Inheritance Metrics: What do they Measure?

G. Sri Krishna and Rushikesh K. Joshi
Department of Computer Science and Engineering
Indian Institute of Technology Bombay
Mumbai, 400 076, India
Email:{srikrishna,rkj}@cse.iitb.ac.in

Abstract

Characteristics of inheritance metrics are compared with size and length measurement using the property based validation framework of Briand et al. About a dozen inheritance metrics are analyzed and compared with some known object oriented metrics. The analysis first performs an identification of viewpoints and projections of the chosen metrics in order to meaningfully apply the property based framework. For the purpose of this analysis, *nested* and *internal* projections are also newly suggested in this paper. The work results in associating most of the chosen inheritance metrics with either the size or the length aspect, while two metrics remain unclassified.
keywords:Metric Validation Properties, Object Oriented Metrics, Size, Length, Inheritance, Measurement Concepts.

1 Introduction

The metric validation framework of Briand et al. [2] includes mathematical properties for measurement concepts such as size and length. Evaluation of metrics against such measurement concepts provides insights into the characteristics and defects of the metrics. Besides serving as a tool for classification and understanding, property based evaluation can also potentially lead to formulation of new metrics.

Inheritance metrics measure various aspects of inheritance such as depth and breadth in a hierarchy and overriding complexity. Many inheritance metrics have been proposed in the literature. Yet, there are no significant attempts to validate and classify them. In this paper, theoretical validation of size and inheritance metrics is performed against the size and length properties of Briand et al.

Size metrics are commonly found in object oriented approaches. Sizes are not bounded, and they are computed as positive integers. The three size properties namely non-negativity, presence of null value and module additivity as outlined by Briand et al. are summarized in Table 1. The third property of modular

Property	Description
S1	The size of a system is nonnegative *(Non-negativity)*
S2	The size of a system is 0 if the set of elements which constitute the system is empty. *(Null value)*
S3	The size of a system can not be more than the sum of the sizes of its modules. In the case of disjoint modules, the size of a system is equal to the sum of the sizes of the modules. *(Module Additivity)*

Table 1: Size Properties of Briand et al.

additivity sets size metrics apart from other metrics classes such as cohesion and coupling metrics.

A dictionary meaning of *length* [5] mentions it as the longer or the longest dimension of an object. From a measurement theory perspective [1], the length of a system is seen as size of the shortest path between two extremes of the system. Length is therefore not the same as size, since it captures the size from the point of view of the extreme limits, whereas, size in general captures the measured as a whole. *Lines of Code* and CLD are examples of size and length metrics respectively. The length properties observed by Briand et al. are summarized in Table 2.

The analysis reported in this paper reveals that some inheritance metrics follow the patterns of size metrics, while some others follow the length properties. Some inheritance metrics have been found to follow none of the two.

The paper is organized as follows. In Section 2, the notion of viewpoints and projections are discussed. Section 3 discusses different kinds of mergers relevant for property based validation. The metrics are analyzed against size and length properties in Sections 4 and 5 respectively.

2 Viewpoints and Projections

The notion of *viewpoints* and *projections* was first introduced in [4] to aid classification of metrics. The notion of viewpoint captures the level of reference abstraction at which the measurement is carried out. Viewpoints can be macroscopic or microscopic. Projections capture the directionality of interaction between the reference abstraction of the view point and the portion of the program that is relevant for the measurement. Projections can be outward, inward or gross projections. While inward projection measurement requires information about contributions and interactions from entities external to the viewpoint, outward projection metrics use outgoing interactions or contributions. On the

Property	Description
L1	The length of a system is nonnegative *(Non-negativity)*
L2	The length of a system is 0 if the set of elements which constitute the system is empty. *(Null value)*
L3	Adding relationships between the elements of a module m in a system does not increase the length of the system. *(Non Increasing Monotonicity)*
L4	A system having modules m1 and m2 such that they are represented by separate connected components in the system; adding relationships from elements of m1 to elements of m2 does not decrease the length of system.*(Non Decreasing Monotonicity)*
L5	The length of a system made of union of two disjoint modules m1 and m2 is equal to the maximum of the lengths of m1 and m2. *(Merger)*

Table 2: Length Properties of Briand et al.

other hand, gross projection indicates bidirectional contributions and interactions with the external entities. In addition to above viewpoints and projections, for the analysis reported in this paper, we further refine the projections to *nested inward*, *nested outward*, *nested gross* and *internal projections*. Metrics following nested projections trace interactions and contributions transitively across multiple levels. Internal projections are used when entities external to the view point are not involved in the measurement.

