Software Metrics

“How do we measure the software?”

Lecture Objectives

- To understand the importance of measurement in software engineering
- To describe and compare the different metrics that can be used for measuring software
- To understand the important factors that affect the measurement of software

Engineering Method

- Scientific
- Precision
- Accuracy
- Repeatable
- Controllable
- Quality

Software Metrics

Software Metrics: What and Why ?

1. How to measure the size of a software?
2. How much will it cost to develop a software?
3. How many bugs can we expect?
4. When can we stop testing?
5. When can we release the software?

Software Metrics

6. What is the complexity of a module?
7. What is the module strength and coupling?
8. What is the reliability at the time of release?
9. Which test technique is more effective?
10. Are we testing hard or are we testing smart?
11. Do we have a strong program or a week test suite?

Why Do We Measure?

- To indicate the quality of the product
- To assess the productivity of the people who produce the product
- To assess the benefits derived from new software engineering methods and tools
- To form a baseline for estimation
- To help justify requests for new tools or additional training

Definitions

- *Measure* - quantitative indication of extent, amount, dimension, capacity, or size of some attribute of a product or process.
  - E.g., Number of errors

- *Measurement* - the act of determining a measure

- *Metric* - quantitative measure of degree to which a system, component or process possesses a given attribute. “A handle or guess about a given attribute.”
  - E.g., Number of errors found per person hours expended

Motivation for Metrics

- Estimate the cost & schedule of future projects
- Evaluate the productivity impacts of new tools and techniques
- Establish productivity trends over time
- Improve software quality
- Forecast future staffing needs
- Anticipate and reduce future maintenance needs

Software Metrics

- Areas of Applications

The most established area of software metrics is cost and size estimation techniques.

The prediction of quality levels for software, often in terms of reliability, is another area where software metrics have an important role to play.

The use of software metrics to provide quantitative checks on software design is also a well established area.

Software Metrics

- Categories of Metrics

  i. Product metrics: describe the characteristics of the product such as size, complexity, design features, performance, efficiency, reliability, portability, etc.

  ii. Process metrics: describe the effectiveness and quality of the processes that produce the software product. Examples are:

     - effort required in the process
     - time to produce the product
     - effectiveness of defect removal during development
     - number of defects found during testing
     - maturity of the process

Software Metrics

ii. Project metrics: describe the project characteristics and execution. Examples are :

- number of software developers
- staffing pattern over the life cycle of the software
- cost and schedule
- productivity

Types of Measures

- Direct Measures (internal attributes)
  - Cost, effort, LOC, speed, memory

- Indirect Measures (external attributes)
  - Functionality, quality, complexity, efficiency, reliability, maintainability

Example: length of a pipe is a direct measure.
the quality of the pipes can only be measured indirectly by finding the ratio of the accepted pipes to the rejected.

Size Oriented Metrics

Size-Oriented Metrics

- Based on the “size” of the software produced.
- Project data measured, including cost and effort, pages, defects…etc.
- Mainly uses the LOC as the normalization value.
- Advantages: easily counted, large body of literature and data based on LOC
- Disadvantages: language dependent, programmer dependent.
- Useful for projects with similar environment.
- Therefore, size-oriented metrics are not universally accepted as the best way to measure the software process.

Size-Oriented Metrics

- Size of the software produced

- *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

Example of Project Measurements

project	effort (person-month)	cost ($)	kLOC	Doc. (pgs)	errors	people
Proj_1	24	168,000	12.1	365	29	3
Proj_2	62	440,000	27.2	1224	86	5

Example of Size-Oriented Metrics

• Productivity	= Size / Effort
	= kLOC / person-month
• Quality	= Errors / Size
	= Errors / kLOC
• Cost	= $ / kLOC
• Documentation	= pages / kLOC

- Other metrics can also be developed like: errors/KLOC, page/KLOC...etc.
- Or errors/person-month, LOC/person-month, cost/page.

Function Oriented Metrics

Function-Oriented Metrics

- Based on “functionality” delivered by the software as the normalization value.
- Functionality is measured indirectly
- Function points (FP) measure- derived using an empirical relationship based on countable (direct) measures of software’s information domain and assessments of software complexity

Alan Albrecht while working for IBM, recognized the problem in size measurement in the 1970s, and developed a technique (which he called Function Point Analysis), which appeared to be a solution to the size measurement problem.

The principle of Albrecht's function point analysis (FPA) is that a system is decomposed into functional units.

▪ Inputs	:	information entering the system
▪ Outputs	:	information leaving the system (reports, screens, error messages)
▪ Enquiries	:	requests for instant access to information e.g. an on-line input that generates an immediate on-line output response
▪ Internal logical files	:	information held within the system
▪ External interface files	:	information held by other system that is used by the system being analyzed.

Steps In Calculating FP

1. Count the information domain values.
2. Assess the complexity of the values.
3. Calculate the raw FP (see next table).
4. Rate the complexity factors to produce the complexity adjustment value (CAV)

   CAV $= \sum F_i$ i = 1 to 14
5. Calculate the adjusted FP as follows:

   FP = raw FP x [0.65 + 0.01 x CAV]

Counting function points

Functional Units	Weighting factors		
	Low	Average	High
External Inputs (EI)	3	4	6
External Output (EO)	4	5	7
External Inquiries (EQ)	3	4	6
External logical files (ILF)	7	10	15
External Interface files (EIF)	5	7	10

Table 1 : Functional units with weighting factors

Table 2: UFP calculation table

Functional Units	Count	Complexity		Complexity Totals	Functional Unit Totals
External Inputs (EIs)		Low x 3	=		
		Average x 4	=		
		High x 6	=		
External Outputs (EOs)		Low x 4	=		
		Average x 5	=		
		High x 7	=		
External Inquiries (EQs)		Low x 3	=		
		Average x 4	=		
		High x 6	=		
External logical Files (ILFs)		Low x 7	=		
		Average x 10	=		
		High x 15	=		
External Interface Files (EIFs)		Low x 5	=		
		Average x 7	=		
		High x 10	=		
Total Unadjusted Function Point Count					

The weighting factors are identified for all functional units and multiplied with the functional units accordingly. The procedure for the calculation of Unadjusted Function Point (UFP) is given in table shown above.

The procedure for the calculation of UFP in mathematical form is given below:

$$UFP = \sum_{i=1}^{5} \sum_{J=1}^{3} Z_{ij} w_{ij}$$

Where i indicate the row and j indicates the column of Table 1

W_{ij} : It is the entry of the ith row and jth column of the table 1

Zij : It is the count of the number of functional units of Type *i* that have been classified as having the complexity corresponding to column *j*.

