Object-oriented product metrics: A generic framework

Vijay K. Vaishnavi a,*, Sandeep Purao b, Jens Liegle a

a Department of Computer Information Systems, Robinson College of Business, Georgia State University, Atlanta, GA 30302, United States
b School of Information Sciences and Technology, The Pennsylvania State University, University Park, State College, PA 16802, United States

Received 11 May 2005; received in revised form 5 November 2005; accepted 9 June 2006

Abstract

In spite of considerable prior research, a generic framework has not emerged for structuring work on object-oriented (OO) metrics. We propose such a framework (Generic Framework) for object-oriented product metrics. The framework captures the generic structure of the underlying metrics space (Metrics Space) based on a mereological and set theoretic perspective of the building blocks of OO systems and relational measurement theory. We validate the framework by applying it to a repository of about 350 product metrics. The validation shows that the framework does, indeed, capture the underlying metrics space, and can be useful in identifying gaps and additional metrics that can extend the manner in which Metrics Space is currently populated.

© 2006 Elsevier Inc. All rights reserved.

Keywords: Object-oriented metrics; Product metrics; Generic framework; Metrics Space

1. Introduction

A key research area in engineering of object-oriented software is measurement and the development of metrics [18,25]. A large number of object-oriented (OO) metrics have been proposed (e.g., [19,25,43]), which have been summarized and studied (e.g., [5,10,31]), and classified into frameworks (e.g., [1,20]). However, the dimensions of these frameworks are not well-defined, mixing the metaphors of process and product [20], or internal and external attributes (e.g., size, complexity, reuse, productivity, and quality [1]). In spite of considerable prior work, no generic frameworks are currently available to characterize the structure of the underlying metrics space drawing on first principles such as mereology (a theory of 'parthood'), set theory, and the theory of measurement. Such a framework could be useful in providing a deeper understanding of the existing metrics as well as that of potential metrics that do not currently exist.

© Partially supported by a research grant to the first author by the Robinson College of Business, Georgia State University.
* Corresponding author.
E-mail addresses: vvaishna@gsu.edu (V.K. Vaishnavi), spurao@ist.psu.edu (S. Purao), jliegle@gmail.com (J. Liegle).

0020-0255/$ - see front matter © 2006 Elsevier Inc. All rights reserved.
doi:10.1016/j.ins.2006.06.002
The objective of this research, therefore, is to develop a generic framework (Generic Framework) that clearly brings out this underlying structure of the OO product metrics space. The underlying metrics space (Metrics Space) is the theoretical space that metrics for information systems may populate. The metrics available for use at any point in time represent a partial instantiation of this theoretical metrics space. Metrics Space, thus, is the universal set of potential metrics that may be proposed, and is likely to be larger than the set of metrics currently known and available. This research, thus, stands in contrast to existing research on frameworks in at least two ways. First, current research deals with specific properties, e.g., coupling [2,5] or cohesion [4]. Second, its primary objective is understanding the differences among existing metrics. Unlike the existing work, this research starts from first principles, and develops a mathematical foundation that allows specification of axioms and results that provide a way to structure the overall, theoretical metrics-space. The research, thus, builds on the work by Fenton [14] and some of the fundamental ideas (e.g., mapping between empirical and formal worlds) in Kitchenham et al. [23].

We draw on two theoretical bases: mereology [29,33,39] along with set theory and the representational theory of measurement [15,26] – applying these to the products of object-oriented software development processes. The developed framework is accompanied by a theorem (inferences) that indicate how this Metrics Space should be populated. These inferences reflect the composite nature of the OO system artifacts. Generic Framework is partially validated by demonstrating its application to classify existing metrics [31], and to identify gaps and overlaps in current research.

The developed framework is valuable in a number of ways. First, it allows the examination of existing or new metrics in regard to their compliance with the framework (Generic Framework). Second, it aids in the automatic generation of variations of existing metrics through application of the inferences that one can draw from the framework. Third, since the framework clearly links entities to their properties and in turn to their related metrics, the framework aids in grouping metrics to their respective properties, allowing higher levels of analysis. Finally, the formalism of the framework clearly identifies the components of compound metrics, allowing researchers to further investigate these components and not just the overall compound metrics.

The paper is organized into five sections. Section 2 provides an overview of the theory base used for development of Generic Framework including a mereological and set theoretic perspective on OO concepts and the representational theory of measurement. Section 3 uses the theory base to construct Generic Framework for the underlying OO product Metrics Space, and illustrates the framework using examples. Section 4 discusses approaches to validation of the framework, reports results from applying the framework to an existing repository of over 350 metrics, and discusses the findings in the context provided by the proposed framework. Section 5 concludes the paper.

2. Theory base

Object-oriented metrics can be classified into two related, but distinct, categories: product and process metrics [31]. The former are concerned with properties of existing things, while the latter are concerned with the process of how the things come to be. Product metrics, thus, deal with measurement of ‘things’, treating the information system and its components as ‘things in the world’ in their own right. Following Bunge [6, p. 17; 40; 41], Information Systems are complex artifacts that contain component elements, each of which may possess its own properties. These component elements can be identified in terms of the underlying development paradigm. For example, in information systems built using the object-oriented paradigm, the fundamental building blocks may be objects, organized into classes, which may have attributes and methods, and the relationships among classes including inheritance and aggregation. Each element can possess its own properties, and may also have aggregated properties (i.e., properties that are aggregation of the corresponding properties of its constituents), or emergent properties. An example of an aggregated property is the number of methods in a system, which can be computed as the sum of the number of methods of all classes in the system. An example of an emergent property is the number of methods of a class, which cannot be computed at the method level itself. These properties can be realized with metrics, which can be classified as atomic (i.e., quantifying a property of an entity), cumulative (i.e., as an accumulation of metric values for a set of constituent entities), or compound (i.e., metrics that are not atomic or cumulative).
Theoretical approaches to measuring these properties must, therefore, combine a mereological perspective along with set theoretical perspective, to supplement the relational theory of measurement. The mereological perspective is necessary to capture the parthood relationships among different levels of the hierarchy in an object-oriented system. The set theoretical perspective is necessary to ensure that elements of concepts at each level are considered. Finally, the relational theory of measurement is necessary to provide a sound basis for measurement.

2.1. A mereological and set theoretic perspective

The mereological perspective allows us to view the information system as an aggregation across several levels in terms of its constituent elements. Mereology refers to the theory of ‘parthood’, and therefore, is the source of the notion of aggregation (whole/part) used in OO technology. The basic primitive of mereology is PartOf. The mereological perspective has been part of the research discourse on data abstractions and has proved useful as a source theory to capture the notion of ‘parthood’ or ‘aggregation hierarchy’. For example, a system is an aggregation of classes and relationships among classes, and a class represents an aggregation of methods and attributes. The aggregation hierarchy among the artifacts contained within the information system is, therefore, appropriately captured by a mereological perspective. Fig. 1 shows how these core components may be represented, with mereological perspective, as an aggregation hierarchy. At each level of the hierarchy, traditional set theory, with its membership primitive, continues to apply.