Viewpoints and projections used by the metrics analyzed in this paper are enlisted in Table 3. For example, the inheritance metric *Specialization Ratio* is defined with system viewpoint, a macroscopic viewpoint, and with internal projection. Few metrics such as DIT and *Fandown* can't be computed using one level of interactions. DIT is a class viewpoint metric generating values per class. It follows nested outward projection as it traces outgoing parent links in the inheritance chain. On the other hand, *Class-leaf Depth*, a class viewpoint metric uses nested inward projection as it recursively traces incoming parent relations from child classes. NoVM uses nested gross projection since it requires the information about the methods declared in ancestor chain, and the concrete implementations that the reference abstraction contributes into the ancestors through overriding.

Viewpoints play a role in property based evaluation of metrics. For example, consider the case of property $L4$, which refers to system and its modules. When this property is applied to a metric with class viewpoint such as NOC, *class* takes

the role of *system*, while components of the class play the role of modules that form the system. Thus *adding relations between modules* as defined in property $L4$ translates to adding relations between member functions and attributes that form the class under measurement. On the other hand, property $L5$ requires that the metric that is being validated is defined over modules and the resultant module merger. In this case, *module* refers to *viewpoint*, which happens to be *class* for metric NOC. In this way, while *class* plays the role of *system* w.r.t. one property it may play the role of *module* as referenced in another property.

Projections when seen in the perspective of viewpoints help in disambiguating this assignment of program entities to roles referred by the properties. For example applying $S3$ to metric NOC requires that the role *system* be mapped to *class*. However, computation of NOC requires information from outside the system. Therefore, one would ask, *is a class to be seen as system or the entire collection of the classes that are involved in the measurement of NOC?* This apparent contradiction is resolved by noting that the projection for NOC is *nested outward*. In other words, the projection of the metric confirms the fact that the metric requires information from outside the viewpoint entity that plays the role of *system* as far as property $S3$ is concerned.

3 Module Mergers

Module mergers are often referred in validation properties to capture the phenomena of decomposition and composition. The focus of the discussion in this section is mainly on size and length properties of the validation framework of Briand et al. For example, to apply $S3$ or $L5$ on a system and its disjoint modules, a *merge operation* on the disjoint modules of the system under measurement needs to be defined. Also, both the properties require that the metric be defined on the modules as well on the system.

There are various ways to merge disjoint classes. A merger can be obtained at source code level or also through inheritance techniques such as mixin [7]. At source code level *union merge* merges two classes by removing duplicates. An example of union merge is *Hard Merge* [7], in which, no separate copies of common superclasses are made.

An *alpha renaming merge* on the other hand requires appropriate renaming of clashing names. For example, source level alpha renaming is used for class merger in [6]. Alpha renaming is also required in inheritance based mergers to eliminate ambiguities such as when used in *MixIn inheritance* based merge [7].

An example of source level merger is the merger of a base class and its derived class in *collapse hierarchy* refactoring [3]. In this case, property duplications are handled by generalization. However, such classes not being disjoint due to method or attribute sharing that automatically occurs through inheritance. Property $S3$ defined over merger of disjoint modules is not directly applicable to such mergers.

Throughout this paper, we consider the source code level mergers i.e, *alpha renaming* and *union merge*. Alpha renaming is useful in ambiguous situations

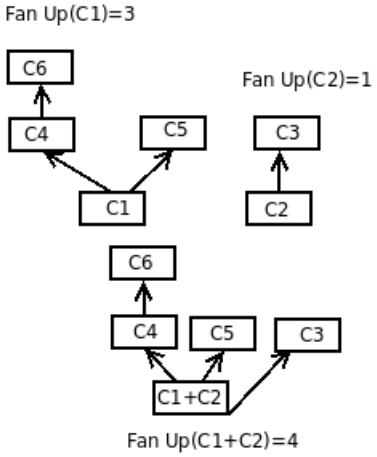

Figure 1: *Fanup* satisfying $S3$

arising out of using same names. Union merge is used when redundancies need to be removed. In a merger, both kinds of merges may be applicable. Irrespective of whether the names are same or different, when the entities are semantically equivalent, a union merge is applicable. Similarly, when the entities are semantically different, a hard merge based on alpha renaming is applicable when the names are same.

