Measuring Internal Product Attributes

Size Attributes

- Size measures only indicate how much of an entity we have.
- Size alone cannot directly indicate external attributes such as effort, productivity, and cost
- Rejecting simple size measures because they do not indicate attributes like coding difficulty reflects a misunderstanding

Properties of Valid Software Size Measurement

- Three properties for any valid measure of software size:
  - Nonnegativity: All systems have nonnegative size.
  - Null value: The size of a system with no elements is zero.
  - Additivity: The size of the union of two modules is the sum of sizes of the intersection.
- The two modules after subtracting the size of the intersection.

Counting Lines of Code (LOC) to Measure Code Size

- We must explain how each of the following is handled:
  - Blank lines
  - Comment lines
  - Data declarations
  - Lines that contain several separate instructions
- Jones reports that one count can be as much as five times larger than another, simply because of the difference in counting technique

LOC Definitions

- NCLOC: Non Commented Lines of Code (Comments and blank lines removed)
- CLOC: Number of comment lines of program text (CLOC)
- Total size (LOC) = NCLOC + CLOC
- Density of comments = CLOC / LOC
- Is NCLOC valid measure?
- Number of executable statements (ES) : counts separate statements on the same physical line as distinct. It ignores comment lines, data declarations, and headings

Halstead's Approach

Halstead's software science attempted to capture attributes of a program that paralleled physical and psychological measurements in other disciplines. He began by defining a program *P* as a collection of tokens, classified as either operators or operands. The basic metrics for these tokens are the following:

μ_1 = Number of unique operators
μ_2 = Number of unique operands
N_1 = Total occurrences of operators
N_2 = Total occurrences of operands

Halstead's Approach

For example, the FORTRAN statement

$$A(I) = A(J)$$

has one operator (=) and two operands ($A(I)$ and $A(J)$).

The *length* of P is defined to be $N = N_1 + N_2$, while the *vocabulary* of P is $\mu_1 = \mu_2 + \mu_3$. The *volume* of a program, akin to the number of mental comparisons needed to write a program of length N or the minimum number of bits to represent a program, is

$$V = N \times \log_2 \mu$$

Halstead derived measures for a number of other attributes including *program level*, *difficulty*, and *effort*. According to Halstead, one can use these metrics and an estimate of the number of mental discriminations per second to compute the time required to write a program.

Halstead’s Approach

- What is the meaning of attributes such as volume, difficulty, or program level?
- Does it contradict software size validity properties?

Alternative Size Measures

If α is the average number of characters per line of program text, then we have the rescaling

$$\text{CHAR} = \alpha \, \text{LOC}$$

which expresses a stochastic relationship between LOC and CHAR. Similarly, we can use any constant multiple of the proposed measures as an alternative valid size measure. Thus, we can use KLOC (thousands of LOCs) or KDSI (thousands of delivered source instructions) to measure program size.

Design Size

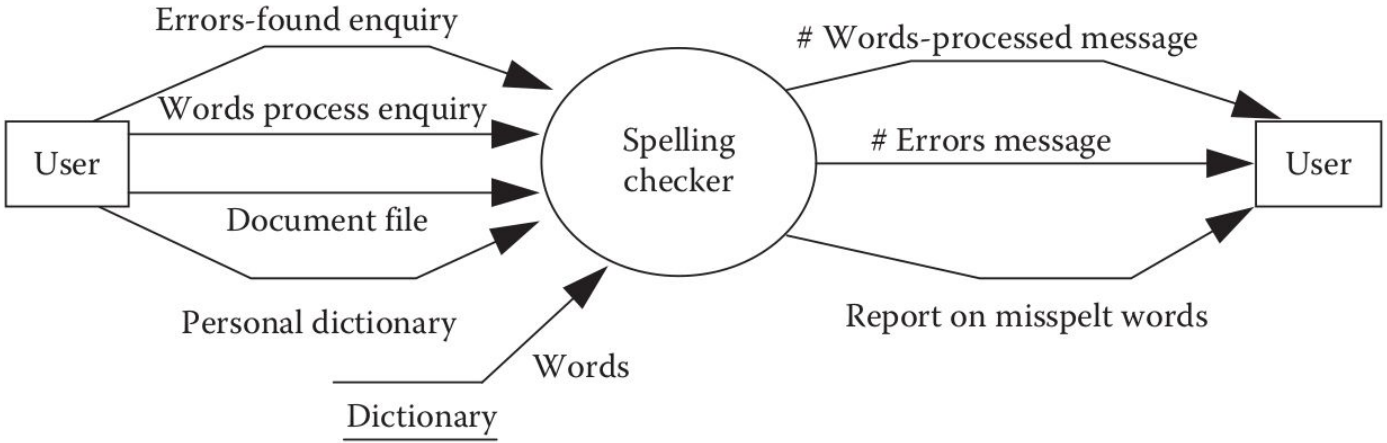
- Size in terms of packages, design patterns, classes, interfaces, abstract classes, operations, and methods.
- Packages: Number of subpackages, number of classes, interfaces (Java), or abstract classes (C++)
- Design patterns:
  - Number of different design patterns used in a design
  - Number of design pattern realizations for each pattern type
  - Number of classes, interfaces, or abstract classes that play roles in each pattern realization
- Classes, interfaces, or abstract classes: Number of public methods or operations, number of attributes
- Methods or operations: Number of parameters, number of overloaded versions of a method or operation