Since mereology was independently invented by different people, it has no standard terminology, notation or axioms. We, therefore, adopt the usual notation to capture the aggregation between elements. For s, the system, let A be the set of attributes, M be the set of methods, C be the set of n classes, and R be the set of relationships among classes. Then, the set of attributes in s is given by a simple union of the sets of attributes within each class. A := \( \bigcup A_k \), \( k = 1, \ldots, n \). Similarly, the set of methods in s is given by a simple union of the sets of methods within each class. M := \( \bigcup M_k \), \( k = 1, \ldots, n \). The disjoint union of all methods and attributes contained within s is defined as: \( M \oplus A := \{(0, m) | m \in M \} \cup \{(1, a) | a \in A \} \), where the indices 0 and 1 allow distinction between methods and attributes. Using this definition, we define C, the set of classes in s as \( C := A \oplus M \), and a particular class, c \( \in C \) as \( c := A_j \oplus M_k \). Classes within a system can also have relationships to other classes. R := \( \bigcup R_{ju} \), \( u = 1, \ldots, m \). We can now define s in set-theoretic terms as: \( s := C \cup R \).

The combination of mereological and set theoretic perspectives, thus, allows us to show that elements at a level are grouped to identify elements at the next higher level. For example, methods and attributes are grouped to form their respective classes, which can be grouped to form subsystems, and eventually, a system. The progression from each lower level to the higher level, thus, signifies the formation of composites, which can have emergent properties in addition to the aggregation of properties of its constituent elements. This is an important distinction that we leverage with the help of the relational measurement theory.

2.2. Relational measurement theory

The relational measurement theory (see Fig. 2) provides an appropriate basis for measurement of the underlying phenomenon, the IS artifact. It seeks to formalize our intuition about properties of things in

---

1 A good source of information about mereology is at [http://plato.stanford.edu/entries/mereology](http://plato.stanford.edu/entries/mereology) (last accessed: September 27, 2005).
the world. Since one tends to perceive the real world by comparing things in them, not by assigning numbers to them [15, pp. 24–25], relationships form the basis of measurement. One, then, attempts to develop metrics that represent attributes of the entities we observe, such that the values of metrics reflect relationships we observe among entities.

The relational system \((E, R)\) with respect to a specific attribute (property) is defined as the set of entities, \(E\), together with the set of empirical relations, \(R\). Measuring the property characterized by \((E, R)\) requires a mapping \(M\) into a numerical relation system \((N, P)\). The representation condition asserts that \(M\) maps entities in \(E\) to numbers in \(N\), and empirical relations in \(R\) to numerical relations in \(P\), such that the empirical relations preserve and are preserved by the numerical relations [15, p. 31]. For example, consider the binary relation \(#\), which is mapped by \(M\) to the numerical relation \(!=\). Then, formally, we can assert: \(x # y \iff M(x)! = M(y)\). The real world is called the domain of the mapping, and the mathematical world is called the range [15, p. 29]. A metric, then, is a function that maps an entity to a number or symbol in order to characterize a property of the entity [15]; and measurement is the process by which numbers or symbols are assigned to properties of entities in the real world in such a way as to describe them according to clearly defined rules [16,32]. As an example, consider the expression “class.density.average-method-size”. It identifies the entity ‘class’, a made-up property called ‘density’, and a metric called ‘average-method-size’. In this expression, the property ‘density’ of different ‘classes’ will exhibit a relationship among one another in an empirical relational system (such as: the density of class ‘\(a\)’ is more than that of ‘\(b\)’). This will be captured by a mapping into the numerical relational system, i.e., by the metric ‘average-method-size’, which will preserve the relationships observed in the empirical system into the numerical system. Fig. 2 provides an outline of this fundamental measurement perspective that we adopt as one of the foundations of our framework.

Following the set theoretic perspective and the aggregation hierarchy described earlier, properties of artifacts at different levels in the aggregation hierarchy may belong to either single elements or may be aggregated across elements of sets at a level to a higher level, or may emerge based on values of elements at lower levels in the hierarchy. Specifically, we can distinguish between properties that represent aggregation across levels (e.g., class.average-method-size that corresponds to method.size), and those which do not, i.e., they do not derive directly from a property at a lower level of the aggregation hierarchy. Such non-aggregated properties may exist at the lowest level of the aggregation hierarchy (elementary property), or may exist at higher levels (emergent property [6]), e.g., number of methods or class.usage-of-attributes-by-methods. Such distinctions, i.e., properties of elements/sets or properties at lower/higher levels or properties that are elementary/emergent represent opportunities to develop a cogent framework that can capture the generic structure of Metrics Space.

3. A generic framework for the structure of the OO metrics space

In this section, we build a generic framework, Generic Framework (GF), for capturing the structure of Metrics Space by applying the theoretical perspectives outlined in the previous section to an internal measurement perspective similar to that suggested by [15].
Definition. Metrics Space is the set of all possible OO product metrics for measuring all possible properties of entities in an object-oriented system with the assumption that a metric for an entity can be measured directly from the entity or from other metric values for the entity or its constituent entities.

We call it the Entity–Property–Metric perspective, contrasting it against the Goal-Question-Metric perspective suggested by [3]. The framework reflects the application of fundamental measurement principles to ‘properties’ of ‘entities’ in object-oriented software. The entities (artifacts) are arranged as an aggregation (following the mereological perspective), and their properties, which are constrained by the position (level) of the artifacts in the hierarchy, are identified following the precepts of Bunge’s ontological principles. The term ‘level’ as we use it in the framework directly corresponds to aggregation levels in the aggregation hierarchy in object-oriented systems. For example, variables in a method belong to the lowest level, methods belong to a level above, classes which are composed of attributes and methods belong to a still higher level, and so on. For any given level ($v$), therefore, one can identify the next lower level ($v-1$) if $v$ is not the lowest level or the next higher level ($v+1$) if $v$ is not the highest level. Unlike the levels in a binary tree, where the root is often considered to be at level 0, and the levels are marked with increasing numbers as one traverses the tree, we use a more intuitive labeling. For example, the lowest level (marked with the lowest number, say, $v$) in the aggregation hierarchy is the level at which variables reside. The next higher level ($v+1$) contains attributes and methods, the further next level ($v+2$) contains classes and relationships among classes; higher still ($v+3$) may be clusters of classes or sub-systems; and the highest (marked by $v+4$) would contain the system itself. If a metric needs to consider a set of systems, yet another higher level (say, $v+5$) may be identified for this set of systems. This manner of defining levels allows a more natural mapping to levels of an aggregation hierarchy.

Following the discussion above, the properties can be classified at the lowest level they exist as non-aggregated properties, and at higher levels as aggregated properties. Non-aggregated properties are referred to as elementary properties at the lowest level and as emergent properties at higher levels of the aggregation hierarchy. Each of these properties may be measured with one or more metrics, each metric representing a specific realization of the property (Fig. 3).

