COMP23420 Software Engineering

Semester 2

Week 9: Software Metrics

Dr Liping Zhao
School of Computer Science

Today's Objectives

- What are metrics?
- Why do we need them?
- What can we measure?
- How do we measure?

Definitions

- ***Metric*** – a quantitative measure of degree to which a system, component or process possesses a given attribute.
  - E.g. Two errors were discovered by customers in 18 months (more meaningful than saying that 2 errors were found)

- ***Measure*** – a quantitative indication of extent, amount, dimension, capacity, or size of some attribute of a product or process.
  - E.g., Two errors were discovered by customers.
  - Joe's body temperature is 99° Fahrenheit

Why Measure Software?

- Determine the quality of the current product or process
- Predict quality of a product/process
- Improve quality of a product/process
- Estimate software development cost and effort

The 3 Ps of Software Measurement (What can we measure?)

With regards to software, we can measure:

- Product
- Process
- People

Measuring the Product (Product Metrics)

- 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 (Process Metrics)

- What lifecycle do we use?
- What deliverables are produced?
- How can the process help to produce products faster?
- How can the process help to produce better products?

Measuring the People (People Metrics)

- 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.
- BUT: Measuring the people is a commonplace in the academic world!

Product Metrics (How do we measure?)

- Size-oriented metrics
- Defects-based metrics
- Complexity metrics
- Object-oriented metrics
- Software quality metrics
- Cost-metrics
- Time metrics
- High-level design metrics
- Metrics for coupling
- etc

9

Product Metrics (Our Focus)

- Size-oriented metrics
- Defects-based metrics
- Complexity metrics
- Object-oriented metrics
- Software quality metrics

10

Size-Oriented Metrics

- Size of a system
- *LOC* - Lines Of Code
- *KLOC* - 1000 Lines Of Code
- *SLOC* – Statement Lines of Code (ignore whitespace)

- Typical Measures:
  – Errors/KLOC, Defects/KLOC, Cost/LOC, Documentation Pages/KLOC

Problems with LOC Metrics

- Same system developed with different programming languages will give different LOC readings
- Same system developed by different developers using the same language will give different LOC readings
- To calculate LOC you have to wait until the system is implemented
- This is not adequate when management requires prediction of cost and effort

12

Problems with LOC Metrics

A poor indicator of productivity:

- Ignores software reuse, code duplication, benefits of redesign
- The lower level the language, the more productive the programmer!
- The more verbose the programmer, the higher the productivity!

13

Defect Density

- A rate-metric which describes how many defects occur for each size/functionality unit of a system

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

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

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

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

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

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

$= 0.013$ **Failures per day**

Complexity Metrics

- Complexity is an important attribute to measure
- Halstead's Complexity Metrics
- Cyclomatic Complexity Metrics

Halstead's Complexity Metrics

- Proposed by Maurice Howard Halstead in 1977 & still in use today
- **View a program as sequence of operators and their associated operands**
- Computed directly from source code, in a static manner
- Parameters:
  - n_1 - number of distinct operators
  - n_2 - number of distinct operands
  - N_1 - total number of operators
  - N_2 - total number of operands

19

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 Complexity Metrics

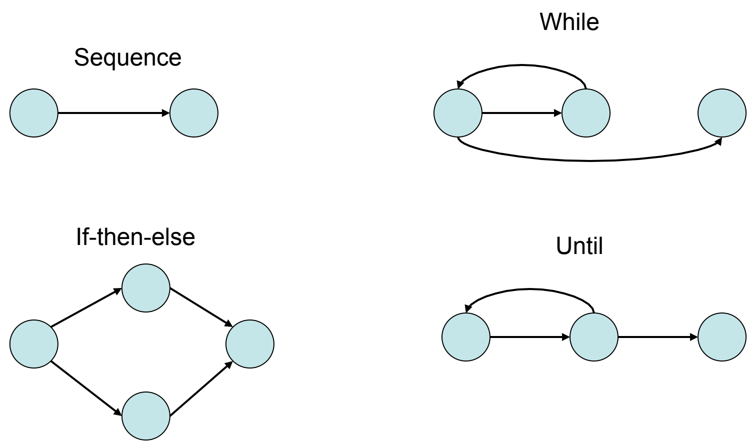
Metric	Meaning	Mathematical Representation
n	Vocabulary	$n = n_1 + n_2$
N	Length	$N = N_1 + N_2$
$\hat{N}$	Estimated Length	$\hat{N} = n_1 \log_2 n_1 + n_2 \log_2 n_2$
V	Volume	$V = N \log_2 n$
D	Difficulty	$D = n_1/2 * N_2/ n_2$
E	Efforts	$E=D*V$
B	Errors	$B = V/3000$
T	Testing Time	$T = E/S$, where $S = 18$ seconds
PR	Purity Ratio	$PR = \hat{N} /N$

21

Cyclomatic Complexity Metrics

- Designed by McCabe in 1976 and still in use today
- **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

