Measuring Object Oriented Design

Rushikesh K Joshi

Department of Computer Science and Engineering
IIT Bombay

Tutorial at SoDA Workshop
Jan 23, 2004
at Taj Westend, Bangalore
Measurement in day-to-day activities

- What’s the temperature today?
- How much time did it take to travel?
- Was it a comfortable journey?
- Are you joking?
- How cold is it at Bangalore?
- How many participants in SoDA?
- Was the morning tea too sweet?
- How popular is soda amongst the students as compared to tea?
- Which is the best team in the world?
Measurement

• Fundamental in any engineering discipline
• Software Engineering is no exception

• If you can measure, you can predict
• If you can measure, you can control
• If you can measure, you can compare
• If you can measure, you can improve
• You can estimate, cost, plan, investigate, assess …
• And if you can measure, you can relax too!
Quality

• Is there a definition for quality?

• How do you spot it?

• How do you measure it?

• Quantification of Quality
  – Quantification process is subjective
  – Once quantified, the measure is objective
Some example uses of metrics

- Selecting Data structures and algorithms
- Choosing over alternative implementations
- Adjust/refactor designs based on coupling and cohesion
- Use defect rates for process improvement
- Use effort and productivity metrics to assess the use of a technology or tool
- For improving understandability, maintainability, testability of software
Some basics of measurement theory

- How do we understand a given quality (or attribute)?
- By comparing samples
  - Example: height
    - We say person A is taller than person B.
    - Person C’s wife is not that much taller than him.
  - We understand the taller than relation through comparison
  - We do not use numbers
- Taller than is an empirical relation for attribute height and not a numerical relation
- When there is difference of opinion, we take consensus
- Empirical relations on a set need not be binary
  - is tall
- We can map the empirical world to numerical world or formal relational world \( \rightarrow \) measurement
- E.g. > relation on height in cms
Measurement: Quantifying Quality

• Are there any restrictions to be followed when this mapping is done?

• The mapping must be such that the numerical relation preserves and is preserved by the empirical relation

• i.e. as in
  – A is taller than B if and only if H(A) > H(B)
  – A is tall if and only if H(A) > 70
  – A is much taller than B if and only if H(A) > H(B)+15
Quantification

• Define following mappings from empirical to numerical world
  – A is taller than B if and only if $H(A) > H(B)$
  – A is tall if and only if $H(A) > 70$
  – A is much taller than B if and only if $H(A) > H(B) + 15$

• Now given
  • sticks A, B, C and
  • $H(A) = 84$ $H(B) = 72$ $H(C) = 42$

  We can say
  – A is taller than B and $H(A) > H(B)$

And so on

• The interpretation
Different relational properties

- Reflexive: $aRa$ for all $a$’s in the set
- Irreflexive: not $aRa$ for all $a$’s
- Non-reflexive: It is not reflexive
- Symmetric: $aRb \rightarrow bRa$ for all $a$’s and $b$’s
- Asymmetric: $aRb \rightarrow \text{not } bRa$
- Antisymmetric: $aRb$ and $bRa \rightarrow a=b$
- Transitive: $aRb$ and $bRc \rightarrow aRc$
Measurement scales

- Nominal
  - Specs fault, design fault, coding fault
- Ordinal
  - Trivial, simple, moderate, complex, incomprehensible
- Interval
  - Temperature Celsius/Fahrenheit, relative time
- Ratio
  - Length, weight, time intervals, temperature Kelvin
- Absolute
  - counts
Nominal scale

- Empirical system consists of only different classes
- There is no ordering among the classes
  - Numbering of classes is okay
  - but that is only to distinguish them and no notion of magnitude is associate

- i.e. classes are not ordered, and even if they are numbered from 1 to n, that is only for identification
- Civil engg students 001, cse students 002, Mech students 003
Ordinal Scale

- Empirical system consists of classes
- Classes are ordered with respect to the attribute
  - Any mapping that preserves the ordering is acceptable
  - The number represents ranking only
  - Hence no functions such as addition, subtraction