Organizations that use function point methods develop a criterion for determining whether a particular entry is Low, Average or High. Nonetheless, the determination of complexity is somewhat subjective.

$$FP = UFP * CAF$$

Where CAF is complexity adjustment factor and is equal to [0.65 + 0.01 x ΣF_i]. The F_i (i=1 to 14) are the degree of influence and are based on responses to questions given in next slide

Rate Complexity Factors

For each complexity adjustment factor (F), give a rating on a scale of 0 to 5

0 - No influence

1 - Incidental

2 - Moderate

3 - Average

4 - Significant

5 - Essential

Complexity Adjustment Factors (F_i)

1. Does the system require reliable backup and recovery?
2. Are data communications required?
3. Are there distributed processing functions?
4. Is performance critical?
5. Will the system run in an existing, heavily utilized operational environment?
6. Does the system require on-line data entry?
7. Does the on-line data entry require the input transaction to be built over multiple screens or operations?
8. Are the master files updated on-line?
9. Are the inputs, outputs, files, or inquiries complex?

Complexity Adjustment Factors (Continued)

10. Is the internal processing complex?
11. Is the code designed to be reusable?
12. Are conversion and installation included in the design?
13. Is the system designed for multiple installations in different organizations?
14. Is the application designed to facilitate change and ease of use by the user?

Complexity Adjustment Value

- The rating for all the factors, F_1 to F_{14}, are summed to produce the complexity adjustment value (CAV)
- CAV is then used in the calculation of the function point (FP) of the software

Example of Function-Oriented Metrics

• Productivity	= Functionality / Effort
	= FP / person-month
• Quality	= Errors / Functionality
	= Errors / FP
• Cost	= $ / FP
• Documentation	= pages / FP

Example: 1

Consider a project with the following functional units:

Number of user inputs	= 50
Number of user outputs	= 40
Number of user enquiries	= 35
Number of user files	= 06
Number of external interfaces	= 04

Assume all complexity adjustment factors and weighting factors are average. Compute the function points for the project.

Solution

We know

$$UFP = \sum_{i=1}^{5}\sum_{J=1}^{3} Z_{ij} w_{ij}$$

UFP = 50 x 4 + 40 x 5 + 35 x 4 + 6 x 10 + 4 x 7
= 200 + 200 + 140 + 60 + 28 = 628

CAF = (0.65 + 0.01 ΣF_i)
= (0.65 + 0.01 (14 x 3)) = 0.65 + 0.42 = 1.07

FP = UFP x CAF
= 628 x 1.07 = 672

Example: 2

An application has the following:

10 low external inputs, 12 high external outputs, 20 low internal logical files, 15 high external interface files, 12 average external inquiries, and a value of complexity adjustment factor of 1.10.

What are the unadjusted and adjusted function point counts ?

Solution

Unadjusted function point counts may be calculated using as:

$$UFP = \sum_{i=1}^{5} \sum_{J=1}^{3} Z_{ij} w_{ij}$$

= 10 x 3 + 12 x 7 + 20 x 7 + 15 + 10 + 12 x 4

= 30 + 84 +140 + 150 + 48

= 452

FP = UFP x CAF

= 452 x 1.10 = 497.2.

FP Characteristics

- Advantages: language independent, based on data known early in project, good for estimation
- Disadvantages: calculation complexity, subjective assessments, FP has no physical meaning (just a number)

Extended Function Point Metrics

- Feature points
  - applied to systems and engineering software.
  - includes assessment for complex algorithms.

- 3D Function points
  - applied to real-time systems and engineered products (by Boeing)
  - integrates data dimension (normal FP) with functional and control dimensions

Reconciling Metrics Approaches

- A number of studies attempted to relate FP and LOC measures.
- Relationship depends on programming language and quality of design
- Example: one LOC of Ada provides approximately 1.4 times as much “functionality” (on average) of one LOC of Fortran

Comparison of LOC/FP

Programming Language	LOC/FP (average)
Assembly Language	320
C	128
Cobol	105
Fortran	105
Pascal	90
Ada	70
Object-oriented language	30
Fourth-generation language	20
Code generators	15

Halstead’s Software Metrics

Halstead's Software Science

Token Count

The size of the vocabulary of a program, which consists of the number of unique tokens used to build a program is defined as:

$$\eta = \eta_1 + \eta_2$$

where η : vocabulary of a program

η_1 : number of unique operators

η_2 : number of unique operands

Halstead's Metrics

The length of the program in the terms of the total number of tokens used is

$$N = N_1 + N_2$$

where N : program length

N_1 : total occurrences of operators

N_2 : total occurrences of operands

Halstead's Metrics

Volume

$$V = N * \log_2 \eta$$

The unit of measurement of volume is the common unit for size "bits". It is the actual size of a program if a uniform binary encoding for the vocabulary is used.

Program Level

$$L = V^* / V$$

The value of L ranges between zero and one, with L=1 representing a program written at the highest possible level (i.e., with minimum size).

Halstead's Metrics

Program Difficulty

$$D = 1 / L$$

As the volume of an implementation of a program increases, the program level decreases and the difficulty increases. Thus, programming practices such as redundant usage of operands, or the failure to use higher-level control constructs will tend to increase the volume as well as the difficulty.

Effort

$$E = V / L = D * V$$

The unit of measurement of E is elementary mental discriminations.

Halstead’s Metrics

- Estimated Program Length

$$\hat{N} = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2$$

$$\hat{N} = 14 \log_2 14 + 10 \log_2 10$$

= 53.34 + 33.22 = 86.56

Halstead's Metrics

$$N_B = \eta_1 Log_2 \eta_2 + \eta_2 \log_2 \eta_1$$

$$N_c = \eta_1 \sqrt{\eta_1} + \eta_2 \sqrt{\eta_2}$$

$$N_s = (\eta \log_2 \eta) / 2$$

The definitions of unique operators, unique operands, total operators and total operands are not specifically delineated.

Halstead's Metrics

- Counting rules for C language

1. Comments are not considered.
2. The identifier and function declarations are not considered.
3. All the variables and constants are considered operands.
4. Global variables used in different modules of the same program are counted as multiple occurrences of the same variable.

Halstead’s Metrics

5. Local variables with the same name in different functions are counted as unique operands.

6. Functions calls are considered as operators.

7. All looping statements e.g., do {…} while ( ), while ( ) {…}, for ( ) {…}, all control statements e.g., if ( ) {…}, if ( ) {…} else {…}, etc. are considered as operators.

8. In control construct switch ( ) {case:…}, switch as well as all the case statements are considered as operators.

Halstead’s Metrics

9. The reserve words like return, default, continue, break, sizeof, etc., are considered as operators.

10. All the brackets, commas, and terminators are considered as operators.

11. GOTO is counted as an operator and the label is counted as an operand.

12. The unary and binary occurrence of “+” and “-” are dealt separately. Similarly “*” (multiplication operator) are dealt separately.