Mapping these properties to the numerical relational system, therefore, involves classifying these metrics as either cumulative or non-cumulative. Informally, a cumulative metric for an entity is one that represents the application of a cumulative operator to metric values for its constituent entities at a lower level, i.e., a cumulative metric value for an entity is computed from the metric values for the set of its constituent entities. On the other hand, a non-cumulative metric is one that may be further categorized as an atomic or a compound metric. An atomic metric is computed for an entity directly, and quantifies a single property of an entity. On the other hand, a compound metric of an entity is computed using metric values of entities at the same level or at lower levels of the aggregation hierarchy, that is, it eventually involves a (mathematical) combination of several atomic metric values.

3.1. Formalizing Generic Framework

In the following, we formalize Generic Framework, GF, for organizing Metrics Space. We represent an object-oriented system as a multi-level aggregation of entities such as classes, attributes, and methods. The aggregation hierarchy is, thus, represented as a composition of entities across levels (Table 1). Properties of entities in $E$ are organized by recognizing that they may exist at different levels (Table 2). We also note that these properties may be aggregated across different levels or may be non-aggregated
The set of all abstraction levels in \( V \)

The aggregation hierarchy into which entities in \( E \) are organized, with \( s \) as the root of the hierarchy

The set of all abstraction levels in \( H \)

An entity, \( e \in E \)

A level in \( H, v \in V \)

The highest (maximum) level in \( H, V_{\text{max}} \in V \)

The lowest (minimum) level in \( H, V_{\text{min}} \in V \)

The set of all entities in \( s \) at level \( v \) in \( H \)

An entity \( e \) at level \( v \) in \( H, e^{[v]} \in E^{[v]} \)

Composition of entities \( e_1^{[v]}, \ldots, e_k^{[v]} \) into \( e^{[v]} \) in \( H, v' < v; e_1, \ldots, e_k \) represent all the descendents of \( e \) in \( H \) at level \( v' \)

The set of all possible properties of all entities in \( E \)

The set of all properties of entities in \( E \) at level \( v \) of \( H, P^{[v]} \subseteq P \)

The set of all aggregated properties in \( P, P^{[\text{aggregated}]} \subseteq P \)

The set of all non-aggregated in \( P, P^{[\text{non-aggregated}]} \subseteq P \)

The set of all elementary properties, \( P^{[\text{elementary}]} \subseteq P^{[\text{non-aggregated}]} \)

The set of all emergent properties, \( P^{[\text{emergent}]} \subseteq P^{[\text{non-aggregated}]} \)

A property, \( p \in P \)

A property \( p \) of an entity \( e, e \in E \)

A property \( p \) of an entity at level \( v \) of \( H \)

A property \( p \) of an entity, \( e, \) at level \( v \) of \( H \)

An aggregated property, \( p^{[\text{aggregated}]} \in P^{[\text{aggregated}]} \)

A non-aggregated property, \( p^{[\text{non-aggregated}]} \in P^{[\text{non-aggregated}]} \)

An elementary property, \( p^{[\text{elementary}]} \in P^{[\text{elementary}]} \)

An emergent property, \( p^{[\text{emergent}]} \in P^{[\text{emergent}]} \)
The set of cumulative operators such as sum, avg, max, min, std-dev or their compositions such as the application of sum followed by average.

A cumulative operator, $o$, may exist at any level of the aggregation hierarchy and is defined directly from an entity using the count operation.

A non-cumulative metric is a metric in Metrics Space that is not accumulated in one or more steps. The base of a non-cumulative metric of an entity is the same as the level of the entity in the aggregation hierarchy. A non-cumulative metric is a metric in Metrics Space that is not cumulative. Such a metric is either an atomic metric or a compound metric but not both. An atomic metric may exist at any level of the aggregation hierarchy and is defined directly from an entity using the count operation.

Thus, a metric at the system level that uses some operator or a sequence of operators to compute a value from a set of values computed for each attribute would be a cumulative metric. It would be measuring the aggregated property at the system level from the measurements of the corresponding property for each attribute, which is at a lower level in the aggregation hierarchy. On the other hand, a metric of a class that is computing something across classes such as the number of messages sent by methods in the class to methods in other classes would be a compound metric since it is neither a cumulative metric nor an atomic metric. Similarly, a metric that is computing something for a system from values measured for its constituent classes and the values measured for its attributes would be a compound metric.

Here are some examples. An example of an atomic metric at the class level is: $\left| m_{i} \right|$ such that (s.t.) $m_{i}$ in c, which reads as the “number of methods in a given class c”. Consequently, a cumulative metric for a system, $s$, is: $\text{sum}(\left| M(c_{i}) \right|)$ s.t. $c_{i}$ in s, which reads as “the sum of the count of all methods of all classes of a system” [31]. A system level compound metric is the “average number of methods in a class”, computed as fraction(sum($\left| M(c_{i}) \right|$)), $\left| c_{i}(s) \right|$ s.t. $c_{i}$ in C(s), i.e., a compound metric uses other metrics in its computation.

---

**Table 3**

<table>
<thead>
<tr>
<th>Metrics for measuring the properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Psi$</td>
</tr>
<tr>
<td>$\Psi^{(v)}$</td>
</tr>
<tr>
<td>O</td>
</tr>
<tr>
<td>$\Psi_{\text{[cumulative]}}$</td>
</tr>
<tr>
<td>$\Psi_{\text{[non-cumulative]}}$</td>
</tr>
<tr>
<td>$\Psi_{\text{[atomic]}}$</td>
</tr>
<tr>
<td>$\Psi_{\text{[compound]}}$</td>
</tr>
<tr>
<td>$\psi$</td>
</tr>
<tr>
<td>$\psi^{(v)}$</td>
</tr>
<tr>
<td>$A$</td>
</tr>
<tr>
<td>$\psi_{\text{[cumulative]}}$</td>
</tr>
<tr>
<td>$\psi_{\text{[non-cumulative]}}$</td>
</tr>
<tr>
<td>$\psi_{\text{[atomic]}}$</td>
</tr>
<tr>
<td>$\psi_{\text{[compound]}}$</td>
</tr>
<tr>
<td>$V_{\text{base}}(\psi)$</td>
</tr>
<tr>
<td>$\psi_{\text{[non-cumulative]}}$</td>
</tr>
<tr>
<td>$\psi_{\text{[atomic]}}$</td>
</tr>
<tr>
<td>$\psi_{\text{[compound]}}$</td>
</tr>
<tr>
<td>$\psi^{(p)}$</td>
</tr>
<tr>
<td>$\psi^{(p(e))}$</td>
</tr>
<tr>
<td>$\psi^{(p[\text{aggregated}]})$</td>
</tr>
<tr>
<td>$\psi^{(p[\text{non-aggregated}]})$</td>
</tr>
<tr>
<td>$\text{MPM}: \Psi \rightarrow P$</td>
</tr>
</tbody>
</table>

