Product Metrics for Object-Oriented Systems

SANDEEP PURAO

The Pennsylvania State University

AND

VIJAY VAISHNAVI

Georgia State University

We survey metrics proposed for object-oriented systems, focusing on product metrics. The survey is intended for the purposes of understanding, classifying, and analyzing ongoing research in object-oriented metrics. The survey applies fundamental measurement theory to artifacts created by development activities. We develop a mathematical formalism that captures this perspective clearly, giving appropriate attention to the peculiarities of the object-oriented system developmenr process. Consistent representation of the available metrics, following this mathematical formalism, shows that current research in this area contains varying coverage of different products and their properties at different development stages. The consistent representation also facilitates several analyses including aggregation across metrics, usage across metrics, equivalent formulation of metrics by multiple researchers, and exploitation of traditional metrics for object-oriented metrics. We also trace the chronological development of research in this area, and uncover gaps that suggest opportunities for future research.

Categories and Subject Descriptors: D.2.8 [**Software Engineering**]: Metrics—*Complexity measures, performance measures, product metrics*; K.6.3 [**Software Management**]: *Software development, software maintenance*

General Terms: Measurement, Management, Performance

Additional Key Words and Phrases: Software metrics, object-oriented systems, measurement theory, object-oriented metrics, object-oriented product metrics

1. INTRODUCTION

Over the last decade or so, object-orientation has become firmly established as the methodology of choice for developing new systems. Consequently, a significant share of efforts in research and industrial practice has focused on information systems development with the

The work of the second author was partially supported by a research course release granted by the Robinson College of Business.

Authors' addresses: S. Purao, School of Information Sciences and Technology, The Pennsylvania State University, University Park, State College, PA 16802; email: spurao@ist.psu.edu; V. Vaishnavi, J. Mack Robinson College of Business, Georgia State University, Atlanta, GA 30302; email: vvaishna@gsu.edu.

object-oriented paradigm. A key research area in this field has been measurement of and metrics for object-oriented systems [Lorenz and Kidd 1994; Henderson-Sellers 1996]. As differences between traditional and object-oriented system development have been identified, the need for metrics, distinct and relevant to object-oriented systems, has been recognized [Berard 1993]. Research in proposing, applying, validating, and extending metrics for object-oriented systems has reached a critical mass, as evidenced by several publications, presentations, and products (e.g., Lorenz and Kidd 1994; Henderson-Sellers 1996; Zuse 2001).

Much of the research, however, has been opportunistic and fragmented [Henderson-Sellers 1996; Bandi 1998; Briand et al. 1999], without regard for concerns such as completeness of coverage, and has resulted in unnecessary repetition of efforts. Without the benefit of a birds-eye perspective, researchers in different groups have continued to "discover" and "propose" similar metrics. For instance, Chidamber and Kemerer [1991], Chen and Lu [1993], Lorenz and Kidd [1994], Karlsson [1995], Tegarden et al. [1995] and De Champeaux [1997] all propose metrics that deal with related aspects of inheritance in object-oriented systems. On the other hand, Abreu and Carapuca [1994] and Karlsson [1995] propose metrics that have names that appear to be related ("polymorphism factor" and "polymorphism"), though they entail different computations and have different semantics.

The objective of this paper, therefore, is *to present a survey and an analysis of product metrics proposed for object-oriented systems so that research on the topic can be understood, classified, and analyzed*. The survey is organized around a framework and a mathematical formalism.[1] The framework is made up of two perspectives, internal and external, applied to the underlying phenomena—products, processes, and resources—arising from the object-oriented development activities. The internal perspective drives the structure of the mathematical formalism ensuring proper application of measurement principles. The external perspective allows interpretation and use of the metrics.

The focus of this paper is the internal perspective, following fundamental measurement theory, captured by our mathematical formalism. Using this mathematical formalism, we formulate uniform representations of existing metrics. We analyze these representations to discover disparate or related interpretations of similar concepts and to identify gaps in existing metrics, and we speculate on how our mathematical formalism can support the external (utilitarian) perspective.

The rest of the paper is organized as follows. Section 2 outlines the object-oriented metrics framework[2] elaborating on the object-oriented development process and the internal, measurement-theory-based perspective on metrics. Section 3 identifies entities arising out of object-oriented system development, lists attributes that were generated in a bottom-up manner, and explicates our representation formalism by demonstrating its application for a representative sample of metrics. The complete set of metrics identified from a review of the literature—represented with the formalism—is shown in the Appendix (available in the ACM Digital Library). Section 4 demonstrates usefulness of the representation formalism by performing several analyses and suggesting a number of others that may be useful. Finally, Section 5 draws conclusions and speculates on use of the formalism for a mapping against the external, utilitarian perspective.