Halstead’s Metrics

13. In the array variables such as “array-name [index]” “array-name” and “index” are considered as operands and [ ] is considered as operator.

14. In the structure variables such as “struct-name, member-name” or “struct-name -> member-name”, struct-name, member-name are taken as operands and ‘.’, ‘->’ are taken as operators. Some names of member elements in different structure variables are counted as unique operands.

15. All the hash directive are ignored.

Halstead's Metrics

- Potential Volume

η_2^* unique input and output parameters

$$V^* = (2+\eta_2^*)\log_2(2+\eta_2^*)$$

- Estimated Program Level / Difficulty

Halstead offered an alternate formula that estimate the program level.

$$\hat{L} = 2\eta_2 / (\eta_1 N_2)$$

where

$$\hat{D} = \frac{1}{\hat{L}} = \frac{\eta_1 N_2}{2\eta_2}$$

Halstead’s Metrics

- Effort and Time

$$\mathrm{E} = V / \hat{L} = V * \hat{D}$$

$$= (n_1 N_2 N \log_2 \eta) / 2\eta_2$$

$$T = E / \beta$$

β is normally set to 18 since this seemed to give best results in Halstead’s earliest experiments, which compared the predicted times with observed programming times, including the time for design, coding, and testing.

Halstead’s Metrics

- Language Level

$$\lambda = L \times V^* = L^2 V$$

Using this formula, Halstead and other researchers determined the language level for various languages as shown below.

Language	*Language Level* λ	*Variance* σ
PL/1	1.53	0.92
ALGOL	1.21	0.74
FORTRAN	1.14	0.81
CDC Assembly	0.88	0.42
PASCAL	2.54	–
APL	2.42	–
C	0.857	0.445

Example

List out the operators and operands and also calculate the values of software science measures like η, N, V, E, λ *etc.*

1.	int. sort (int x[ ], int n)
2.	{
3.	int i, j, save, im1;
4.	/*This function sorts array x in ascending order */
5.	If (n<2) return 1;
6.	for (i=2; i<=n; i++)
7.	{
8.	im1=i-1;
9.	for (j=1; j<=im; j++)
10.	if (x[i] < x[j])
11.	{
12.	Save = x[i];
13.	x[i] = x[j];
14.	x[j] = save;
15.	}
16.	}
17.	return 0;
18.	}

Solution

The list of operators and operands

Operators	*Occurrences*	*Operands*	*Occurrences*
int	4	SORT	1
( )	5	x	7
,	4	n	3
[ ]	7	i	8
if	2	j	7
<	2	save	3

(Contd.)...

;	11	im1	3
for	2	2	2
=	6	1	3
–	1	0	1
< =	2	—	—
+ +	2	—	—
return	2	—	—
{ }	3	—	—
$\eta_1 = 14$	$N_1 = 53$	$\eta_2 = 10$	$N_2 = 38$

Here N_1=53 and N_2=38

1. The program length $N=N_1+N_2=91$

2. Vocabulary of the program $\eta = \eta_1 + \eta_2 = 14 + 10 = 24$

3. Volume $V = N \times \log_2 \eta$

$= 91 \times \log_2 24 = 417$ bits

4. The estimated program length $\hat{N}$ of the program

$= 14 \log_2 14 + 10 \log_2 10$

$= 14 * 3.81 + 10 * 3.32$

$= 53.34 + 33.2 = 86.45$

Conceptually unique input and output parameters are represented by η_2^*

$\eta_2^* = 3$ {x: array holding the integer to be sorted. This is used both as input and output}.

{N: the size of the array to be sorted}.

5. The potential volume $V^* = 5 \log_2 5 = 11.6$

6. Program level $L = V^* / V$

$$= \frac{11.6}{417} = 0.027$$

7. Difficulty Level

D = I / L

$$= \frac{1}{0.027} = 37.03$$

8. Estimated program level

$$\hat{L} = \frac{2}{\eta_1} \times \frac{\eta_2}{N_2} = \frac{2}{14} \times \frac{10}{38} = 0.038$$

9. Effort and Time

$$T = E / \beta = \frac{10974}{18} = 610 \text{ seconds} = 10 \text{ minutes}$$

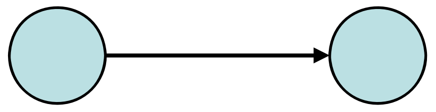
McCabe’s Complexity Measures

McCabe’s Complexity Measures

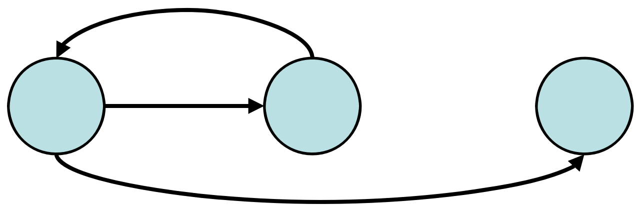
- A software metric used to measure the complexity of software
- Developed by Thomas McCabe
- Described (informally) as the number of simple decision points + 1
- McCabe’s metrics are based on a control flow representation of the program.
- A program graph is used to depict control flow.
- Nodes represent processing tasks (one or more code statements)
- Edges represent control flow between nodes