---

**Table 4**

<table>
<thead>
<tr>
<th>Metrics as realization of measurement of properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\psi^{(p)}$</td>
</tr>
<tr>
<td>$\psi^{(p(e))}$</td>
</tr>
<tr>
<td>$\psi^{(p[\text{aggregated}]})$</td>
</tr>
<tr>
<td>$\psi^{(p[\text{non-aggregated}]})$</td>
</tr>
<tr>
<td>$\text{MPM}: \Psi \rightarrow P$</td>
</tr>
</tbody>
</table>

- $\psi$ realizing the measurement of $p$
- $\psi$ realizing the measurement of $p$ of entity $e$
- $\psi$ realizing the measurement of $p[\text{aggregated}]$
- $\psi$ realizing the measurement of $p[\text{non-aggregated}]$

Metrics-properties mapping; $\psi \rightarrow p \in \text{MPM}$ iff $\psi^{(p)}$
Table 5
Definitions: types of metrics and their bases

Cumulative metrics:
\[ \psi = \Psi^i_{[\text{cumulative}]}(\psi) \mid p(e) \text{ such that } e = (e_1^{[i]}, \ldots, e_k^{[i]}), \psi' = \Psi^i_{[\text{non-cumulative}]} | p' (e_1^{[i]}, \ldots, e_k^{[i]}), 1 \leq i \leq k \]
\[ \text{iff } \psi(p(e)) = o(\psi'(p'(e_1), \ldots, \psi'(p'(e_k))) \land v' < v) \]
//A cumulative metric for measuring the property \( p \) of an entity \( e \) at level \( v \) is computed directly using a cumulative operator from the values obtained from measuring another property \( p' \) for each of the entities at level \( v' \) constituting \( e \); according to the definition of an aggregated property given below, \( p \) is an aggregated property – aggregation of \( p' // \)

Base of a cumulative metric, \( \psi = \Psi^i_{[\text{cumulative}]}(\psi) \) such that \( \psi' = \Psi^i_{[\text{non-cumulative}]} \rightarrow V_{\text{base}}(\psi) = v' \);
\[ \psi' = \Psi^i_{[\text{non-cumulative}]} \rightarrow V_{\text{base}}(\psi) = V_{\text{base}}(\psi') \]
//The base of a cumulative metric is the base of the non-cumulative metric from which the metric is directly computed in one or more steps//

Non-cumulative metrics:
\[ \psi = \Psi^i_{[\text{non-cumulative}]} \text{iff } \psi \not\in \Psi_{[\text{cumulative}]} \]
//A non-cumulative metric is a metric in the Metrics Space that is not cumulative//

Base of a non-cumulative metric:
\[ \psi = \Psi^i_{[\text{non-cumulative}]} \rightarrow V_{\text{base}}(\psi) = v, \]
//The base of a non-cumulative metric for an entity is the level of the entity itself in the aggregation hierarchy//

Atomic metrics and compound metrics:
\[ \Psi_{[\text{non-cumulative}]} = \Psi_{[\text{atomic}]} \cup \Psi_{[\text{compound}], \Psi_{[\text{atomic}]} \cap \Psi_{[\text{compound}]} = \Phi} \]
//A non-cumulative metric can be classified as an atomic metric or a compound metric but not both//

Atomic metrics:
\[ \psi = \psi^{[i]}_{[\text{atomic}]}, p(e) \text{ iff } \psi \in \Psi_{[\text{non-cumulative}]} \land \psi(e) = \text{count}(e) \]
//An atomic metric of an entity is a non-cumulative metric that is computed directly from the entity using the count operation.//

Compound metrics:
\[ \psi = \psi^{[i]}_{[\text{compound}]}, \text{iff } \psi \in (\Psi_{[\text{non-cumulative}]} - \Psi_{[\text{atomic}]}) \]
//A compound metric is a non-cumulative metric that is not atomic//

* While this definition may appear naïve, counting is a difficult operation that requires precise definition of elements and counting procedures. For instance, lines of code may mean statements, non-commented lines, non-blank lines, etc.

Table 6
Definitions: types of properties

Aggregated properties
\[ p \in P_{[\text{aggregated}]}, \text{iff } \exists \psi \in \Psi_{[\text{cumulative}]} \text{ such that } \psi | p \]
//An aggregated property is realized by a cumulative metric//

Non-aggregated properties – elementary and emergent properties
\[ P_{[\text{non-aggregated}]} = P_{[\text{elementary}]} \cup P_{[\text{emergent}]} \]
\[ p = p^{[i]}_{[\text{elementary}]}, \text{iff } p \in P_{[\text{non-aggregated}]}, v = V_{\text{min}} \]
\[ p = p^{[i]}_{[\text{emergent}]}, \text{iff } p \in (P_{[\text{non-aggregated}]} - P_{[\text{elementary}]}), v > V_{\text{min}} \]
//A non-aggregated property is either an elementary property or an emergent property. At the lowest level of the aggregation hierarchy, a non-aggregated property is called elementary; at higher levels it is called emergent//

in this case spanning different levels – the number of classes in a system and the number of methods of all classes in a system. Additional examples are provided in the discussion around Fig. 6. Next, two kinds of properties, aggregated and non-aggregated properties are defined (Table 6).

An aggregated property is a property that is realized by at least one cumulative metric. A non-aggregated property is either an elementary property or an emergent property. A non-aggregated property at the lowest level of the aggregation hierarchy is called an elementary property and the one at a higher level of the aggregation hierarchy is called an emergent property.
Axioms of the Framework
Based on the concepts defined above, the framework is governed by the following axioms:

**Axiom I:** \( P = P_{\text{aggregated}} \cup P_{\text{non-aggregated}}; P_{\text{aggregated}} \cap P_{\text{non-aggregated}} = \emptyset \)
//The set of all properties is the union of two non-overlapping sets, the set of all aggregated properties and the set of all non-aggregated properties.//

**Axiom II:** \( \Psi = \Psi_{\text{cumulative}} \cup \Psi_{\text{non-cumulative}}; \Psi_{\text{cumulative}} \cap \Psi_{\text{non-cumulative}} = \emptyset \)
//The set of metrics in Metrics Space is the union of two non-overlapping sets, the set of all cumulative metrics and the set of all non-cumulative metrics.//

**Axiom III:** MPM: \( \Psi \rightarrow P \) is a surjective (onto) but not injective (one-to-one) mapping
//Metrics in Metrics Space measure all possible properties of all entities of an object-oriented system; however, a metric cannot measure two different properties. A unique property (of an entity at a certain level of the aggregation hierarchy) exists for every existing metric and for each property there exist one or more metrics for measuring it.//

Axiom III suggests that there may be different ways of measuring a property, and thus, a metric ‘realizes’ i.e., it represents a specific instantiation of that property. It may, of course, be possible to interpret a metric to ‘inform’ multiple properties. For example, Stein et al. [36] distinguish between syntactic and semantic properties, indicating that the two may be correlated but not equivalent. The axiom is analogous to this argument, suggesting that the decision to use a metric to measure multiple properties does not necessarily make the metric equivalent for measuring these properties. Yet another analogy to the arguments from Stein et al. [36] that is similar to our presentation is the distinction between internal and external properties. Theoretically, a metric cannot measure two different properties.