Properties $S3$ and $L5$ requiring merger of modules themselves require the modules to be disjoint. The disjointness between the modules can be both syntactic and semantic. Syntactically disjoint modules may not refer to common entities defined in the program. Semantically disjoint modules may not have internal units that are semantically equivalent. If two modules are not semantically disjoint a union merge may be used to eliminate redundancies. If they are semantically disjoint, an alpha renaming may still be required to eliminate ambiguities.

4 Size Properties

In this section, 10 inheritance metrics are evaluated against size properties given in Table 1. The metrics and their evaluation against size properties is summarized in Table 4. It can be noted that *non negativity* and *null value* are satisfied by all metrics. Hence, the rest of the discussion focuses on property $S3$.

The eight size metrics considered satisfy all the three size properties as they are relationship-independent counting measures defined in terms of entities. Also the mergers considered for evaluating property $S3$ do not introduce a new entity in the system as opposed to inheritance based mergers such as mixin. The property $S3$ may not be satisfied under mergers of the latter kind.

Fanup is a measure with class viewpoint. For a metric with class viewpoint,

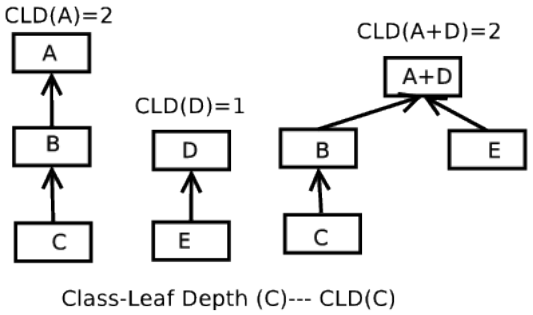

Figure 2: Class-Leaf Depth does not satisfy $S3$

the class under measurement plays the role of *system*. Since the metric uses nested outward projections, the external entities that are indirectly related to class and that are essential for metric calculation are also included as parts of the system. On applying the properties against the metric, it is observed that the metric satisfies property $S3$. A typical situation satisfying $S3$ is shown in Figure 1, in which, C1 and C2 are two disjoint classes. Source level merger does not increase the number of ancestor classes between the modules as merging would club the two classes into a single class borrowing the subclasses and the superclasses from the two disjoint hierarchies into the newly formed hierarchy after the merger.

On the same lines, *Fandown* metric also satisfies property $S3$. In this case the descendents from two disjoint trees are considered instead of superclasses. NOC is a sub case of $Fandown$ with the difference that NOC considers only the immediate subclasses, while $Fandown$ considers all the subclasses.

Class-leaf depth (CLD) measures the maximum length of the path from a class to a leaf. The metric uses nested inward projection to measure the maximum depth since the depth is a consequence of incoming upward inheritance links. The metric does not satisfy property $S3$. A counter example proving the same is shown in Figure 2. *Depth of inheritance (DIT)* is complementary to CLD. The latter measures the length of inheritance from a class up to the root class. DIT does not satisfy property $S3$ as proved by the counter example shown in Figure 3. However, the metrics satisfy the five length properties as discussed in the next section.

Specialization ratio (S), a metric with system viewpoint and internal projection does not satisfy property $S3$ as shown in the counter example given in Figure 4. Similarly, *Reuse ratio (U)* does not satisfy $S3$ as shown in the same counter example.

Number of inherited attributes (NIA), *Number of inherited methods (NIM)* and *Number of overridden methods (NoVM)* for a class are dependent on the ancestor classes of the class that is under measurement. When two semantically disjoint inheritance trees are merged, the resulting merged class has all the inherited attributes and methods from the ancestors of both the classes.

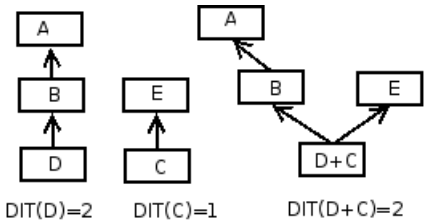

Figure 3: DIT does not satisfy $S3$

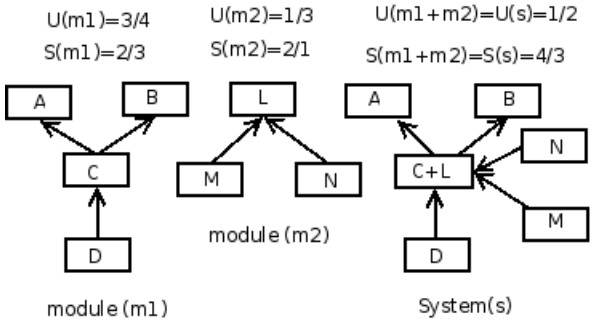

Figure 4: Specialization Ratio and Reuse Ratio do not satisfy $S3$

Therefore, they satisfy property $S3$. All the three metrics use nested outward projections.