Flow Graph Notation

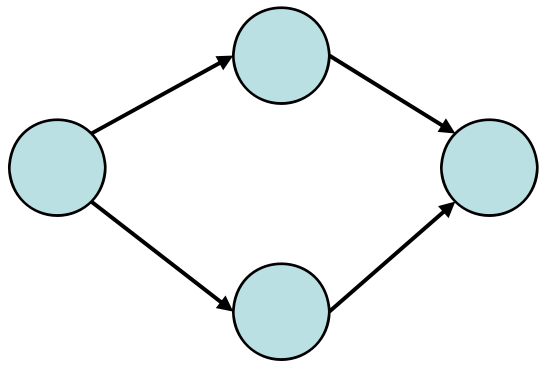
Sequence



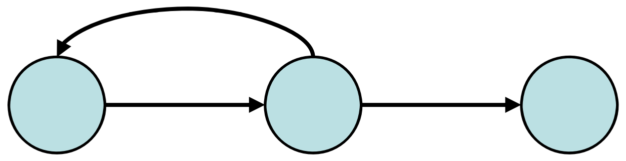
While



If-then-else



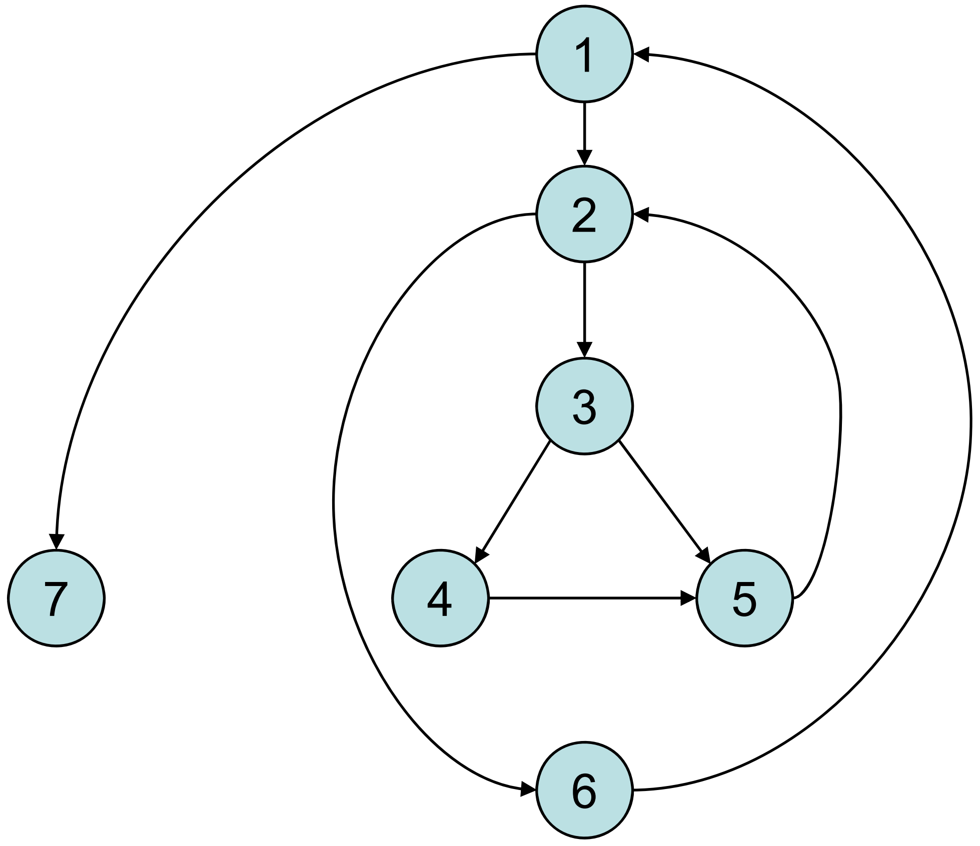
Until



Steps to calculate complexity

- Draw the control flow graph of the code
- Count the number of edges = E
- Count the number of nodes = N
- Count the number of connected components = P
- **Complexity = E – N + 2P**

Example Flow Graph

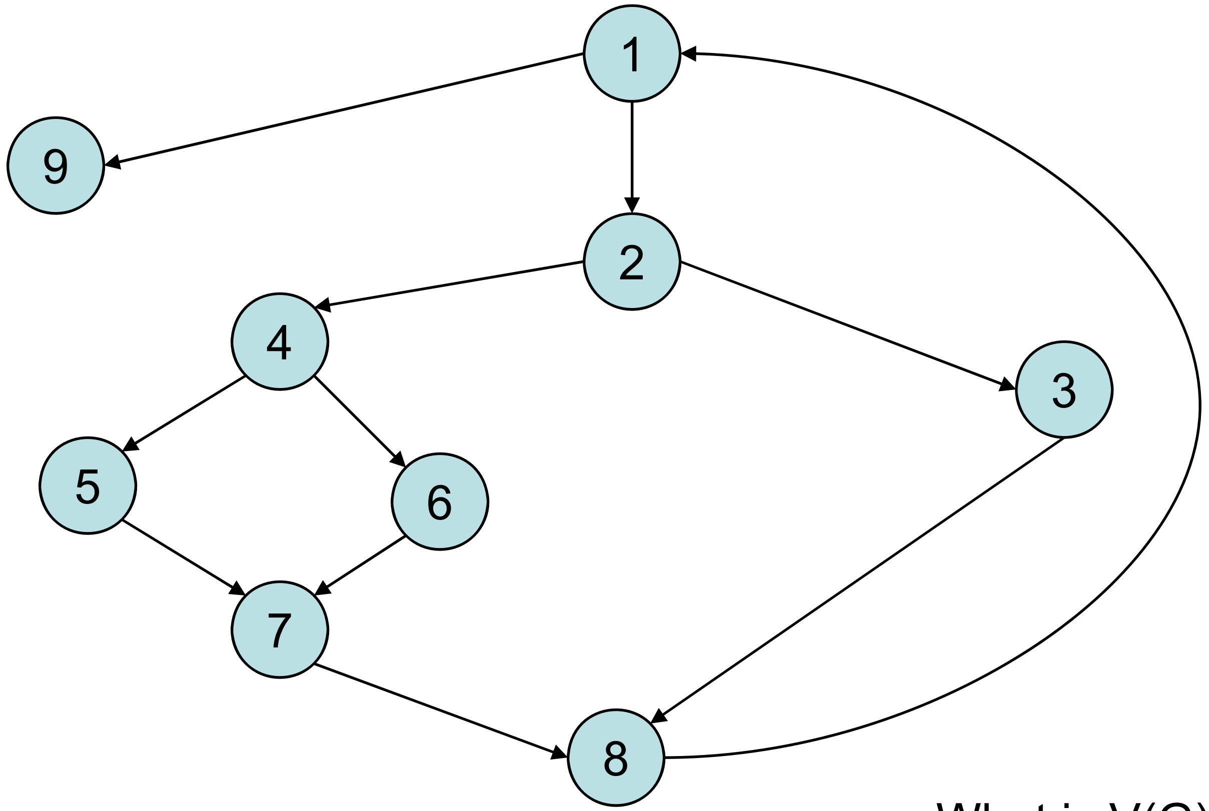

$$\begin{tikzcd}
& & 1 \arrow[d] \arrow[dddll, bend right] & \\
& & 2 \arrow[d] \arrow[dddd, bend right] & \\
& & 3 \arrow[dl] \arrow[dr] & \\
7 & 4 \arrow[rr] & & 5 \arrow[uuul, bend right] \\
& & 6 \arrow[uuuu, bend right] &
\end{tikzcd}$$

Computing V(G)

Complexity = E – N + 2P

- 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

Chidamber and Kemerer’s (CK) Metrics

CK Metrics: Objective

CK metrics were designed:

- To measure unique aspects of the OO approach.
- To measure complexity of the design.
- To improve the development of the software

HOW?

Introduction

- Chidamber and Kemerer (CK) metric suite consists of six metrics
    - weighted methods per class (WMC),
    - coupling between objects (CBO),
    - depth of inheritance tree (DIT),
    - number of children (NOC),
    - response for a class (RFC),
    - lack of cohesion among methods (LCOM).