Good, very good, excellent, exceptional
Interval Scale

• It is more powerful than nominal or ordinal
• Captures the size of the intervals that separate classes
  – Preserves orders as with ordinal
  – Preserves differences but not ratios
    • Addition, subtraction is accepted
    • Multiplication and division is not

Temperature change: 20°C to 21°C at Mumbai
  is same as 30°C to 31°C in Chennai
But we cannot say it is 2/3rd as hot in Mumbai as in Chennai
We cannot say Delhi is 50% hotter than Mumbai
Ratio Scale

- Preserves ordering
- Preserves the size of intervals between entities
- And ratios between entities
- Has a 0 element representing lack of the attribute
- Measurement mapping starts at 0 and increases in equal units (intervals)
- All arithmetic can be performed
  - Length of an object
Absolute scale

• Measurement is made by counting the number of elements in the set
  – There is only one possible measurement mapping, i.e. the count

  – All arithmetic is useful

  • Number of project engineers
Measurement in software (Product)

• Measuring specification
  – Internal attributes: size, reuse, modularity, redundancy, functionality, syntactic correctness
  – External attributes: comprehensibility, maintainability
Measurement in software (Product)

• Measuring designs
  – Internal attributes: size, reuse, modularity, coupling, cohesiveness, functionality
  – External attributes: quality, complexity, maintainability
Measurement in software (Product)

• Measuring code
  – Internal attributes: size, reuse, modularity, coupling, structured-ness
  – External attributes: reliability, usability, maintainability
Measurement in software (Processes)

• Measuring process stages
  – Time, efforts, number of changes made, number of faults
  – Cost, stability, cost-effectiveness
Measurement in software (Resources)

• Measuring personnel, s/w, hardware, offices
  – Size, price, temperature level, light in office, speed, memory size
  – Productivity, experience, usability, utilization, availability
Measuring size - length

• LoC
  – But explain how blank lines, comments, data declarations, lines containing more language statements are handled

• NCLoC
  – Non commented lines (or ELoC-effective LoC)
  – But for storage requirements this may be needed

• Or use LOC=NCLoc+CLOC (non-commented + commented lines of code measured separately)

• Ratio CLOC/LOC : density of comments
Measuring size - length

• ES: number of executable statements
  • Separate statements on same line are still distinct
  • Ignores comments, declarations, headings

• DSI: number of delivered source instructions
  • Like ES, but includes data definitions, and headings
Measuring size - length

- **Halstead’s ideas**
- Given program P:
  - $u_1$ no of unique operators
  - $u_2$ no of unique operands
  - $n_1$ total occurrences of operators
  - $n_2$ total occurrences of operands
- $n = \text{Length of } P = n_1 + n_2$
- $u = \text{Vocabulary of } P = u_1 + u_2$
- $v = \text{Volume of } P = n \times \log_2 u$
  - $v$ = number of mental comparisons needed to write a program of length $n$
- Potential volume $v^* = \text{volume of minimum size implementation of } P$
- $L = \text{Program level of } P = v^*/v$
- $D = \text{Difficulty level of } P = i/L$
Weyuker’s axioms for software complexity measures

- P, Q, R program bodies
- |P| complexity of P wrt some hypothetical measure
- |P| is non-negative
- For any P,Q
  - |P|<=|P| or |Q|<=|P|
- Complexities can be compared and ordered
Weyuker’s axiom 1

- There exists p and q such that
  - $|P|$ is not equal to $|P|$

- Tries to stress that a measure in which all programs are equally complex is not really a measure
Weyuker’s axiom 2

• Let $c$ be a nonnegative number

• There are only finitely many programs of complexity equal to $c$

• This says that measure is not sensitive enough if it divides all programs into just a few complexity classes
Weyuker’s axiom 3

- There are distinct programs P and Q such that $|P| = |Q|$

- i.e. the measure should not assign unique value to every program and thus should not be too fine level
Weyuker’s axiom 4

• There exist programs $p$ and $q$ such that

  They are equivalent but $|P| \neq |Q|$

• i.e. even when 2 programs do the same thing, their implementation complexity varies
Weyuker’s axiom 5

- For all Ps and Qs,
  - $|P| \leq |P;Q|$ and
  - $|Q|$ not= $|P;Q|$ 

- Components of the program are no more complex than program itself
Weyuker’s axiom 6

• Whether or not the concatenation of a given program body with another should always affect the complexity of the resultant program in a uniform way?