**Axiom IV:** \( \psi \in \Psi_{v} \land v < V_{\text{max}} \Rightarrow \exists \psi' \in \Psi_{v'} \) such that \( \psi' = \psi_{v' \text{[cumulative|atomic|compound]}} \land v < v' \leq V_{\text{max}} \)
//A metric not at the highest level of the aggregation hierarchy can be accumulated to a higher level using a cumulative operator in O.//

**Axiom V:** MPM (\( \psi_{\text{cumulative}} = p \Rightarrow p \in P_{\text{aggregated}} \))
//If a cumulative metric realizes the measurement of a property then the property must be an aggregated property.//

3.2. Making inferences in Metrics Space

Metrics Space, defined by Generic Framework, provides several opportunities for defining properties and making inferences. We describe some of these as a *Theorem (Theorem 3.1)* that describe such inferences (a) across levels of the aggregation hierarchy, (b) within a selected level of the aggregation hierarchy, (c) for defining compound metrics, or (d) across metrics and properties. The theorem follows from the above axioms and definitions.

**Theorem 3.1**

**Inter-Level Inferences:**

* I: \( \psi \in \Psi_{v} \Rightarrow \psi = \psi_{v} \psi_{v'} \) such that \( \psi' = \psi_{v' \text{[cumulative|atomic|compound]}}, v < v' < V_{\text{max}} \)
//A cumulative metric at level \( v \) exists in a related form at a lower level in the hierarchy.//

* II: \( \psi = \psi_{v} \psi_{v'} \land v < v - 1 \Rightarrow \exists \psi'' \in \Psi_{v+1} \land \psi'' = \psi_{v' \text{[cumulative]}}, v = \psi_{v' \text{[atomic]}}, v = \psi_{v' \text{[compound]}} \)
//If a metric \( \psi \) is accumulated directly from a metric \( \psi' \) at a level other than the next lower level then there exists another metric \( \psi'' \) whose value is accumulated from \( \psi' \) and from which \( \psi \) can be accumulated directly.//

**Intra-Level Inference:**

* III: \( \psi \in \Psi_{v} \Rightarrow \exists \psi' \in \Psi_{v'} \mid \psi' = \psi_{v' \text{[atomic]}}, v' = v_{0} \land v_{0} \in O \)
//If one form of a cumulative metric exists at a level, then other variations are possible.//
Compound Metrics Inference:

IV: \( \psi = \psi^{[\text{compound}]|w} \Rightarrow \psi' \in \Psi' \iff (\psi' \in \Psi^{[m]}_{\text{cumulative}}) \lor (\psi' \in \Psi^{[m]}_{\text{atomic}}) \lor (\psi' \in \Psi^{[m]}_{\text{compound}}) \), \( m \leq v \)

//A compound metric at a certain level of the aggregation hierarchy is a non-cumulative metric that is computed directly from metrics at the same level or lower levels that are cumulative, atomic, or compound metrics.//

Metrics – Properties Inferences:

V: \( \text{MPM}(\psi) = p_{\text{aggregated}} \iff \psi \in \Psi_{\text{cumulative}} \)

//A metric realizes the measurement of an aggregated property if and only if the metric is a cumulative metric.//

VI: \( \text{MPM}(\psi) = p_{\text{non-aggregated}} \iff \psi \in \Psi_{\text{non-cumulative}} \)

//A metric realizes a non-aggregated property if and only if the metric is a non-cumulative metric.//

3.3. An illustration

We illustrate how the proposed framework captures the underlying metrics space with an extended example. The example demonstrates how properties and metrics can be defined and expanded over multiple levels of the aggregation hierarchy. Specifically, we use the attribute and method aggregation hierarchy that defines a class. An elementary property ‘size-of-method’ can be identified for the artifact ‘method’. This property can be aggregated, and becomes the aggregated property2 ‘size-of-class’ for the artifact ‘class’. The corresponding metrics include the atomic metric ‘Lines of Code’ for the artifact ‘method’ and the cumulative metric ‘Lines of Code’ for the artifact ‘class’. This represents the simplest form of a cumulative metric: an extension of the count of the corresponding metric at the lower level. Thus, the metric ‘Lines of Code’ for the artifact ‘class’ can be defined as the sum of the metric ‘Lines of Code’ for its constituent elements, i.e., its methods (Fig. 4). Other forms of cumulative metrics include average, maximum, minimum or standard deviation. Properties may also emerge at higher levels, measured using compound metrics (Fig. 5).

For a more complete demonstration of the framework, we provide an example using the formalism proposed by Purao and Vaishnavi [31], and borrow a specific set of metrics from their repository, which contains standardized expressions of about 350 product metrics. The example we have selected allows us to demonstrate several features of the framework including, but not limited to those demonstrated in Figs. 4 and 5 above. For example, an emergent property (not shown) is a class coupling metric such as \( |C_{\text{interacting}}| \), i.e., the number of classes that interact with a given class [8]. This metric can only exist beginning at the class level, but could later be aggregated to the system level in form of an average, etc.

Fig. 6 illustrates a hypothetical emergent property ‘Comments’ for the artifact ‘class’, measured by the compound metric \( M_6 \) defined as: fraction(\( |M_{\text{commented}}(c)|, |M(c)| \)). This compound metric can be broken down

---

2 For illustration, we use this definition of the property ‘Size’ of the entity ‘Class’. Other definitions are possible.
into its components, in this case, \(|M_{\text{commented}}(c)|\) and \(|M(c)|\) of a given class. Metric \(M_5\) simply measures the number of commented methods of a class, defined as, \(|M_{\text{commented}}(c)|\). This is an example of an atomic metric measuring an emergent property. Metrics \(M_4\) and \(M_3\) represent cumulative metrics for the property complexity for the artifact class. While the figure shows two of these metrics, other statistical forms (e.g., average, min, max or other mathematical or positional statistics) are possible as well. For the artifact class, metric \(M_2\) ‘LinesOfCode(c)’, in short LOC(c), meaning Lines of code of a class, is computed by summing the lines of code of the methods in the class, i.e., \(|\text{LOC}(m_i)|\) s.t. \(m_i \in c\), which translates to count of the lines of code for each method \(m_i\) such that \(m_i\) is in the class \(c\). This metric is based on the number of lines of code in the methods of a given class. Thus, it measures an aggregated property. The corresponding metric must, therefore, exist at a lower level. Metric \(M_1\), \(|\text{LOC}(m)|\), at the method level counting the lines of code of a method.
(measuring the elementary property, size) fulfills this requirement. We remind the reader that the property 'size' can be defined in several ways; this example is just one way of measuring size.

The illustration shows how the aggregation framework can be used to position a metric in relation to the properties of the artifact it measures, and other metrics in the aggregation hierarchy – existing or possible. In this manner, the framework shows how additional metrics may be identified which may not have yet been proposed. The inverse is also true. Based on existing cumulative metrics, corresponding non-cumulative metrics can also be derived. A systematic analysis of the metric repository can therefore help in the discovery of these potentially missing metrics.