[1] These organizing principles are developed for all object-oriented metrics. The listing (in the Appendix, available online in the ACM Digital Library) and analysis of the existing literature focuses on product metrics, which includes approximately 375 metrics.

[2] The paper is not a survey of metrics frameworks (e.g., Abreu and Carapuca [1994]; Tegarden et al. [1995]; Henderson-Sellers [1996]), though these were consulted during the development of our formalism for representing metrics.

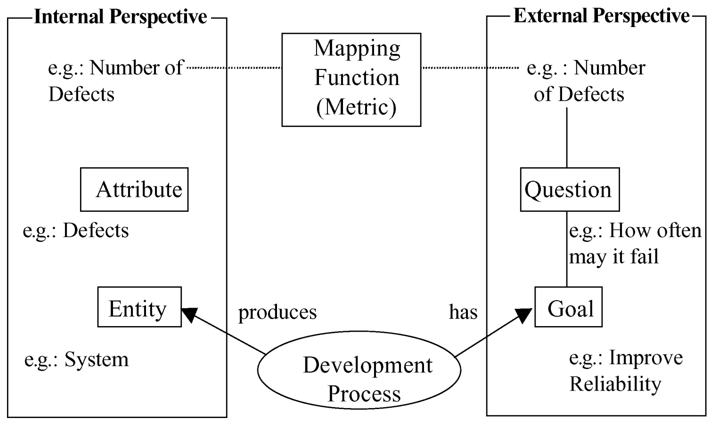

Fig. 1. An object-oriented measurement framework.

2. FRAMEWORK

We use the framework shown in Figure 1 to organize, understand, and classify the extensive research on object-oriented metrics. The framework is based on two distinct perspectives about *object-oriented development activities*, which give rise to products, execute processes, and expend or reuse resources. The first perspective, based on measurement theory [Roberts 1979], is the *internal perspective*. It operationalizes the representational or the entity-attribute-metric view [Fenton and Pfleeger 1997] to ensure a sound scientific basis for each *metric*. The other perspective, *external perspective*, is based on the purposive or the goal-question-metric paradigm [Basili 1992] and allows the use of *metrics* in response to prediction or assessment needs of system development professionals.

Following the internal perspective, a metric, then, is a function that assigns a number or symbol to an entity in order to characterize an attribute or a group of attributes. The value of the metric is called a *measure*. Measurement is the process by which a metric is computed from attributes of entities in the real world using clearly defined rules [Fenton and Pfleeger 1997; Finkelstein 1984; Roberts 1979].

Rigor in software measurement requires separating the scientific basis for measurement from the eventual use of the resulting metric. The commonly held viewpoint that no metric is “valid” unless it is a good predictor can undermine this rigor. An analogy would be to reject the usefulness of measuring a person’s height on the grounds that it does not tell us anything about that person’s intelligence [Fenton and Pfleeger 1997]. The result is that potentially valid metrics of important internal attributes become distorted. Over the last few years, researchers have questioned the scientific basis of many software metrics [Baker et al. 1990; Fenton 1991, 1994; Zuse and Bollman 1989; Zuse 1991; Henderson-Sellers 1996]. A large body of literature on metrics has been criticized for not considering their basis in measurement theory [Fenton 1994]. Our focus in this paper, accordingly, is predominantly on this internal perspective. Such a perspective is necessary to ensure that the metrics are constructed with a proper scientific basis; however, it is not sufficient.

A corresponding, external view can add a utilitarian perspective—that is, interpreting measurements in light of the goals, following the goal-question-metric approach [Basili 1992]. Given the almost limitless variety of interpretations of goals and few validation efforts [Bandi 1998; Li and Henry 1993; Basili et al. 1996], we only demonstrate (in Section 5) a possible approach to GQM mapping based on educated speculations about a specific instance of the external perspective. Our primary focus, thus, is on the internal perspective of the framework.

2.1. Object-Oriented System Development

A meaningful approach to classifying object-oriented metrics can start with examining the underlying phenomena of object-oriented system development. There is general consensus in academe and practice that developing systems using the object-oriented paradigm requires new development approaches. This stream of thought can be traced to Brooks [1987], who argued for eliminating accidental (software) complexity and controlling essential complexity by matching the method and process to the application. With object-orientation, the accidental complexity can be minimized and the essential complexity can be better controlled if a process that is appropriate for this paradigm can be followed. Over the last few years, a number of researchers and practitioners have discussed and prescribed object-oriented software development approaches (e.g. Rational [2001]; Baudoin and Hollowell [1996]; De Champeaux [1997]; Booch [1994]; Jacobson et al. [1992]). One can discern two *recurring themes* in these approaches (also see Mili et al. [1995]): (a) an iterative-incremental process, and (b) reuse-based development.[3]

More broadly, a software development process is a model for "the progression from identification of a need in some application domain to the creation of a software product that responds to that need" [Blum 1994, p. 83]. It identifies development tasks, establishes expectations for intermediate deliverables, and suggests review and evaluation criteria. The two recurring themes identified above change, in several ways, the form and shape of development process for object-oriented systems, compared to the traditional development processes.