Design Size

- Weighted Methods per Class (WMC) measure (Chidamber and Kemerer suite): measured by summing the weights of the methods in a class, where weights are unspecified complexity factors for each method
- Both the number of methods and the number of attributes can serve as class size measures
- One set of studies found that the number of methods is a better predictor of class change-proneness than the number of attributes

Function Points

Spell-checker spec: The checker accepts as input a document file and an optional personal dictionary file. The checker lists all words not contained in either of these files. The user can query the number of words processed and the number of spelling errors found at any stage during processing.



A = # external inputs = 2, B = # external outputs = 3, C = # inquiries = 2,
D = # external files = 2, and E = # internal files = 1

Function Points

TABLE 8.2 Function Point Complexity Weights

Item	Simple	Weighting Factor Average	Complex
External inputs	3	4	6
External outputs	4	5	7
External inquiries	3	4	6
External files	7	10	15
Internal files	5	7	10

Function Points

Consider the spelling checker introduced in Example 8.11. If we assume that the complexity for each item is *average*, then the UFC is

$$UFC = 4A + 5B + 4C + 10D + 7E = 58$$

If instead we learn that the dictionary file and the misspelled word report are considered *complex*, then

$$UFC = 4A + (5\times 2 + 7\times 1) + 4C + 10D + 10E = 63$$

Function Points

To continue our FP computation for the spelling checker in Example 8.11, we evaluate the technical complexity factor. After having read the specification in Figure 8.2, it seems reasonable to assume that F_3, F_5, F_9, F_{11}, F_{12}, and F_{13} are 0, that F_1, F_2, F_6, F_7, F_8, and F_{14} are 3, and that F_4 and F_{10} are 5. Thus, we calculate the TCF as

$$\text{TCF} = 0.65 + 0.01(18 + 10) = 0.93$$

Since UFC is 63, then

$$\text{FP} = 63 \times 0.93 = 59$$

Function Points Disadvantages

- Subjectivity
- Double Counting
- Counter-intuitive values
- Problems with accuracy
- Problems with changing requirements
- Problems with measurement theory

Structural Complexity Properties

- Nonnegativity: System complexity cannot be negative.
- Null value: The complexity of a system with no links is zero.
- Symmetry: The complexity of a system does not depend on how links are represented.
- Module monotonicity: System complexity “is no less than the sum of the complexities of any two of its modules with no relationships in common.”
- Disjoint module additivity: The complexity of a system of disjoint modules is the sum of the complexities of the modules.

Structural Complexity Properties Example

- Nonnegativity: System complexity cannot be negative.
- Null value: The complexity of a system with no links is zero.
- Symmetry: The complexity of a system does not depend on how links are represented.
- Module monotonicity: System complexity “is no less than the sum of the complexities of any two of its modules with no relationships in common.”
- Disjoint module additivity: The complexity of a system of disjoint modules is the sum of the complexities of the modules.

- Coupling?
- Cohesion?

OO Structured Attributes and Measures : Coupling

The *coupling between object classes* (*CBO*) is metric 4 in a commonly referenced suite of object-oriented metrics (Chidamber and Kemerer 1994). CBO is defined to be the number of other classes to which the class of interest is coupled. Briand et al. identified several problems with CBO (Briand et al. 1999b). One problem is that the treatment of inherited methods is ambiguous. Another problem is that CBO does not satisfy property 5, disjoint module additivity, one of the coupling properties given in Section 9.1.3. If class A is coupled to classes B and C, both classes B and C have CBO = 1, assuming no other coupling connections. Now assume that classes B and C are merged creating new class D. Class D will have CBO = 1, as it is only coupled to class A. Property 5 is not satisfied because the original classes A and B were disjoint, and the property requires that the coupling of the merged classes be the sum of their coupling. Property 5 would be satisfied if the measure counted the number of connections rather than the number of classes that the class of interest is coupled to.

