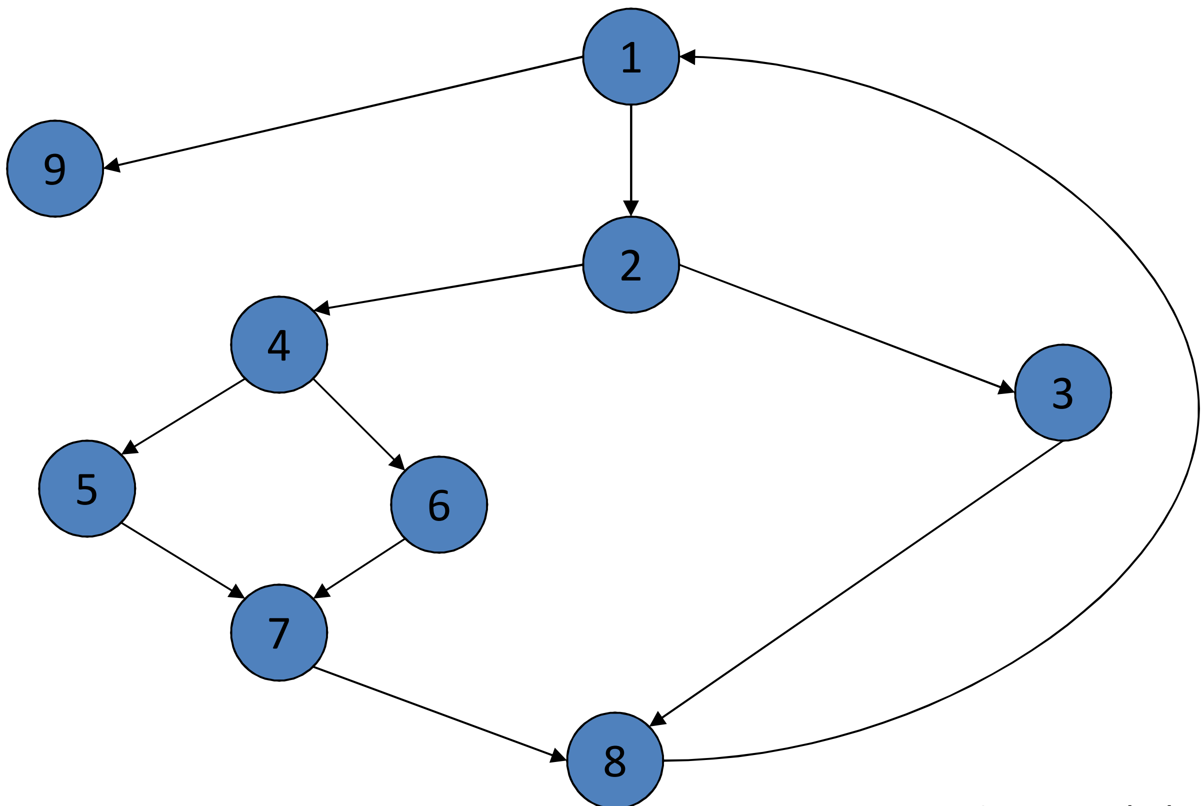
Flow Graph



Computing V(G)

- $V(G) = 9 - 7 + 2 = 4$
- $V(G) = 3 + 1 = 4$
- Basis Set
  - 1, 7
  - 1, 2, 6, 1, 7
  - 1, 2, 3, 4, 5, 2, 6, 1, 7
  - 1, 2, 3, 5, 2, 6, 1, 7

Another Example



What is V(G)?