Weighted Methods per Class

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

- c_i is the complexity of each method, n is the number of methods

- Viewpoints: (of Chidamber and Kemerer)

  -The number of methods and complexity of methods is an indicator of ***how much time and effort is required to develop and maintain*** the object

  -The ***larger the number of methods*** *in an object*, *the greater the potential* ***impact on the children***

  -Objects with ***large number of methods*** are likely to be more application specific, ***limiting the possible reuse***

Depth of Inheritance Tree

- DIT is the maximum length from a node to the root (base class)

- Viewpoints:
- Lower level subclasses inherit a number of methods making behavior harder to predict
- Deeper trees indicate greater design complexity

Number of Children

- NOC is the number of subclasses immediately subordinate to a class

- Viewpoints:
- As *NOC grows*, *reuse increases* - but the abstraction may be diluted

- *Depth is generally better* than breadth in class hierarchy, since it *promotes reuse* of methods through inheritance

- Classes *higher up in the hierarchy* should have *more sub-classes* then those lower down

- NOC gives an idea of the potential *influence a class has on the design*: classes with large number of children may require more testing

Coupling between Classes

- CBO is the number of collaborations between two classes (fan-out of a class C)
  - the number of other classes that are referenced in the class C (a reference to another class, A, is an reference to a method or a data member of class A)
- Viewpoints:
- As collaboration increases reuse decreases
- High fan-outs represent class coupling to other classes/objects and thus are undesirable
- High fan-ins represent good object designs and high level of reuse
- Not possible to maintain high fan-in and low fan outs across the entire system

1. ‘FAN IN’ is simply a count of the number of other Components that can call, or pass control, to Component A.
2. ‘FANOUT’ is the number of Components that are called by Component A.

Response for a Class

- RFC is the number of methods that could be called in response to a message to a class (local + remote)

- Viewpoints:

  As RFC increases

- testing effort increases
- greater the complexity of the object
- harder it is to understand

Example LCOM

- Take class C with M_1, M_2, M_3
- I_1 = {a, b, c, d, e}
- I_2 = {a, b, e}
- I_3 = {x, y, z}
- $P = \{(I_1, I_3), (I_2, I_3)\}$
- $Q = \{(I_1, I_2)\}$

- Thus LCOM = 1

LCOM

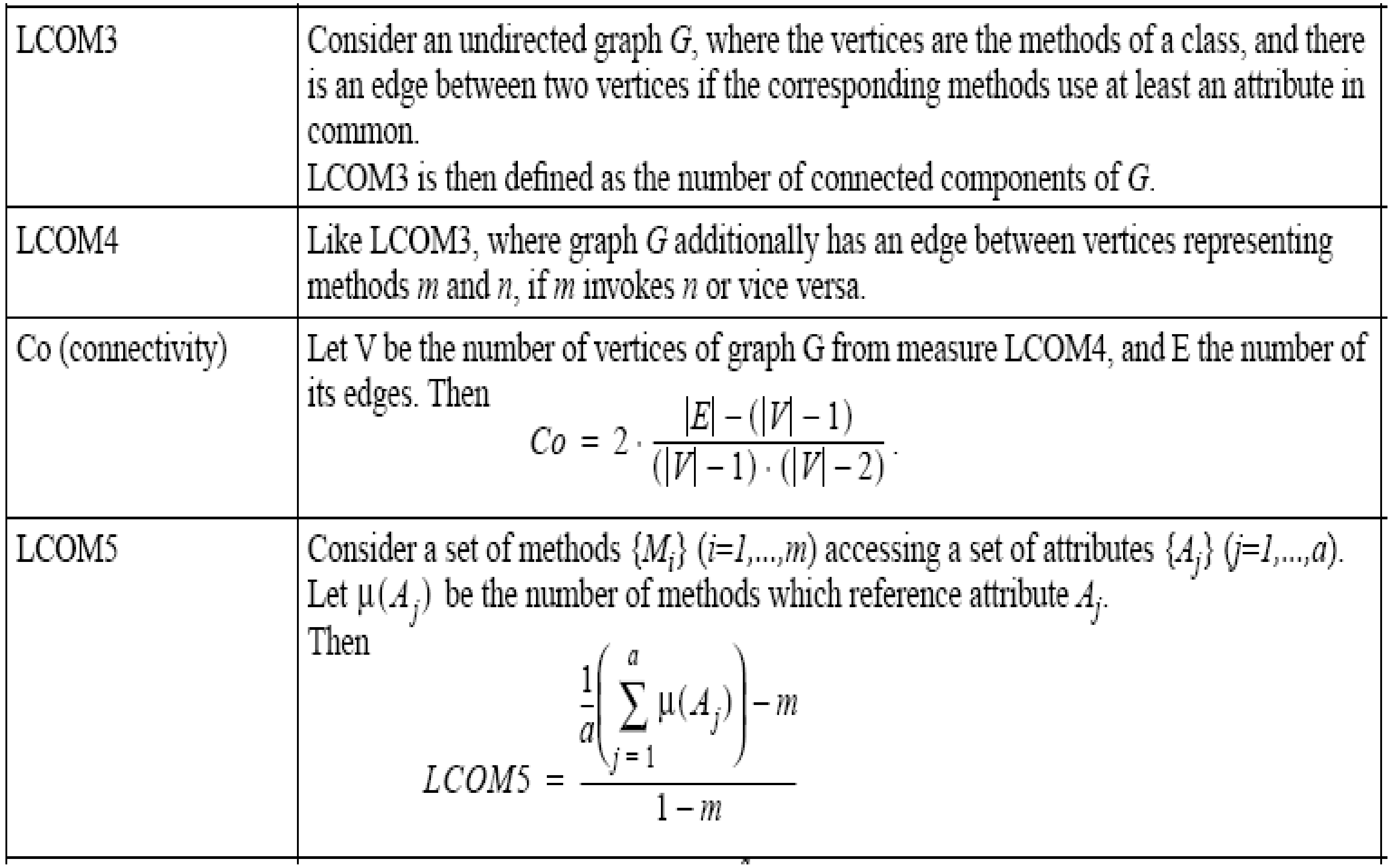
- There are n such sets $I_1, \ldots, I_n$
  - $P = \{(I_i, I_j) \mid (I_i \cap I_j) = \emptyset\}$
  - $Q = \{(I_i, I_j) \mid (I_i \cap I_j) \neq \emptyset\}$
- If all n sets I_i are $\emptyset$ then $P = \emptyset$