So far, the discussion was pertaining to merging of modules that are both syntactically and semantically disjoint. However, it can be noted that in the case of modules that are syntactically disjoint but not semantically disjoint, under application of union merge to remove semantic redundancies, property $S3$ is not satisfied by all size metrics. Therefore, while applying size property $S3$, both syntactic and semantic disjointedness need to be observed.

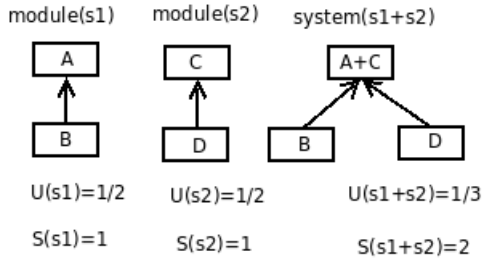

Figure 5: Reuse ratio and Specialization Ratio do not satisfy $L5$

5 Length Properties

Length properties $L1$ and $L2$ are the same as $S1$ and $S2$ respectively. Table 4 evaluates the selected inheritance and size metrics against properties $L3$, $L4$ and $L5$. Size metrics satisfy *non-increasing monotonicity* ($L3$) and *non-decreasing monotonicity* property ($L4$), since adding relationships between the elements within a module or between the elements of different modules in the system does not change the count of the entities already present in the system. However, the considered size metrics do not satisfy *module merger* ($L5$) because the size metrics are sum oriented and not comparison oriented. Thus, it can be observed that metrics satisfying sum oriented size property $S3$ do not satisfy comparison oriented length property $L5$.

Class-Leaf Depth is a class viewpoint based inheritance metric. For a metric with class viewpoint, the modules are inner components of a class, which are methods and attributes. Adding new relations among the inner components does not effect the metric value of the enclosing class as the metric uses external inward projection. Therefore, CLD satisfies both $L3$ and $L4$. Property $L5$ is also satisfied by the metric as the definition of the metric itself inherently captures the comparison based property. *Depth of Inheritance Tree (DIT)* follows the same evaluation results as that of CLD, since the two metrics are complementary in nature in terms of path lengths up to either leaves or roots.

Reuse Ratio and *Specialization Ratio* are system viewpoint based inheritance metrics. Modules for these metrics are classes with the system playing the role of the connected component under measurement. In this case, adding relations within modules does not affect the metric value. Therefore, property $L3$ is satisfied by both reuse ratio and specialization ratio. However, unlike other inheritance metrics, adding relations among the modules may change the metric value, since modules being classes, new inheritance relations can be created between them. Property $L4$ is satisfied by Reuse Ratio since addition of new inheritance relations does not reduce the number of superclasses. However, Specialization Ratio does not satisfy $L4$ as shown in Figure 6. The metric is defined with system viewpoint such that the system under measurement is a hierarchy. As shown in Figure 5, property $L5$ is not satisfied by both the metrics.

Number of inherited attributes (NIA) , *Number of inherited methods (NIM)* and *Number of overridden methods methods (NoVM)* satisfy properties $L3$ and $L4$ as they consider only the methods and attributes defined in ancestor classes and are not dependent on relations within a class. They do not satisfy $L5$ as they are counting metrics satisfying $S3$.

Fanup, a classes viewpoint metric satisfies properties $L3$ and $L4$, as the metric does not use the relations among the methods and attributes. Adding new relations among the methods and attributes does not change the metric value. From Figure 1, it can be observed that $Fanup$ does not satisfy $L5$. *Fandown*, a complementary metric and *Number of Children (NOC)*, a special case of $Fandown$ exhibit the same results as that of $Fanup$.