• There exist p,q,r such that
  – Complexity of p and q are same: $|P| = |Q|$
  – Complexity of $P;R$ not same as complexity of $Q;R$
    • $|P;R| \neq |Q;R|$
  – i.e. $R$ may not interact same with $p$ and $q$
Weyuker’s axiom 7

• There are two program bodies P and Q such that
  – Q is formed by permuting the order of statements of P and |P| not= |Q|

• Program complexity should be responsive to order of statements, and hence interaction among statements
Weyuker’s axiom 8

• If P is a renaming of Q then $|P| = |Q|$

• This is in terms of Psychological complexity (actually relabeling of variables)
Weyuker’s axiom 9

- There exist P, Q such that

  \[ -|P| + |Q| < |P;Q| \]

- Complexity of a program formed by concatenating 2 program bodies can be greater than sum of their individual complexities
Weyuker’s axioms have been criticized

- E.g. consider KNOT measure = total no of points at which control flow crosses

- It is 0 for all structured programs and it does measure unstructuredness of programs

- But property 1 states that every program should not have the same value else it is not a metric
Some Criticism on Weyuker’s axioms

• Property 5 asserts that adding code cannot decrease complexity.
• This reflects a view that program size is key factor in complexity
• And also that low comprehensibility is not a key factor
• It’s widely believed that we understand a program more easily as we see more of it
• Whereas, axiom 6 has to do with comprehensibility and little to do with size
• Thus they cannot be both satisfied by a single measure
• Zuse concluded 5 needs ratio scale, and 6 excludes it
• Useless metrics may be created satisfying all the properties
Object Oriented Metrics

• Why do we need them?

• In non-OO software complexity is in structure of code itself, larger portion of code is imperative.

• In OO code, complexity lies in interaction between objects, a large portion of code is declarative, OO models real life objects: classes, objects, inheritance, encapsulation, message passing.
CK Metric suit

- Chidamber and Kemerer’s suit for object oriented systems
  - Weighed methods per class (WMC)
  - Depth of inheritance tree (DIT)
  - Number of children (NOC)
  - Coupling between object classes (CBO)
  - Response for a class (RFC)
  - Lack of cohesion in methods (LCOM)
Weighted methods per class (WMC) metric

- Every class has methods $M_1 \ldots M_n$ defined in the class
- $C_1 \ldots C_n$ are complexities of methods

Then

$$WMC = \text{sum of all } C_i \text{'s from } C_1 \text{ to } C_n$$
Weighted methods per class (WMC) metric

- Method complexity is left undefined
- Scale used for it must be at least interval scale so that summation is possible
Weighted methods per class (WMC) metric

• **Viewpoints**

• No. of methods and complexity of methods is a predictor for time and efforts for a class

• Larger the no of methods, greater the impact on subclasses

• Classes with large no. of methods will have less reuse and will be application specific
Depth of inheritance tree (DIT)

- It’s a metric for a class
- Maximum length from node to the root of the tree
- May be in presence of multiple inheritance
- Measures how many ancestor classes can potentially affect this class
Depth of inheritance tree (DIT)

• **Viewpoints**

• The deeper the class, greater the number of methods it will have, making it more complex

• Deeper tree indicates more design complexity as more classes and methods are involved
Number of Children (NOC)

• Number of immediate subclasses
• Measures how many subclasses will inherit the parent class

• Viewpoint
  – Greater the no. of children, greater the reuse
  – Greater the no. of children, greater the possibility of improper abstraction of the class: it could be a misuse of subclassing
  – No. of children measure efforts needed on testing a class
Coupling between objects (CBO)

- Count of no. of other classes to which it is coupled
- Objects are coupled if one of them acts on the other

**Viewpoint**
- Excessive coupling shows decline in modularity
- More independent a class is, easier it is to reuse
- Less coupling promotes modularity and encapsulation
- Indicates how complex the testing could be
Coupling between objects (CBO)