- LCOM = $|P| - |Q|$, if $|P| > |Q|$
- LCOM = 0 otherwise

Explanation

- LCOM is the number of empty intersections minus the number of non-empty intersections
- This is a notion of degree of similarity of methods
- If two methods use common instance variables then they are similar
- LCOM of zero is not maximally cohesive
- $|P| = |Q|$ or $|P| < |Q|$

Some other cohesion metrics



LCOM3	Consider an undirected graph *G*, where the vertices are the methods of a class, and there is an edge between two vertices if the corresponding methods use at least an attribute in common. LCOM3 is then defined as the number of connected components of *G*.
LCOM4	Like LCOM3, where graph *G* additionally has an edge between vertices representing methods *m* and *n*, if *m* invokes *n* or vice versa.
Co (connectivity)	Let V be the number of vertices of graph G from measure LCOM4, and E the number of its edges. Then $$Co = 2 \cdot \frac{\lvert E\rvert - (\lvert V\rvert - 1)}{(\lvert V\rvert - 1) \cdot (\lvert V\rvert - 2)}.$$
LCOM5	Consider a set of methods $\{M_i\}$ ($i=1,\ldots,m$) accessing a set of attributes $\{A_j\}$ ($j=1,\ldots,a$). Let $\mu(A_j)$ be the number of methods which reference attribute A_j. Then $$LCOM5 = \frac{\frac{1}{a}\left(\sum_{j=1}^{a} \mu(A_j)\right) - m}{1 - m}$$

Example

Fig. 1

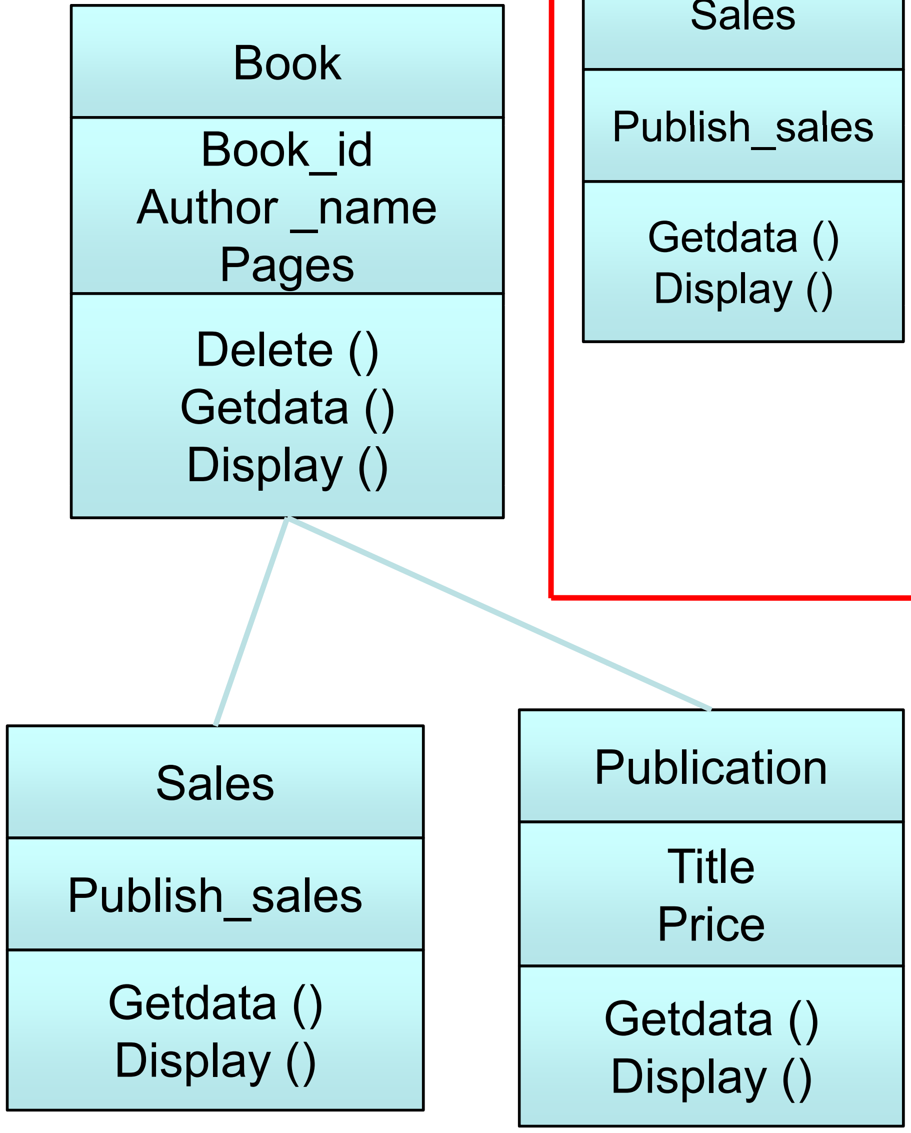


Fig. 2

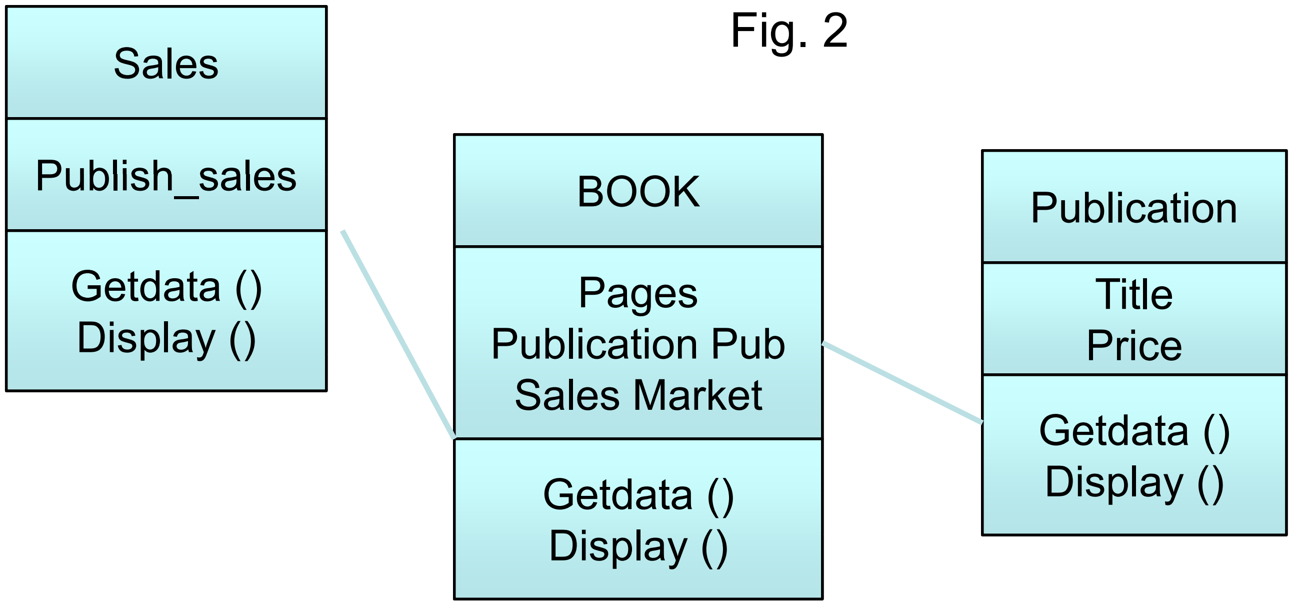
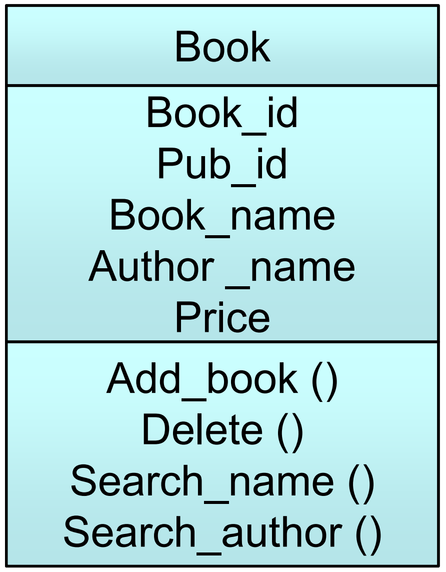