Meaning

- V(G) is the number of (enclosed) regions/areas of the planar graph
- Number of regions increases with the number of decision paths and loops
- A quantitative measure of testing difficulty and an indication of ultimate reliability
- Experimental data shows value of V(G) should be no more then 10 - testing is very difficulty above this value

McClure's Complexity Metric

- Complexity = C + V
  - C is the number of comparisons in a module
  - V is the number of control variables referenced in the module
  - decisional complexity
- Similar to McCabe's but with regard to control variables

Method Inheritance Factor

$$\text{MIF} = \frac{\sum_{i=1}^{n} M_i(C_i)}{\sum_{i=1}^{n} M_a(C_i)} \quad .$$

- $M_i(C_i)$ is the number of methods inherited and not overridden in C_i
- $M_a(C_i)$ is the number of methods that can be invoked with C_i
- $M_d(C_i)$ is the number of methods declared in C_i

Metric tools

- McCabe & Associates (founded by Tom McCabe, Sr.)
  - The Visual Quality ToolSet
  - The Visual Testing ToolSet
  - The Visual Reengineering ToolSet
- Metrics calculated
  - McCabe Cyclomatic Complexity
  - McCabe Essential Complexity
  - Module Design Complexity
  - Integration Complexity
  - Lines of Code
  - Halstead

CCCC

- A metric analyser C, C++, Java, Ada-83, and Ada-95 (by Tim Littlefair of Edith Cowan University, Australia)

- Metrics calculated
  - Lines Of Code (LOC)
  - McCabe's cyclomatic complexity
  - C&K suite (WMC, NOC, DIT, CBO)

- Generates HTML and XML reports

- freely available

- http://cccc.sourceforge.net/

Jmetric

- OO metric calculation tool for Java code (by Cain and Vasa for a project at COTAR, Australia)

- Requires Java 1.2 (or JDK 1.1.6 with special extensions)

- Metrics
  - Lines Of Code per class (LOC)
  - Cyclomatic complexity
  - LCOM (by Henderson-Seller)

- Availability
  - is distributed under GPL

- http://www.it.swin.edu.au/projects/jmetric/products/jmetric/

JMetric tool result

Project | Package | Class | Method | Variable

```
PACKAGE: com.rolemodelsoft.drawlet.awt

Classes: 9
Public Classes: 9

Lines Of Code: 442
Statement Count: 264

Methods: 101
Public Methods: 92
```

Metric	Average	Std. Dev	Median	Mode	Max	Min	Skew
Class Stats							
Lines of C...	49.1	34.1	27.0	26.0	112.0	19.0	1.9
Statements	29.3	22.6	16.0	15.0	78.0	9.0	1.8
LCOM	0.8	0.2	0.8	0.5	1.0	0.5	-1.0
No. Metho...	11.2	8.7	8.0	8.0	27.0	3.0	1.1
Collaborat...	8.7	4.0	9.0	4.0	17.0	4.0	-0.3
Public							
Methods	10.2	8.0	8.0	3.0	25.0	3.0	0.8
Variables	0.0	0.0	0.0	0.0	0.0	0.0	0.0
No Scope							
Methods	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Variables	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Methods							
Lines of C...	4.0	4.0	2.0	2.0	24.0	1.0	1.5
Statements	2.6	3.2	1.0	1.0	16.0	0.0	1.5
Cyclomati...	1.3	0.6	1.0	1.0	4.0	1.0	1.6
Collaborat...	1.8	1.5	1.0	1.0	9.0	0.0	1.6

GEN++

(*University of California, Davis and Bell Laboratories)*

- GEN++ is an application-generator for creating code analyzers for C++ programs
  - simplifies the task of creating analysis tools for the C++
  - several tools have been created with GEN++, and come with the package
  - these can both be used directly, and as a springboard for other applications
- Freely available
- http://www.cs.ucdavis.edu/~devanbu/genp/down-red.html

More tools on Internet

- A Source of Information for Mission Critical Software Systems, Management Processes, and Strategies

http://www.niwotridge.com/Resources/PM-SWEResources/SWTools.htm

- Defense Software Collaborators (by DACS)

http://www.thedacs.com/databases/url/key.hts?keycode=3

http://www.qucis.queensu.ca/Software-Engineering/toolcat.html#label208

- Object-oriented metrics

http://me.in-berlin.de/~socrates/oo_metrics.html

- Software Metrics Sites on the Web (Thomas Fetcke)

- Metrics tools for C/C++ (Christofer Lott)

http://www.chris-lott.org/resources/cmetrics/