4. Validating the framework

Validating the proposed framework (Generic Framework) is a difficult proposition because it attempts to formalize and lend structure to the theoretical space underlying object-oriented product metrics (Metrics Space). Existing metrics provide only a partial manifestation of this underlying space. Clearly, the framework and accompanying axioms and inferences must withstand the critical test of being able to represent existing research in object-oriented product metrics. This ability can provide prima facie evidence of the appropriateness of the proposed framework. Such an exercise can also result in identification of gaps and overlaps in current research.

To validate the proposed framework, it is necessary to show correspondence between the metrics predicted by Generic Framework and Metric Space. The ability to represent metrics proposed by existing research in this space is, then, a necessary condition to demonstrate this correspondence. This necessary condition can be specified as:

\[ \Psi \rightarrow GF \]

i.e., that there cannot be a metric in Metrics Space that is not implied by Generic Framework. Verifying this would entail examining all existing metrics to ensure that the framework predicts them. A brute force method to verifying this condition would require ensuring that the framework can explain all existing metrics proposed by researchers. A corresponding, sufficient, condition needs to be verified to fully validate the framework. This condition can be specified as:

\[ GF \rightarrow \Psi \]

i.e., there cannot be a metric that is predicted by the framework but is not in Metrics Space. A brute force method to verifying this condition would require generating all possible metrics based on the framework, and ensuring that researchers have proposed or can propose them. This, however, is an impossible task. The properties of artifacts are not well defined. For example, the commonly referred to property ‘size of a system’ can be defined in a number of ways, such as number of lines of code in a system, number of classes of a system, number of attributes of a system, etc. For each of these metrics, there are several ways of measuring it, e.g., lines of code can be measured as number of non-commented lines of code, number of overall lines of code, number of non-empty lines of code, etc. Since each property is realized by one or more metrics, there are countless ways of measuring the properties, and there are countless possible variations of metrics. The underlying metrics space (Metrics Space) is, thus, vast, in stark contrast to the sparsely populated manifested Metrics Space. Our approach to test the sufficient condition, therefore, involves examining summary outcomes from the repository and comparing these against expectations indicated by the inferences, followed by hypothesizing systematic extensions of the manifested Metrics Space to demonstrate that these variations either lead to other existing metrics or suggest potential metrics that could exist in the theoretical metrics space.

For the analysis that leads to the verification of both conditions, we use a metrics repository (Repository) that employs a standard formalism suggested to represent about 350 product metrics [31]. The formalism expresses each metric as the application of mathematical operators on arguments, which can be elements or sets representing values representing measurement of properties of entities. Complex operators are defined as functions, which are mapped to primitive operators. The formalism uses a predetermined set of symbols...
for denoting different artifacts providing a common representation language, and the notations for elements,
sets, and operations provides a grammar for this representation. For example, ‘Attribute Hiding Factor’ [1] is
specified as System/structure/fraction_hidden_attribs() and its computation is shown as fraction(|A_hidden|C(s)|, |A(C(s))|). The computation is read as follows. First, count hidden attributes of all classes in the
system. Next, count all attributes of all classes in the system. Finally, compute the fraction using the first count
as the numerator and the second as the denominator. Fig. 7 shows the structure of the repository using a
standard entity-relationship notation that was extended to represent information following the proposed
framework. The figure shows the underlying Entity–Property–Metric structure and indicates additional information (in italics) added to the repository, which allowed carrying out analyses necessary for validation.

Analyses reported in the paper, thus, represent queries executed against a database structured in this manner,
and populated with the metrics represented in the repository indicated above.

4.1. Verifying the necessary condition: Metrics Space ⇒ Generic Framework

A brute force approach was used to verify that no existing metrics contradicted the framework i.e., each
metric in the repository respected the constraints that specified how the framework should be populated.
The algorithm used for this analysis is shown in Fig. 8 below. The algorithm found no violations except
for two metrics, which were specified by researchers using the temporal dimension. Table 7 below shows these
metrics.

These two metrics specify operations applied to properties of a system over time i.e., the set used for the
cumulative metric consists of the system at time t₁, t₂, ..., tₙ versus the system at a given time t for the atomic
metric. Inclusion of the metrics in the repository suggests a possible breach of the bounds set on the repository
itself i.e., the first metric shows consequences of the behavior of the artifact in an environment over time, and
the second metric shows the effort expended in refining the artifact. These properties, thus, exist at the inter-
section of multiple dimensions (entities): in the first case, the system and the environment, and in the second
case, the system and the developer (effort). Additional metrics of this nature have been proposed (see, for
example,[10]) that go beyond the scope of the proposed framework. Further support for our argument is
found in the characterization of these metrics as representing the process (in the first case, of maintenance;
in the second, of development) as opposed to the product, the focus of the proposed framework. Verification
of the necessary condition was, therefore, considered successful.

This exercise further showed that the framework does, in fact, allow us to determine if metrics are mutually
consistent. This consistency across metrics, which may have been proposed by different researchers, can be
investigated in several ways. The following discussion demonstrates two of these investigation paths. First,
one can investigate whether terms (e.g., metric names, names of elements computed) are used in a consistent
manner by different researchers while defining metrics. Second, one can investigate whether the computation
of metrics (e.g., computation of number of methods, computation of lines of code) is done in a consistent man-
ner as proposed by different researchers.

4.2. Verifying the sufficient condition: Generic Framework ⇒ Metrics Space

The problem of verifying the sufficient condition is not as straightforward and simple as the algorithm
employed for the necessary condition. As discussed earlier, the task here entails ensuring that the theoretical
space the framework describes in terms of inferences (see Section 3.2) can be mapped to the current metrics
space (manifested Metrics Space). These inferences provide expectations that the current metrics space and its
extensions need to meet to verify that the framework is sufficient to represent Metrics Space. A cautionary
note is in order here. In the discussion that follows, we do not advocate defining metrics that can be measured
for the sake of measurement itself. Our analogy here is that of the periodic table, where the discovery of some
elements in the table might suggest the possible existence of other elements. The framework we have developed
provides structure to the underlying metrics space. The exercise of discovering additional metrics, therefore,
suggests the potential for existence of additional metrics that may be measured if a need is perceived.

4.2.1. Inter-level inferences

Following Theorem 3.1 (I, II) (inter-level inferences), we expected to find a smaller number of metrics at the
attribute and method level, and – due to accumulation of metrics at higher levels and additional emergent
properties – a larger number of metrics at the class and system level. Similarly, given a cumulative metric
at one level, we expected to find a corresponding metric at the lower levels e.g., if a system level metric com-
puted average number of methods per class in a system, then there should also be a corresponding metric for
counting methods in a class at the class level. Our analysis of the metric repository did show that there were, in
fact, few metrics available at the attribute and method level. The number of metrics available at the class level
was, however, high and even exceeded those available at the system level (see Table 8).