Fig. 3



Compute WMC, RFC, CBO, LCOM. Consider complexity to be 1

1. WMC for book is 3, sale is 2 and publication is 2

- Weighted Number Methods in a Class (WMC)
  - Methods implemented within a class or the sum of the complexities of all methods

2. RFC = 3+2+2 = 7

- Response for a Class (RFC)
  - Number of methods (internal and external) in a class.

3. CBO = 2 (class book) and 0 (class publication and sales)

Coupling between Objects (CBO)

- Number of other classes to which it is coupled.

LCOM

LCOM: Lack of cohesion in methods

– Consider a class C_1 with n methods M_1, M_2…., M_n. Let (I_j) = set of all instance variables used by method M_i. There are n such sets $\{I_1\}$,…….$\{I_n\}$. Let

I_1 {add_book ( )} = { book_id, Pub_id, Book_name, Author_name, Price}
I_2 {delete ( )} = { book_id}
I_3 {search_name ( )} = {Book_name}
I_4 {search_author( )} = {Author_name}

$I_1 \cap I_{2,}\ I_1 \cap I_{3,}\ I_1 \cap I_4$ are non null sets
$I_2 \cap I_{3,}\ I_2 \cap I_{4 \text{ and}}\ I_3 \cap I_4$ are null sets

Thus LCOM = 0, if no of null interactions are not greater than number of non null interactions. Hence, LCOM = 0 [| P| = |Q| =3]

Other OO software Metrics

- Other Size, coupling, cohesion and inheritance metrics are:
- Number of Methods per Class (NOM): 2 for publication class (ref. fig 1)
- Number of Attributes per Class (NOA): 2 for publication class (ref. fig 1)
- Data Abstraction Coupling(DAC)
  - Number of Abstract Data Types in a class.
  - Referring to fig 3: Class Book has two ADT's: Pub and market. Hence DAC is 2 for book class.
- Message Passing Coupling (MPC)
  - Number of send statements defined in a class.
  - Ref to fig 3: MRC is 4 for class book as 4 methods in class book calls sales::getdata (), sales::display (), publication::getdata(), publication::display().

Method Inheritance Factor

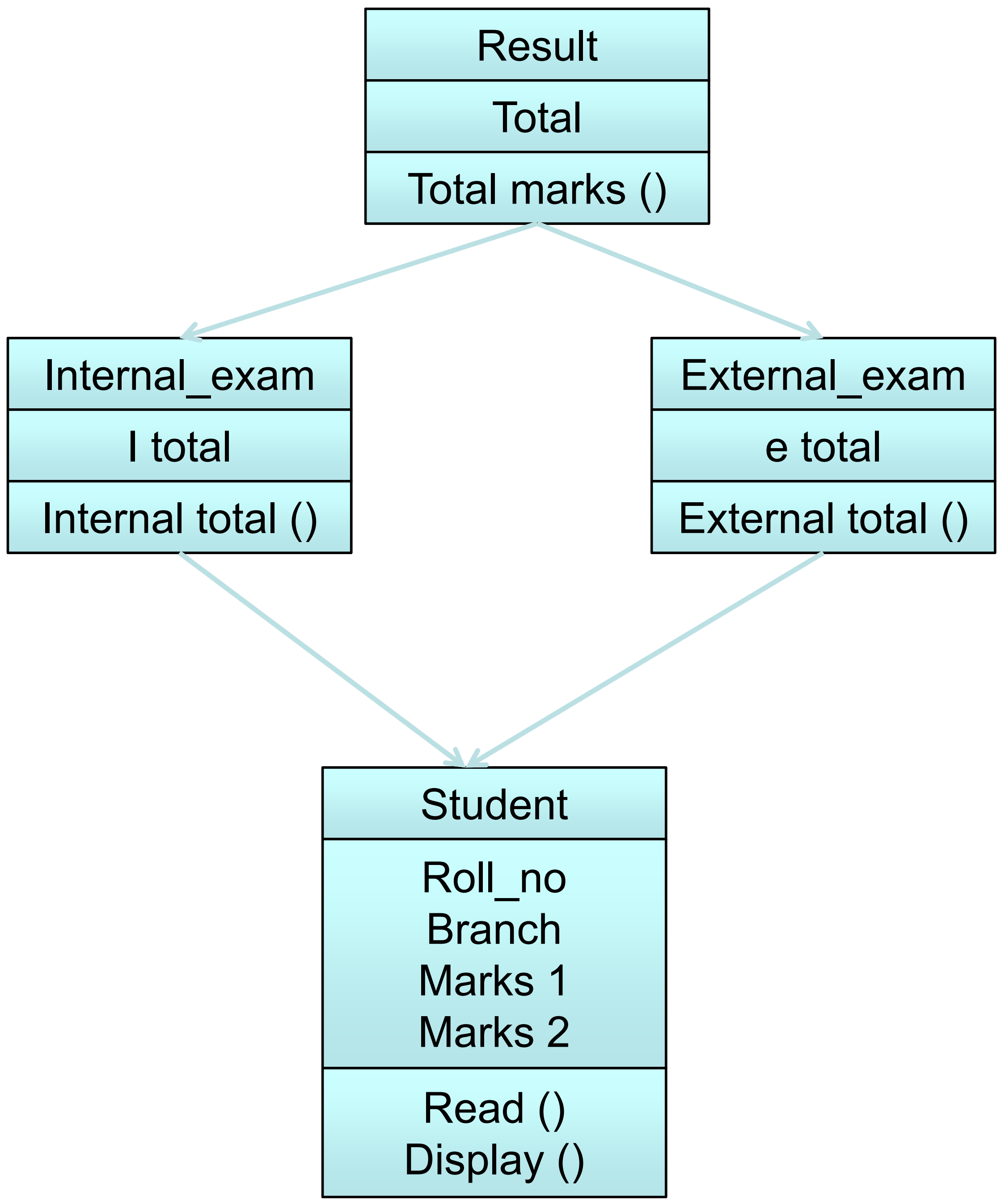
$$\text{MIF} = \frac{\sum_{i=1}^{n} M_i(C_i)}{\sum_{i=1}^{n} M_a(C_i)} \quad .$$