First, the distinction between analysis, design, and implementation often blurs in object-oriented system development. For example, a developer may deal with the concept of "class" during the design and construction as well as the deployment phases, refining the understanding through these successive phases. This is in contrast to earlier paradigms of development, which may have used, say, data flow diagrams during design, hierarchical charts during construction, and modules during deployment, that is, transforming the representations through the successive phases. Since the artifacts created at each stage go through refinement (instead of transformation) as the development of an object-oriented system progresses, this blurring is not only natural but inevitable. The iterations are, therefore, a key aspect of the development process. Instead of the traditional, linear, phase-oriented approaches, other forms such as micro-macro [Booch 1994], recursive-parallel [Berard, 1993], and iterative-incremental [McGregor and Korson 1995]—which suggest iteratively building a system in increments that represent increasing levels of functionality—become more appropriate for object-oriented system development.

2.2. An Internal Measurement Perspective

An internal measurement perspective provides the appropriate scientific basis for measurement of the underlying phenomena. It suggests that we consider artifacts produced via the object-oriented system development process as "things," similar to the ontological stance proposed by Wand and Weber [1995]. Metrics, then, represent measurements of properties of these things. A formal manifestation of this theory is the relational model [Marciniak 1994] (see Figure 2). It seeks to formalize our intuition about the way the world works. Since one tends to perceive the real world by comparing things in them, not by assigning numbers to them [Fenton and Pfleeger 1997, pp. 24–25], *relationships*, that is, *empirical observations*, form the basis of measurement. One, then, attempts to develop metrics that represent attributes of the entities we observe, such that manipulation of the

[3] We do not include in the survey those product metrics whose purpose is to measure reuse because of the paucity of such metrics at the present time.

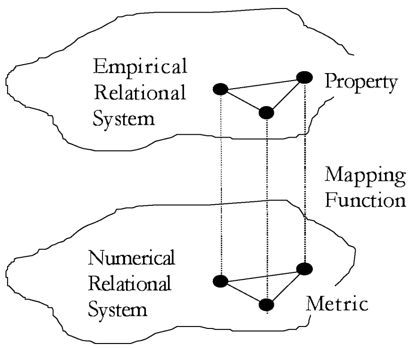

Fig. 2. The relational model.

metrics preserve *relationships* we observe among entities.

The set of entities, E, together with the set of empirical relations, R, is called an *empirical relation system* (E, R), defined with respect to a specific attribute. Measuring the attribute 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 [Fenton and Pfleeger 1997, p. 31]. For example, consider the binary relation #, which is mapped by M to the numerical relation !=. Then, formally, we can assert: $x \# y \Leftrightarrow M(x) \mathrel{!=} M(y)$. The real world is called the *domain* of the mapping, and the mathematical world is called the *range* [Fenton and Pfleeger 1997, p. 29].

A *metric*, then, is a function that assigns a number or symbol to an entity in order to characterize an attribute or a group of attributes; the value of the metric is called a *measure*. *Measurement* is the process by which a metric is computed from attributes of entities in the real world using clearly defined rules [Fenton and Pfleeger 1997; Finkelstein 1984; Roberts 1979]. Following measurement theory, metrics can be classified along two dimensions. The first represents the distinction between direct and indirect measurement. A *direct metric* is one that can be computed without any interpretation. An *indirect metric* is one that requires/involves interpretation [Rombach 1990]. The interpretation is applied to the computation of the "metric" instead of to the "attribute," since it is the metric that is "named" by the researchers. The second dimension distinguishes between elementary and composite metrics. An *elementary metric* is one that maps a single attribute of an entity to a number or symbol; a *composite metric* is one that requires a (mathematical) combination of several elementary metrics for one or multiple entities [Abreu et al. 1995]. Figure 3 summarizes these two dimensions.