Control Flow Graph Notation



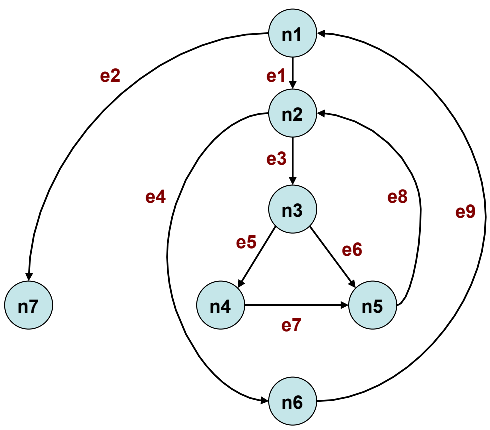
Cyclomatic Complexity Metrics

- $V(G) = e - n + 2$
  - $V(G)$ is Cyclomatic complexity
  - e is the total number of edges
  - n is the total number of nodes
- Cyclomatic complexity of a module should not exceed 10, as testing is very difficulty above this value

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]);
      j=j+1;
end do;
  i=i+1;
end do;
```

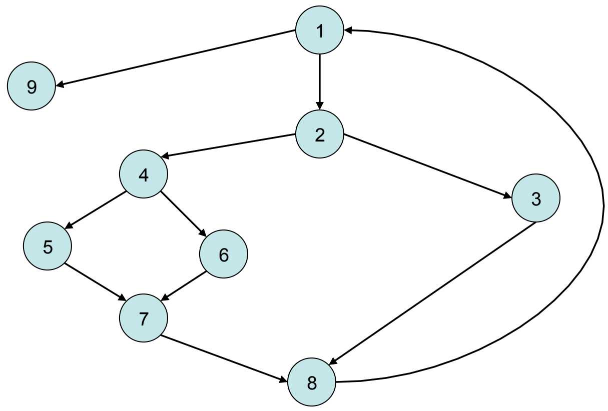
Control Flow Graph




```
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]);
      j=j+1;
end do;
  i=i+1;
end do;
```

$V(G) = 9 - 7 + 2 = 4$

Another Example



What is V(G)?

Object-Oriented Metrics and their Indicators

- **Class size (CS)**
  - Total number of operations
  - Total number of attributes
  - ➢ A large number indicates too much responsibility for a class
- **Number of operations overridden (NOO)**
  - ➢ A large number indicates possible design problems
  - ➢ Poor abstraction in inheritance hierarchy
- **Number of operations added (NOA)**
  - ➢ As depth of inheritance increases NOA should decrease
- **Number of children (NOC)**
  - ➢ As NOC grows, reuse increases, but the abstraction may be diluted

28

Software Quality Metrics

- **Correctness** – degree to which a program operates according to specification
  - Defects/KLOC
  - Defect is a verified lack of conformance to requirements
  - Failures/hours of operation
- **Maintainability** – degree to which a program is open to change
  - Mean time to change
  - Change request to new version (Analyze, design etc)
  - Cost to correct
- **Integrity** - degree to which a program is resistant to outside attack
  - Fault tolerance, security & threats
- **Usability** – easiness to use
  - Training time, skill level necessary to use, Increase in productivity, subjective questionnaire or controlled experiment

Metric Tools

- **Jmetric**: OO metric calculation tool for Java code (by Cain and Vasa, Australia)
  - Available: http://jmetric.sourceforge.net/
  - Metrics calculated:
    - Lines Of Code per class (LOC)
    - Cyclomatic complexity
    - LCOM (Lack of Cohesion in Methods)
- **CCCC**: A metric analyser C, C++, Java, Ada-83, and Ada-95 (by Tim Littlefair of Edith Cowan University, Australia)
  - Available: http://cccc.sourceforge.net/
  - Metrics calculated
    - Lines Of Code (LOC)
    - McCabe's cyclomatic complexity

Summary

- A software metric is a quantitative measure of degree to which a system, component or process possesses a given attribute
- We measure software for all sorts of reasons, such as determining, improving and predicting the quality of products or processes, and estimating development cost and effort
- We can measure products, processes and people
- There are a large number of software metrics. This lecture has introduced some of the most important product metrics: size-oriented, defect density, failure rate, and complexity metrics.
- The lecture has provided a high-level overview of object-oriented metrics and software quality metrics.
- There are many metric tools available. You can try out Jmetric and CCCC in your project

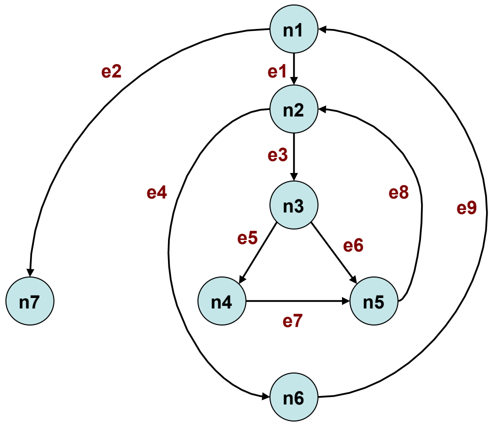
31

References

- Software Metrics, Michael L. Collard, http://www.sdml.info/collard/seF08/notes/
- Software Measurement, Lecture Notes, Mark Micallef, University of Malta.
- Software Metrics: A Rigorous & Practical Approach, Norman E. Fenton, Shari I. Pfleeger, Thompson Computer Press, Third Ed., 2015 (First Ed. 1996).
- Software Engineering, I. Sommerville, Addison-Wesley, 10th Ed., 2012.
- The Measurement of a Design Structural and Functional Complexity, Dan Braha and Oded Maimon, IEEE Trans SMC—Part A, Vol. 28, No. 4, July 1998

32

Control Flow Graph



V(G) = 9 – 7 + 2 = 4