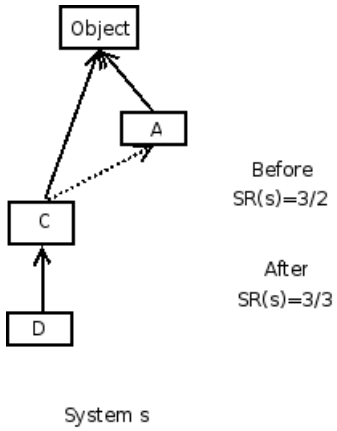

Figure 6: Specialization Ratio does not satisfy $L4$

6 Conclusions and Future Work

To meaningfully apply the framework of Briand et al., a mapping of terms *system, module, elements, and relations* with appropriate program components needs to be defined. Identification of viewpoints and projections has been found to be essential for this task. New projections were proposed, and the viewpoints and projections were identified for 20 metrics for measuring size, length and inheritance properties. The property based evaluation also requires an analysis of different kinds of mergers. The evaluation of the selected metrics revealed that metrics NIA, NIM, NoVM, NOC, $Fanup$ and $Fandown$ are size metrics as far as inheritance hierarchies are concerned. On the other hand, metrics DIT and CLD were found to follow length properties. Specialization Ratio and Reuse Ratio do not fall under size and length concepts. Further analysis revealed that they also do not satisfy the complexity property *disjoint module additivity* of Briand et al. Similarly, it was found that in general inheritance metrics do not satisfy the *symmetry* property associated with complexity metrics since they are dependent on the directionality of parent child relations between classes. The work also points at a need for further work on the scope of existing validation properties and also on measurement concepts that are relatively less explored.

References

[1] E. B. Allen. Measuring graph abstractions of software: An information-theory approach. In *METRICS '02: Proceedings of the 8th International Symposium on Software Metrics*, page 182, Washington, DC, USA, 2002. IEEE Computer Society.

[2] L. C. Briand, S. Morasca, and V. R. Basili. Property-based software engineering measurement. *IEEE Trans. Software Eng.*, 22(1):68–86, 1996.

[3] M. Fowler. *Refactoring: Improving the Design of Existing Code.* Addison-Wesley, Boston, MA, USA, 1999.

[4] P. Joshi and R. K. Joshi. Microscopic coupling metrics for refactoring. *Proceedings of the Conference on Software Maintenance and Reengineering*, pages 145–152, 2006.

[5] Merriam-Webster. *Merriam-Webster Online Dictionary.* 2010.

[6] F. Tip, C. Laffra, P. F. Sweeney, and D. Streeter. Practical experience with an application extractor for java. *SIGPLAN Not.*, 34(10):292–305, 1999.

[7] S. Yacoub and H. Ammar. *Pattern-Oriented Analysis and Design: Composing Patterns to Design Software Systems.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

Metric	**Definition**	**Viewpoint**	**Projection**
LoC	Lines of Code	Program	Internal
NoC	Number of concrete Classes defined in a system	Class	Internal
NoM	Number of Methods defined in a class	Class	Internal
NoA	Number of Attributes defined in a class	Class	Internal
SIZE2	Number of methods defined in a class + Number of attributes defined in a class	Class	Internal
NOK	Number of occurrences of a keywords in a program	Program	Internal
NOAOP	Number of occurrences of a arithmetic operators in a program	Program	Internal
class-leaf depth (CLD)	Length of the path from the class to farthest leaf	Class	Nested Inward
Reuse Ratio	No. of superclasses / total no. of classes	System	Internal
Specialization Ratio	No. of subclasses/ no. of superclasses	System	Internal
DIT	Depth of Inheritance of a class	Class	Nested Outward
NOC	Number of Children is the number of immediate subclasses subordinated to a class	Class	Nested Inward
Fandown	Number of subclasses of a class	Class	Nested Inward
Fanup	Number of super classes of a class	Class	Nested Outward
NIA	Number of Inherited Attributes in a class	Class	Nested Outward
NIM	Number of Inherited Methods in a class	Class	Nested Outward
NoVM	Number of Overridden Methods in a class	Class	Nested Gross

Table 3: Metrics with their Viewpoint and Projections

Metric	**S1, L1**	**S2, L2**	**S3**	**L3**	**L4**	**L5**
LoC	√	√	√	√	√	X
NoC	√	√	√	√	√	X
NoM	√	√	√	√	√	X
NoA	√	√	√	√	√	X
SIZE2	√	√	√	√	√	X
NOK	√	√	√	√	√	X
NOAOP	√	√	√	√	√	X
CLD	√	√	X	√	√	√
Reuse Ratio	√	√	X	√	√	X
Specialization Ratio	√	√	X	√	X	X
DIT	√	√	X	√	√	√
NOC	√	√	√	√	√	X
Fandown	√	√	√	√	√	X
Fanup	√	√	√	√	√	X
NIA	√	√	√	√	√	X
NIM	√	√	√	√	√	X
NoVM	√	√	√	√	√	X

Table 4: Evaluation Matrix