Following the discussion of object-oriented system development (Section 2.1), another distinction—between syntax-based (i.e., static analysis, e.g., average number of methods per class) and execution-based (i.e., dynamic analysis, e.g., average frequency of method execution) metrics is also possible [Barnes and Swim 1993]. Metrics in the first category are functions that can be computed for artifacts without reference to the execution environment, whereas those in the second category can only be computed with the knowledge of runtime behavior. Separate consideration of this dimension is not necessary in our framework since a mapping to the underlying phenomenon—system development process—captures this aspect.[4]

3. METRICS

3.1. Entities and Attributes for Object-Oriented Systems

Bush and Fenton [1990]; Rombach [1990]; Ince [1990], and Fenton [1991, 1994] have proposed a classification of entities in software engineering that are amenable to measurement in three distinct categories: *Processes*, *Products*, and *Resources*. Processes are software-related temporal

[4] Yet another dimension, objective versus subjective metrics, deals with the measurement technique. Objectivity means precisely defined and repeatedly obtainable results, regardless of collector or time; subjectivity means depending either on the collector's judgment or use of arbitrary procedures. Subjectivity is a larger concern with the external, utilitarian, perspective.

	Direct Metrics that involve no interpretation	**Indirect** Metrics that require/involve interpretation
Composite Measuring multiple attributes from one/more entities	instance_methods_in_class() = public, private, and protected methods available to instances [Lorenz and Kidd 1994]	requirements_quality() = a complex formula requiring judgment [Karlsson 1995]
Elementary Measuring single attribute of an entity	classes() = number of classes [Jacobsen et al. 1993]	fraction_class_attribs_in_class() = fraction of attributes defined as class attributes [Karlsson 1995]

Fig. 3. Dimensions for classifying metrics.

activities; Products are artifacts, deliverables or documents that arise out of these processes; and Resources are elements consumed by the processes. Specific instantiations in each of the above three categories can differ across different situations. For example, for a large system, the entity Subsystem would be a meaningful entity, though it may not be so for a smaller system. Further, the terms used for different entities may also differ. For example, the term *Link* may be used in some cases to denote all associations such as relationships, inheritance, or aggregation links. On the other hand, each of these may be considered separate entities in another case. Such varying usage is also observed in the different metrics proposed by researchers. To ensure a consistent representation of the metrics, our first task was to arrive at a comprehensive superset of entities.

To achieve this, we iterated using a combination of top-down and bottom-up approaches. The top-down approach was characterized by an examination of the iterative-incremental development lifecycle as the basis for identifying the entities of interest (see Section 2.1). The products and processes of the lifecycle along with an identification of resources consumed in these processes provided us with the first set of entities. These results were supported by the bottom-up approach, which was characterized by a thorough survey of metrics proposed for object-oriented systems—adjusting the results to reflect any additional entities considered by the researchers, which were not identified by the top-down process. Iterating through this combination of approaches resulted in a set of entities, that forms the basis for our analysis. Following the scope of the paper, the discussion in this and the following sections focuses on metrics for entities in the Product category.

Another aspect that emerged during this combination of approaches was the importance of the different lifecycle stages for characterizing the entities in the Product category. Due to the incremental nature of the object-oriented development process, and the progressive refinement paradigm (see Section 2.1), products in object-oriented systems go through few transformations. Instead, they are subjected to *refinements* and progress from requirements to specifications to implementation and then into an operational state. Consider, for instance, the entity Class, which may be captured as a *requirement*, defined as part of a *design* specification, coded as part of an *implementation*, and run as part of an *operational* system. At each of these stages, it provides different opportunities for measurement.

We observed that most researchers have relegated this aspect to a discussion of "applicability" of a proposed metric at different stages. We argue that this is an important aspect of the definition of an entity, as it clarifies the measurement possibilities for an entity as it proceeds through the different phases. The appropriate mechanism to represent this is to characterize the different

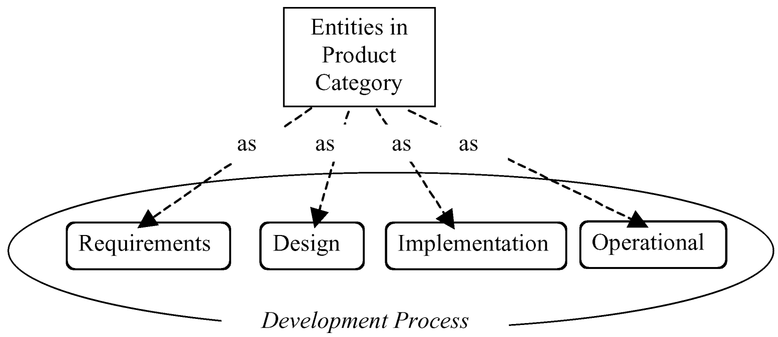

Fig. 4. States of products.