The emphasis on class metrics may reflect several possibilities. First, it may represent a bias that developers
and managers have about treating classes as the important manageable unit [30, p. 658]. Second, it may be an
indication that a number of emergent properties have been identified at the class level, with no corresponding
aggregations suggested at the system level. It, therefore, may simply represent a peculiarity that is due to the
population of the current metrics space as opposed to that of the underlying theoretical space (Metrics Space).
One reason for this could be that in systems built with OO languages, the class is the primary data abstraction
mechanism, with procedural abstraction (i.e., methods) playing a secondary role. Very few OO languages

Table 7
Contradictions: cumulative metrics for non-aggregated properties (see Theorem 3.1(II))

<table>
<thead>
<tr>
<th>Metric</th>
<th>Computation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean time between failures</td>
<td>avg(InterfailureTime(s))</td>
</tr>
<tr>
<td>Maintainability efficiency</td>
<td>avg(TimeToIdentifyAndCorrect(Defect(s)))</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 7 Contradictions: cumulative metrics for non-aggregated properties (see Theorem 3.1(II))</th>
<th>Computation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean time between failures</td>
<td>avg(InterfailureTime(s))</td>
</tr>
<tr>
<td>Maintainability efficiency</td>
<td>avg(TimeToIdentifyAndCorrect(Defect(s)))</td>
</tr>
</tbody>
</table>
provide abstraction mechanisms for the system and the subsystem. However, subsystems are useful constructs for tracking and managing projects; so, the dearth of metrics for this level was still a surprise, and suggests an opportunity for research. To further explore these interpretations, a finer grain analysis was performed (see Table 9).

The table shows the number of metrics that measure the same property at different levels. For example, consider the property ‘number of attributes’. This may be measured at both class as well as system levels. The results of this analysis were even more striking. Most metrics (154) did not measure the same property at the two levels (class and system). Many metrics (106) proposed at the system level (understandably) reflected emergent properties that were not identified at the class level e.g., number of abstract classes. While metrics measuring a property at the class level could be used at the system level, such use was, however, missing from the repository. Only two metrics were exploited in this manner. These were the simple count of attributes and methods at the class as well as system level. A few metrics at both levels did compute a similar property. However, the metrics proposed by researchers for these did not clearly exploit this aggregation across levels. For example, the metrics `messages_sent()` computed as $|E_{sentBy}(O(c))|$ at the class level and the corresponding metric `messages_sent()` computed as $|E_{sentFrom}(M(C(s)))|$ at the system level indicated this similarity. The lack of clear aggregation, however, appeared to miss the opportunity. Note that this analysis focused on properties, instead of metrics, which is shown in the next table. This form of analysis showed that some metrics at the system level represented aggregations from a method or attribute metric, indicating potential metrics at the class level. Table 10 shows the number of metrics at the system level that represent aggregates of metrics from the lower levels. The table shows that there were, indeed, instances of clear aggregation of metrics across different levels.

The numbers also suggest the potential for eight new cumulative metrics at the class level based on seven method level and one attribute level metric. For example, the metric average local method size was computed as $\text{avg}(|T_{logical}(m_{ij})|)$ s.t. $m_{ij} \in M(C(s))$, suggesting a potential metric at the class level as: $\text{avg}(|T_{logical}(m_{ij})|)$ s.t. $m_{ij} \in M(c)$. One additional analysis was performed to understand how atomic metrics can be identified at lower levels by examining metrics proposed at the class and system levels that implicitly assumed these atomic metrics. This analysis revealed that several atomic metrics are indeed missing from the current metrics space. Table 11 shows the results of this analysis.

Table 9
Number of metrics for entities at different levels (see Theorem 3.1(I))

<table>
<thead>
<tr>
<th>Entity</th>
<th>Metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td>System</td>
<td>108</td>
</tr>
<tr>
<td>Class</td>
<td>156</td>
</tr>
<tr>
<td>Method</td>
<td>35</td>
</tr>
<tr>
<td>Attribute</td>
<td>6</td>
</tr>
</tbody>
</table>

* Excluding reuse-related product metrics.

Table 10
System level metrics as accumulations of lower level metrics (see Theorem 3.1(II))

<table>
<thead>
<tr>
<th>Metrics</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>System level metrics that represent aggregation of method level metrics</td>
<td>7</td>
</tr>
<tr>
<td>System level metrics that represent aggregation of attribute level metrics</td>
<td>1</td>
</tr>
<tr>
<td>System level metrics based on aggregation of class level metrics</td>
<td>28</td>
</tr>
</tbody>
</table>
The analysis showed that for as many as 27 metrics proposed at the system level (and 47 metrics proposed at the class level), the corresponding atomic metrics were missing at lower levels. These clearly indicate several possibilities for extending the current metrics space with additional metrics at the lower levels. For example, for the metric \( \text{avg\_messages\_sent}() \), defined as \( \text{avg}\left( \sum_{j} \text{E\_sentFrom}(m_i) \right) \) s.t. \( m_i \) in \( M(C(s)) \) for the class level, the corresponding atomic metric \( \sum_{j} \text{E\_sentFrom}(m) \) for the method level was not available. As another example, for the metric \( \text{Avg\_logical\_method\_size}() \), defined as \( \text{avg}\left( \sum_{j} \text{T\_logical}(m_i) \right) \) s.t. \( m_i \) in \( M(C(s)) \), for the system level, the corresponding atomic metric \( \sum_{j} \text{T\_logical}(m) \) for the method level was not available.

### 4.2.2. Intra-level inference

Following Theorem 3.1 (III) (intra-level inference), we expected that if a cumulative metric of a particular type was proposed, e.g., \( \text{sum} \) – we would find other accumulations, such as \( \{\text{avg, max, min, sum, std-dev}\} \) as well as the corresponding atomic metric.

Our analysis showed that a number of cumulative metrics do, in fact, use mathematical aggregation functions such as \( \text{sum, average, maximum, and minimum} \). Table 12 shows a summary of these metrics at the system level. In each row, the table shows the following: For each type of cumulative metric proposed (e.g., average), it shows metrics proposed with other accumulations (e.g., max, min, sum, std-dev). For example, there are a total of 22 metrics that compute an average at the system level, but no metric that computes the corresponding max, min, sum, or std-dev.

The table shows that the current metrics space (manifested Metrics Space) does not contain a number of possible metrics that the theoretical Metrics Space suggests. For example, for the metric \( \text{avg\_effort\_per\_class}() \), computed as \( \text{avg}(\text{Effort\_toBuild}(c_i(s))) \) s.t. \( c_i \) in \( C(s) \), proposed using the cumulation type ‘average’, but corresponding metrics using the cumulative operators, ‘max’ and ‘min’, may also be useful. Table 13 shows an equivalent analysis for class level metrics.

A striking feature of these results is the large number of metrics proposed at the class level using the cumulative operator ‘sum’ without corresponding metrics with other operators. For example the metric \( \text{sum\_attribute\_boehm\_size}() \), computed as \( \text{sum}(\text{size}(A(c))) \), does not exist in the avg form. Tables 12 and 13 together also suggest a large number of potential metrics in Metrics Space that can be easily identified to populate the actual metrics space. For instance, at the class level, only 8 use average, compared to 42 that use sum, suggesting a possible 34 more metrics that may be proposed. Similar inferences are possible for metrics using other aggregation types. We note that the resulting metrics may not measure a property of interest. However, the potential to discover interesting, hitherto unused, properties exists with such analysis.