- $M_i(C_i)$ 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
- n is the total number of classes

MIF

- $M_a(C_i) = M_d(C_i) + M_i(C_i)$
- All that can be invoked = new or overloaded + things inherited

- MIF is [0,1]
- MIF near 1 means little specialization
- MIF near 0 means large change



Private attributes:
Roll_no
Branch

Protected Attributes:
Marks 1
Marks 2

$n = 4$

Let,

C_1 =Student class,
C_2 =Internal exam class
C_3= External exam class
C_4 = Result class

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

$$= \frac{M_i(C_1) + M_i(C_2) + M_i(C_3) + M_i(C_4)}{M_i(C_1) + M_i(C_2) + M_i(C_3) + M_i(C_4) + M_d(C_1) + M_d(C_2) + M_d(C_3) + M_d(C_4)}$$

Where, $M_i(C_1)$ = number of inherited methods in class student =0

$M_i(C_2)$ = number of inherited methods in class internal exam=2

MIF = 0+2+2+2 / 11 = 6/11

Coupling Factor

$$CF = \frac{\sum_i \sum_j is\_client(C_i, C_j)}{(TC^2 - TC)} \quad .$$

- is_client(x,y) = 1 iff a relationship exists between the client class and the server class. 0 otherwise
- (TC^2-TC) is the total number of relationships possible
- CF is [0,1] with 1 meaning high coupling

Polymorphism Factor

$$\text{PF} = \frac{\sum_i M_o(C_i)}{\sum_i \left[M_n(C_i) * DC(C_i)\right]}.$$

- $M_n()$ is the number of new methods
- $M_o()$ is the number of overriding methods
- DC() number of descendent classes of a base class
- The number of methods that redefines inherited methods, divided by maximum number of possible distinct polymorphic situations

Metric Tools

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

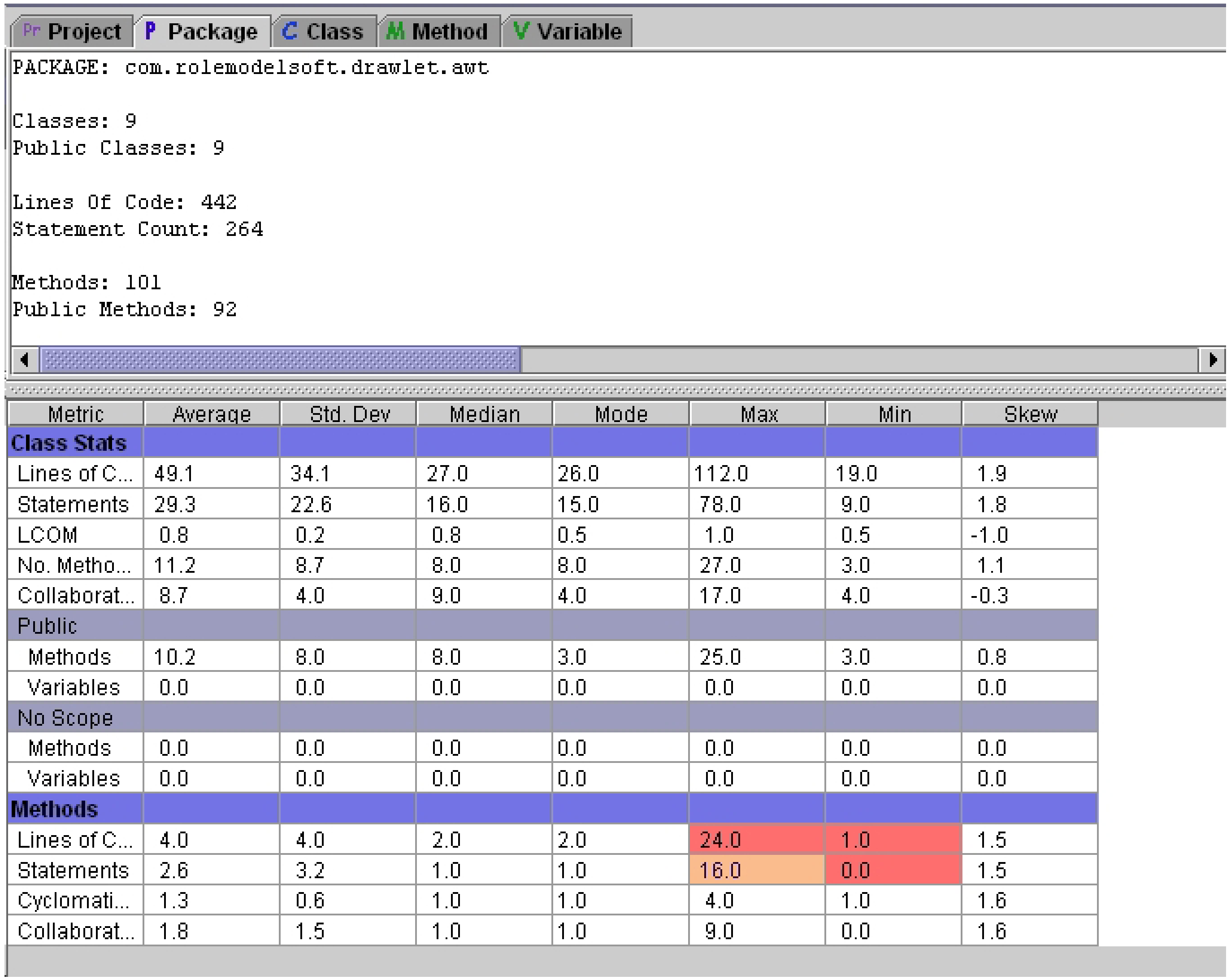
- OO metric calculation tool for Java code (by Cain and Vasa for a project at COTAR, Australia)

- 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



```
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/

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/