OO Structured Attributes and Measures : Coupling

Chidamber and Kemerer defined another coupling measure, *response for class* (*RFC*), which is metric 5 in their suite of metrics (Chidamber and Kemerer 1994). RFC captures the size of the *response* set of a class. The response set of a class consists of all the methods called by local methods. RFC is the number of local methods plus the number of external methods called by local methods. Consider again the merging of classes B and C that use methods in class A, and assume that B and C are disjoint—they do not call each other's methods. The RFC of the merged class D will not be the sum of the RFC of B and C if classes B and C used one or more of the same methods in class A. If RFC was measured by counting the number of uses of external methods rather than counting the number of methods invoked, property 5 would be satisfied.

OO Structured Attributes and Measures : Coupling

Message passing coupling (*MPC*) was introduced by Li and Henry and formalized by Briand, Daly, and Wüst (Li and Henry 1993, Briand et al. 1999b). The MPC value of a class is a count of the number of static invocations (call statements) of methods that are external to the class. MPC satisfies all of the properties in Section 9.1.3 including property 5.

OO Structured Attributes and Measures : Coupling

Robert C. Martin defines two package-level coupling measures (Martin 2003). These indicate the coupling of a package to classes in other packages:

1. Afferent coupling (C_a): "The number of classes from other packages that depends on the classes within the subject package." Only "class relationships" are counted, "such as inheritance and association." C_a is really the *fan-out* (see Section 9.3.8) of a package.
2. Efferent coupling (C_e): "The number of classes in other packages that the classes in the subject package depend on" via class relationships. C_e is a form of the *fan-in* of a package.

These measures satisfy all of the properties in Section 9.1.3, including property 5 as long as a class can only belong to a single package.

Martin suggests that high efferent coupling makes a package *unstable* as it depends on too many imported classes. He defines the following *instability* (I) *metric:*

$$I = \frac{C_e}{C_a + C_e}$$

OO Structured Attributes and Measures : Cohesion

Lack of cohesion metric (*LCOM*) is metric 6 in the Chidamber and Kemerer suite of metrics (Chidamber and Kemerer 1994). Here, the cohesion of a class is characterized by how closely the local methods are related to the local instance variables in the class. LCOM is defined as the number of disjoint (i.e., nonintersecting) sets of local methods. Two methods in a class intersect if they reference or modify common local instance variables. LCOM is an inverse cohesion measure; higher values imply lower cohesion. Briand, Daly, and Wüst found that LCOM violates property 1 of Section 9.1.4, as it is not normalized (Briand et al. 1998). Since LCOM indicates inverse cohesion, properties 2 through 4 are also not satisfied.

OO Structured Attributes and Measures : Cohesion

Tight class cohesion (*TCC*) and *loose class cohesion* (*LCC*) are based on connections between methods through instance variables (Bieman and Kang 1995). Two or more methods have a *direct connection* if they read or write to the same instance variable. Methods may also have an *indirect connection* if one method uses one or more instance variables directly and the other uses the instance variable indirectly by calling another method that uses the same instance variable. *TCC* is based on the relative number of direct connections:

$$TCC(C) = NDC(C)/NP(C)$$

OO Structured Attributes and Measures : Cohesion

where $NDC(C)$ is the number of direct connections in class C and $NP(C)$ is the maximum number of possible connections. *LCC* is based on the relative number of direct and indirect connections:

$$LCC(C) = (NDC(C) + NIC(C))/NP(C)$$

where $NIC(C)$ is the number of indirect connections. The measures do not include constructor and destructor methods in the computation, since they tend to initialize and free all instance variables and will thus artificially increase the measured cohesion. Both *TCC* and *LCC* satisfy all four cohesion properties in Section 9.1.4.

OO Length Metric

- The depth of inheritance tree (DIT) is metric 3 in the Chidamber and Kemerer suite of object-oriented metrics (Chidamber and Kemerer 1994)
- Inheritance in a class diagram is represented as a hierarchy or tree of classes.
- The nodes in the tree represent classes, and for each such class, the DIT metric is the length of the maximum path from the node to the root of the tree
- Chidamber and Kemerer claim that DIT is a measure of how many ancestor classes can potentially affect this class