Table I. Entities for Object-Oriented Systems

Entities		States			
Abbreviation	Name	Requirements	Design	Implementation	Operational
Association	Association				
A	Attribute				
C	Class				
Event	Event				
Hierarchy	Hierarchy				
Link	Link				
E	Message				
M	Method				
Package	Package				
P	Parameter				
Scenario	Scenario				
B	Subsystem				
S	System				
U	Use Case				

Note: Grey cells denote the possible states for which the entity may have metrics.

lifecycle stages as "states" that the entity may exist in. Continuing with the example, the entity Class may exist in all states—requirement, design, implemented, or operational—whereas the entity Use Case may exist in the states requirement and design. We have identified four generic states to which the entities may be mapped. These are: Requirements, Design, Implementation, and Operational. Figure 4 shows the states[5] identified (for entities in products) defined as follows:

—*Requirements*: The expression of user needs about an artifact (e.g., a system) that is yet to be designed or constructed.

—*Design* (i.e., specifications): The set of specifications that captures, in the form of a blueprint, the user requirements, based on which, the construction of the artifact will proceed.

—*Implementation* (i.e., code): The artifact after it has been constructed and built, ready for deployment in its intended environment.

—*Operational*: The artifact after it is deployed so that it is functioning in the intended environment, such as an organization.

Mapping the entities against the states identified above provides us with a possibility matrix, suggesting that possibly different measurements about these entities may be performed in different states. Table I shows the entities and the abbreviation for entities in the category Product

[5] After a system is deployed, it can be in a maintenance (evolution) state. Clearly, metrics are important for this state as well. However, to keep the paper within manageable limits, we do not consider the metrics for this state.

and the possibility matrix, mapping the entities against the states, with the appropriate cells marked to indicate the measurement possibilities.

Attributes for these entities were identified next. We identified these primarily through a bottom-up analysis of the metrics available in the literature. We first identified the set of all the attributes used in the metrics. We then augmented this set by identifying missing attributes through a comparison of the attributes of different entities. For example, if an attribute of the entity System was *structure*, a similar attribute may have been meaningful for the entity Class. If such an attribute was not revealed by the bottom-up analysis for the entity Class, it was added. A number of attributes also fell into clusters following these steps. For example, there were several subattributes related to the attribute *position in hierarchy* for the entity Class. These included *impact on subclasses*, and *impact of superclasses*. The process iterated across differet entities with the addition, deletion, refinement, or renaming of attributes for each in light of decisions made for the other entities. The attributes shown in Table II are the result of this iterative process.

The entities and attributes provided the first two elements of the mathematical formalism for representing the metrics. Section 3.2 describes this formalism.

3.2. A Mathematical Formalism

The mathematical formalism we have developed provides a logical and coherent language for representing the diverse measures proposed by researchers. The formalism follows many of the suggestions put forward by Henderson-Sellers [1996], such as consistency of naming. In general, if a symbol, say *A*, is used to represent one concept of a variable (say, Attribute), then (i) it should always represent that variable and no other and (ii) only this symbol and no other symbol should be used for this particular concept. Such consistency supports both rigor and communication [Henderson-Sellers 1996]. Stylistically, communication and understanding are also improved if symbols referring to a consistent set of concepts or interpretations are similar [Henderson-Sellers 1996].

The following decisions support the expressiveness of our formalism. We assign each generic metric a name that includes the mathematical operator(s) used in the defintion of that metric. The generic metric is also expressed as a computable formulation. The formulation is based on the following decisions. To denote a set of entities, we use a symbol derived from the first letter, where possible, for example, C for class or S for system. To indicate a restricted population of the set, we use the set followed by a qualification, for example, A_inherited represents the attributes inherited. An element of the set is indicated by a lowercase representation (a for an attribute or c for a class). If multiple elements must be represented, we use a count indicator, for example, a_i in A indicates considering each attribute in the set of attributes. The prefix notation is used for the formulation, for example, C_descendents(c) indicates classes that are descendents of a class, c. In addition, we apply primitive operators, for example, |X| to indicate cardinality, and other operations such as sum, avg, and fraction. These expect the usual arguments and return the usual results, for example, 'sum' expects multiple arguments and returns one value. Other than such summative operators, a function returns a single value if the argument is an element, and a set of values if the argument is a set.

The computation of each metric is, therefore, expressed as a combination of mathematical *operators* on *values* assigned to *attributes* of *entities*. These are specified as arguments, as elements or sets, to functions. If the metric requires a nonprimitive operation, it is specified as a *function*(), for example, *fan-out*(), and, where possible, is mapped to the primitive operators.