• Classes responsible for managing interfaces have a high CBO
  – Classes that connect subsystems

• Usable by senior managers and project managers
  – to track integrity of a system
  – to check whether components are developing unnecessary interconnections
Response for a class (RFC)

• RFC = |Response Set RS|

• RS is a set of methods that can potentially be executed in response to a message received by an object of that class
Response for a class (RFC)

- Viewpoint

- Larger the number of methods invoked from a class, greater the complexity

- Worst case RS values will assist in testing estimation

- Large RFC needs greater understanding by the tester and in debugging
Lack of cohesion in methods (LCOM)

- Consider class C with methods M1..Mn
- Let Vi be set of instance variables used by method Mi
- There are such n sets Vi...Vn
- Let Pi’s be set of all tuples (Vi, Vj) such that the intersections of Vi and Vj are null
- Let Qi’s be set of all tuples (Vi, Vj) such that the intersections of Vi and Vj are non-null
- LCOM = |P| - |Q| if |P| > |Q|
  - 0 otherwise
Lack of cohesion in methods (LCOM)

- The degree of similarity is between 2 methods is given by the interaction of $V_i$ and $V_j$
- LCOM is count of no. of method pairs whose similarity measure is 0, minus the count of no of method pairs whose similarity measure is not 0.
- The larger the no of similar methods, more cohesive is the class.
- If none of the methods use any instance variable, they will have no similarity and LCOM value will be 0.
Lack of cohesion in methods (LCOM)

- **Viewpoint**
- Cohesiveness of methods within a class is desirable. It promotes encapsulation
- Lack of cohesion implies classes need splitting or splitting into subclasses
- Design flaws may be detected
- Low cohesion increases complexity and errors
- Can be used to identify classes that are trying to achieve many different objectives
Schroeder’s compilation of metrics

• Categories of metrics
  – System size
    • E.g. how many function calls and objects?
  – Class or method size
    • Small classes typically better than large ones
  – Coupling and inheritance
    • Number of types of relations: interdependence
  – Class of method internals
    • How complex code of a class is
System Size

- Lines of code  LOC
- Total function calls  TFC
- Number of classes  NOC
- Number of windows  NOW
  - (size of user interfaces on the system)
Class/Method Size

- LOC and function calls per class/method
- Number of methods per class
- Public method count per class
- Number of attributes per class
- Number of instance attributes per class
Coupling and Inheritance

- **Class fan-in**
  - Number of classes that depend on a given class
- **Class fan-out**
  - Number of classes on which a class depends
- **Class inheritance level: no. of direct ancestors**
- **Number of children per class**
Class and method internals

• No. of global/shared references per class
  – Break encapsulation
    • Use sparingly if unavoidable

• Method complexity
  – No of different execution paths within a block of code (cyclomatic complexity)

• Number of public attributes per class

• Lack of cohesion among methods
Class and method internals

- Class specialization index
  - Extent to which subclasses override (replace) the behavior of their ancestor classes
  - More the specialization, abstraction may be said to be inappropriate
  - Extending class behavior with new methods vs. heavy overriding

- Percent of commented methods
  - Documentation

- Number of parameters per method
  - Higher the number, complex the interface
MOOD Metrics

- Method hiding factor (MHF)
- Attribute hiding factor (AHF)
- Method inheritance factor (MIF)
- Polymorphism factor (PF)
- Coupling factor (CF)
Method hiding factor

• Invisibility of a method = percentage of total classes from which the method is not visible

• Numerator: sum of invisibilities of all methods in all classes
• Denominator: total number of methods defined in the system under consideration
• Very low: insufficient abstraction
• High value: little functionality
Attribute factor

- Numerator: sum of invisibilities of all attributes in all classes
- Invisibility: percentage of total classes from which attribute is not visible
- Denominator: total number of attributes defined in the system under consideration
- Very low: inefficient design
- Ideally all attributes are hidden
Polymorphism factor

• Actual number of possible different polymorphic situations
  
• Numerator: actual amount of polymorphism
  
• Denominator: maximum attainable polymorphism
Back to basic 2 issues

• Basic properties of measurement

• How to quantify quality