### 4.2.3. Inference related to compound metrics

Following Theorem 3.1 (IV) (compound metrics inference), we expected to find individual components of compound metrics at the same or lower levels as atomic, compound, or cumulative metrics. Perhaps due to
the high degree of specificity of compound metrics, we did not find accumulation of compound metrics at different levels of aggregation, nor did we find variations of them in terms of different cumulative metrics.

An analysis of metrics contained in the repository was performed to examine how the compound metrics used atomic metrics – existing or new. For the analysis, compound metrics were recursively examined for their atomic components. For example, the following metric at the method level, fraction(difference(inheritance_occurrences(i), overload_occurrences(i)), inheritance_occurrences(i)) uses three atomic metrics that are already defined in the repository, indicated by angular brackets. On the other hand, the following metric, also proposed at the method level, fraction(instance_attributes_used(m), A_c(m)) uses one atomic metric available in the repository, indicated by angular brackets and one that is not, indicated without the angular brackets. Table 14 shows the results of this analysis.

Out of a total of 71 metrics, as many as 44 used some or only existing atomic metrics, that is, only 27 used only new atomic metrics that were not in the currently populated metrics-space. The results indicate that a large number of compound metrics do, in fact, use existing atomic metrics for the computation of compound metrics.

The above analysis shows that inferences suggested by the inferences were satisfied by an analysis of the currently populated metrics space, extended using the inferences and their underlying axioms. The examples following each analysis indicate these extensions. The analysis, therefore, can also be seen as directions in which systematic extensions of the current metrics space can be undertaken by leveraging inferences suggested by the inferences. The specific examples described briefly after each table show that there were no contradictions that would violate the framework. A test of the sufficiency condition was, therefore, met by the analyses reported.

4.3. Summary

The analysis revealed that both the necessary and sufficient conditions were met by the analyses performed on the repository. In particular, no contradictions were found except the two metrics that used a temporal dimension meeting the necessary condition. Further, expectations suggested by the inferences were evident in the metrics contained in the repository meeting the sufficient condition. One somewhat disturbing finding from the analysis, however, was that a number of metrics were unrelated, that is, the kinds of relationships posited by the framework were evident in only a small number of cases. No incorrect relationships were, however, revealed by the analysis providing further confirmation that the sufficient condition was met. The paucity of interconnectedness among metrics also suggested that the underlying metrics space (Metrics Space) is sparsely populated indicating the potential to discover additional metrics. The framework, therefore, appears to
provide a concise and expressive definition of the underlying metrics space that is only partially realized by the metrics proposed by the researchers.

5. Concluding remarks

In this paper, we have proposed a framework that captures the generic structure of the underlying product metrics space (Metrics Space) for object-oriented systems. The framework is based on the mereological and set theoretic perspective and uses relational measurement theory. We have outlined the framework as a set of definitions, axioms, and inferences that allow use and validation of the framework. Use of the framework with a repository containing about 350 metrics has allowed us to verify necessary and sufficient conditions for validation that are clearly met by the framework.

The framework does have certain limitations. First, it currently covers only product metrics, but not process metrics. Second, the framework can only be used to classify existing metrics or prescribe variations of existing metrics; it cannot self-describe the entire possible Metrics Space itself. In other words, unless a seed metric is used such as LOC(m), the framework cannot generate the logical variations such as avg(LOC(m)). And third, while the framework can be used to generate a large number of variations of metrics, this does not mean that the metrics are indeed meaningful or useful as predictors. Finally, the framework is not intended to replace research related to analyses of individual metrics such as characterizing behaviors of metrics in response to ranges assumed by input parameters or complexity analyses of existing metrics (e.g., [36]).

The proposed framework has a number of implications for research. First, it allows researchers to examine new metrics for their compliance with the framework. Since the framework is built on a sound theoretical basis and has been validated with the existing metrics, conformance to the framework should add considerable weight to new metrics. Second, the different inferences for the framework can help researchers and practitioners to automatically generate variations of existing metrics. This is, in fact, a key finding from the analysis. Such generation can reduce the burden of creating new metrics (following recent debates such as those seen on [38]), allowing researchers to analyze proposed metrics for their predictive potential [17]. For example, any method-level metric may propagate in cumulative form at the class and system level. While the sum frequently has no meaning, i.e., sum of local variables in a method has little meaning at the class level, the average often does. The framework can, therefore, also be used to channel research efforts towards validating existing metrics.

The framework also allows researchers to analyze different metrics based on the metrics/properties dimensions of the framework. For example, the relationships between entities, properties, and metrics can be used to group related metrics that measure the same property, allowing investigation of appropriateness of metrics, and cost/benefit analyses such as the cost of computations versus benefits derived from their use. The framework can also indicate appropriateness based on what can be calculated during different development stages e.g., design stage (i.e., number of methods) vs. the implementation stage (i.e., lines of code) suggesting decisions that may be possible at different stages of the development process. It is also possible to envision use of the framework to group metrics based on what they measure. The framework provides an essential prerequisite to such assessment. Similar analyses of existing metrics appear elsewhere (e.g., [36]).

Another area where the framework can be useful is the use of compound metrics, which have been found to be of interest to practitioners. By clearly identifying the components of these metrics, researchers can concentrate not only on the overall metric, but also in more detail on their components. If, for example, a low average is found to be desirable, then an examination of the components of the metric reveals that this can be achieved by having either a smaller sum or a larger \( n \). Thus, even with only partial information of the real values of metric components, certain design recommendations can be made. The eventual use of the framework may also include the categorization and grouping of existing metrics with a view to arriving at a ranking of existing metrics based on how well each measures a property.

Because product metrics play an important role in system development, the framework can also be useful for practitioners. By ensuring appropriate computation of measures of internal properties, the framework can

---

3 We estimate that no more than 25% of proposed metrics have been validated. See recent compilations of papers [38] for a sampling of research that proposes versus validates object-oriented and other metrics.
allow practitioners to focus on the issue of appropriate linkages between internal properties and external characteristics such as testability, reliability and maintainability, which makes them very important to practitioners at all levels [12,28]. Such external characteristics are visible to the users, and therefore, remain the goal for measurement practices, and visualization mechanisms [22]. They can, however, be obtained only when the information system is completed. Internal product metrics, on the other hand, cover properties visible to the development team and are thus obtainable during system development. They can therefore be used as leading indicators of the important external characteristics [27]. For example, if a relationship between a certain coupling metric and quality in terms of number of faults can be established, then one can reduce such faults and increase the product quality by minimizing coupling during design [12]. Other examples for successful applications of OO metrics include prediction, management of cost and resources of developing systems [13], testing [9,37], and maintaining [11,34]. Each of these uses can benefit from assurances of appropriate theoretical foundations for measurement of internal properties. The framework we have developed has addressed this goal.

References