Finally, the mapping of entities to states is indicated by the notation <as> following the entity. If an entity can be in multiple states for a measure, all states are

Table II. Attributes for Product Entities in Object-Oriented Systems

Entity		Attribute	
Abbreviation	Name	Name	Subattribute
Association	Association	size	
A	Attribute	interaction	
		position	impact of superclasses
		position	impact on subclasses
		position	impact of super plus impact on sub
C	Class	abstractness	
		Behavior	
		comments	
		effort	
		interaction	object level
		interaction	internal
		interaction	class level
		interface	
		performance	
		position	relative in structure
		position	changing the impact of super
		position	changing the impact in sub
		position	impact of superclasses
		position	impact on subclasses
		reuse	with respect to other classes
		reuse	internal
		size	
		structure	with respect to other classes
		structure	Aggregate
		structure	Arrangement
Event	Event	size	
Hierarchy	Hierarchy	structure	Arrangement
Link	Link	arity	
E	Message	size	
M	Method	abstractness	
		effort	
		interaction	instance level (similar to object level)
		interaction	definition level (similar to class level)
		Interface	
		Performance	
		Position	impact of superclasses plus impact on subclasses
		Position	changing the impact from superclasses
		Position	impact on subclasses/methods of subclasses
		Position	impact of superclasses/methods of superclasses
		Position	changing the impact in subclasses
		reuse	with respect to other methods or classes
		size	
		size	relative to class
		structure	Arrangement
		structure	Aggregate
		structure	
Package	Package	abstractness	
		interaction	
		size	
		structure	
P	Parameter	size	
Scenario	Scenario	size	
B	Subsystem	interaction	
		interface	
		size	
		structure	Aggregate
		structure	Arrangement
S	System	behavior	
		change	
		comments	Aggregate

Continue

Table II. *Continued*

Entity		Attribute	
Abbreviation	Name	Name	Subattribute
		doc	Aggregate
		dynamics	
		effort	
		interface	
		performance	
		requirements	
		reuse	from external repository
		reuse	Internal
		reuse	Result
		size	Aggregate
		size	
		structure	Aggregate
		structure	Arrangement
U	Use Case	interface	
		size	
		structure	

Table III. The Representation Formalism

<table>
<tr><th>Generic Form</th><th>Computable Formulation</th></tr>
<tr><td>Entity/Attrib/Name()</td><td>[Primitive_Operation] [Defined_Operation] (argument[s]) [Constraints]</td></tr>
<tr><th colspan="2">Examples</th></tr>
<tr><td>System/Structure/max_inherit_breadth()</td><td>max (|C_at (level_i) |), level_i in Level(s)</td></tr>
</table>

Notation	Sets and Elements	Examples	
A	set of all elements	A	set of all attributes
A_qualification	a subset of elements	A_inherited	set of inherited attributes
a	an element of the set	a	an attribute
a_i	a countable element	a_i	a specific attribute

<table>
<tr><th>Prefix Notation</th><th>Meaning</th></tr>
<tr><td>P(m)</td><td>parameters of a method</td></tr>
<tr><td>C_usedBy (c)</td><td>classes used by a class</td></tr>
<tr><td>sum (|T_sendMessage(M(c))|)</td><td>sum of send statements in methods of a class</td></tr>
<tr><td colspan="2">Notes:
function (x) = result obtained by applying function to element x.
function (X) = set resulting from applying that function to each element of X.
the function is read “of” unless specified otherwise, e.g., C_usedBy(m).</td></tr>
</table>

<table>
<tr><th>Notation</th><th>Primitives</th><th colspan="2">Examples</th></tr>
<tr><td>|x|</td><td>Non-Unique Cardinality</td><td>|P(M(c))|</td><td>Number of parameters of methods of the class</td></tr>
<tr><td>avg</td><td>Average</td><td>avg (|A(c_i) |) s.t. c_i in C(s)</td><td>Average number of attributes per class for the system</td></tr>
<tr><td>sum</td><td>Summation</td><td>sum (|M(c)|, |M_calledBy(c)|)</td><td>Sum of methods of the class and methods called by the methods of a class</td></tr>
<tr><td>fraction</td><td>Fraction</td><td>fraction (|C_super(s)|, |C(s)|)</td><td>Fraction of classes that are superclasses in the system</td></tr>
</table>

indicated. For example, C <as> Implementation would indicate that the entity is being considered in the state Implementation. Similarly, C <as> Design or Implementation would indicate that the entity is being considered in the state Design or Implementation. Finally, C <as> Design and Implementation would indicate that entity being considered is Class, in the states Design and Implementation, since computation of the metric requires information from both states. The four states outlined earlier and shown in Figure 4 form the basis of this mapping. Table III shows the representation formalism.

Table IV. Representation of Sample Metrics

Generic Metric Name expressed as Entity<as> State(s)/Attribute/Metric	Metric Formulation expressed using the Standard Formalism	Proposed by		Elementary	Direct
Method<as>design or implementation/ abstractness/methods_connected()	|M_connectedTo (m)|	Nakanishi et al.	1995	y	n
Method<as>design or implementation/ interaction/method_users()	|DirectUsersOf (m)|	Jacobson et al.	1992	n	y
Association<as>design or implementation/size/arity()	Arity (association)	De Champeaux	1997	y	y
Class<as>design or implementation/ position/ancestors()	|C_ancestors (c)|	Abreu and Carapuca	1994	n	y
Class<as>design or implementation/ position/ancestors()	|C_ancestors (c)|	De Champeaux	1997	n	y
		Li	1998	n	y
		Tegarden et al.	1995	n	y
Class<as>design or implementation/ position/children()	|C_children (c)|	Abreu and Carapuca	1994	y	y
		Chidamber and Kemerer	1994	y	y
		Marchesi	1998	y	y
Class<as>design or implementation/ structure/sum_attrib_chenlu_value()	sum(ChenLuValue (A (c)))	Chen and Lu	1993	y	y
Use Case<as>design/interface/ classes_offering()	|C_offerring (useCase)|	Jacobson et al.	1992	y	y
System<as>design or implementation/ reuse/classes_reused_by_inheritance()	|C_parents_in_r(C(s))|	Bieman and Karunanithi	1995	y	y
System<as>design or implementation/ reuse/classes_related_to_classes_ reused_by_inheritance()	|C_relatedTo(C_parents_in_ r(C(s)))|	Bieman and Karunanithi	1995	n	n

Note: n = no, y = yes.

3.3. Representing O-O Metrics

There exists a rich body of literature on object-oriented metrics that includes research on object-oriented product metrics, the focus of this paper. Using the formalism outlined in Section 3.2, each metric available from the literature is re-represented as an ordered tuple with three elements <E, A, M>. The first is the entity (and its mapping to one or more states), which is explicitly identified following the set of notations shown in Table I. The second is the attribute, following the set of attributes presented in Table II. Finally, the metric itself is specified along with the manner of computation, following the representation formalism presented in Table III. The re-representation results in a reformulation of the metrics proposed by different researchers to a common language. For example, the metric proposed by Abreu and Carapuca [1994] is re-represented in the table as Class<as>design or implementation/position/ancestors( ) and its computation is shown as |C_ancestors(c)|. The predetermined set of symbols for denoting different entities has provided a common language for the representation, and the format for elements, sets, and different operations outlined earlier has provided a grammar for this representation. Then, the application of the two has resulted in a uniform representation of the generic measures addressing a need identified by Henderson-Sellers [1996].

To present the results, we have adopted a tabular format. The metrics have been grouped following a hierarchy of entities and attributes. A specific generic metric may map to one or more metrics proposed by different researchers. Information on the origin of the metric is preserved in the presentation as secondary information about the metric. Table IV shows a representative sample of measures represented using the above scheme. Observe that the

Table V. Number of Metrics by Entity-Attribute

Entity	Number of Metrics	Attribute	Number of Metrics
Association	1	size	1
Attribute	4	interaction	2
		position	2
Class	164	behavior	8
		comments	3
		interaction	46
		interface	7
		performance	5
		position	23
		reuse	8
		size	10
		structure	54
Event	1	size	1
Hierarchy	2	arrangement	2
Link	0	arity	0
Message	0	size	0
Method	36	abstractness	1
		interaction	14
		interface	1
		performance	2
		position	5
		size	9
		structure	4
Package	5	abstractness	1
		interaction	3
		structure	1
Parameter	0	size	0
Scenario	3	size	3
Subsystem	2	structure	2
System	134	change	1
		dynamics	12
		effort	10
		interface	1
		performance	1
		requirements	5
		reuse	27
		size	19
		structure	57
		unable_to_determine	1
Use Case	3	interface	1
		size	2

information in the Elementary and Direct columns classify the metrics according to the classification scheme shown in Figure 3.

The complete tables along with additional information are presented in an appendix, available in the ACM Digital Library. All the information compiled for the available metrics is stored in a repository, which facilitates and automates its processing for further analysis. For example, all the analysis tables shown in Section 4 are generated from this repository using SQL queries.

4. COVERAGE, ANALYSIS, AND TRENDS

4.1. Coverage

The metrics compiled can be viewed from several perspectives to assess coverage achieved by prior research. We develop and discuss three such perspectives.

4.1.1. Coverage of Entities and Attributes. The first analysis summarizes the proposed metrics using the entity-attribute dimension. Table V shows this summary. The table consists of entities and attributes for these entities arranged vertically, against which is shown a count of metrics proposed for each entity-attribute pair.

The most striking feature of the table is the number of cells for which very few metrics have been proposed. Though information on Process and Resource metrics is beyond the scope of this paper, we note that the paucity of metrics is even more severe for those categories.

4.1.2. Availability of Metrics at Development Stages. One approach to addressing the paucity of Process-based metrics is to reconsider the Product-based metrics for their applicability to different states. This provides the second perspective for our assessment of coverage. For example, a metric such as Number of Classes can be used at different states: Requirements, Design, Implementation, and Operational. Measuring the same attribute at different states can, then, be used as a surrogate for applicability to different stages of the development process. Following the arguments about the development processes for object-oriented systems (see Section 2.1), the refinement in products (instead of transformation) can, therefore, be the basis for measuring the process. These metrics can be the basis for measuring the progress through phases as well as iterations of the development process. Further, some metrics are applicable only in certain states. These can be useful for changes observed across increments (versions). For example, measuring Number of Classes from one version of the implemented system to another can provide an indication of the functionality realized during the increment. Table VI shows the availability of metrics for entities in the category Product across different states.

The table shows the count of metrics for each entity in the Product category mapped against the states. For example, the entity Association may be measured during the Design and Implementation states. Metrics that are applicable to multiple states are also indicated against each entity, indicating the combination of states for which these metrics are applicable. Note that the total across columns, for a row, may not match the corresponding number in Table V due to the possible appearance of a metric in multiple cells along the row.

Table VI indicates that the number of metrics proposed for the Design, and Implementation states far outweigh those proposed for either the Requirements or Operational states. One possible explanation for this bias may be the relative ease with which metrics at the former states can be computed compared to those at the latter two states. Clearly, additional metrics are needed at the Requirements and Operational states. A significant number of metrics apply to multiple states, for example, Design as well as Implementation. This is supported by the nature of the development process for object-oriented systems, which is one of iterative refinement rather than transformation. Metrics that utilize information available in multiple states provide another opportunity. However, few such metrics have been proposed. With the paradigm of refinement for the development process, it would be reasonable to expect that more metrics would be suggested that compare similar properties measured across multiple states. This represents another major shortcoming of existing research.

4.1.3. Classification Based on Elementarity and Directness. Finally, a third perspective is afforded by the nature of the metric following the classification scheme proposed in Figure 3. This scheme classifies the metrics along two dimensions: direct versus indirect, and elementary versus composite. In Table VII, we provide a summary of the metrics proposed along these two dimensions. The table shows a count of metrics for each entity—attribute, mapped against the values of the two classification dimensions.

The table shows that the proposed metrics are concentrated in the Direct-Elementary column (leftmost column)

Table VI. Availability of Metrics at Different States

Entity	Attribute	1	2	3	4	1 or 2	1 or 3	2 or 3	3 or 4	1 or 2 or 3	1 or 3 or 4
Association	size							1			
Attribute	position							2			
	interaction							2			
Class	performance			1	3				1		
	structure			13				39			
	size			1				9			
	position			6				16			
	interface		1					5			
	interaction		1	17				22		1	
	comments			3							
	behavior		7					1			
	reuse			1				1	1		
Event	size		1								
Hierarchy	arrangement							2			
Link	arity										
Message	size										
Method	performance				1				1		
	Interface		1								
	abstractness							1			
	position							5			
	size			6				3			
	structure		1	2				1			
	interaction		1	1				12			
Package	abstractness							1			
	interaction		1					2			
	structure		1								
Parameter	size										
Scenario	size		2			1					
Subsystem	structure							2			
System	dynamics			1				9			
	structure		6	7				42			1
	size	1	3	4		3		8			
	reuse			2				22			
	requirements	5									
	effort			3				1			
	change						1				
	performance				1						
Use Case	size					2					
	interface		1								

Legend: 1 = Requirements, 2 = Design, 3 = Implementation, 4 = Operational.

or Indirect-Composite column (rightmost column). The concentration in the leftmost column is heartening since it indicates that researchers have implicitly followed good measurement principles, which clearly map to a single entity and a single attribute. The focus also represents an opportunity. The elementary metrics can be combined in several ways to derive additional, interesting metrics that could become the basis of mappings against external concerns. The concentration in the rightmost column appears problematic as it indicates that several metrics are being proposed that may lack a sound measurement basis, particularly as their measurement inherently requires interpretation from the developers.

4.2. Analysis

The representation formalism allows several paths for analyzing the metrics. In this section, we demonstrate a few of these possibilities. These do not constitute an exhaustive list. Several additional comparisons and analyses are possible with the consistent representation scheme we have suggested.

4.2.1. Aggregations Across Metrics. Aggregation reflects how object-oriented

Table VII. Number of Metrics by Metric-Type

Entity	Attribute	Direct		Indirect	
		Elementary	Composite	Elementary	Composite
Association	size	1			
Attribute	interaction			2	
	position			2	
Class	behavior	5		1	2
	comments	3			
	interaction	4	4	10	28
	interface	2	1	1	3
	performance				5
	position	6	4	1	12
	reuse	4	1		3
	size	5	1	1	3
	structure	14	2	8	30
Event	size	1			
Hierarchy	arrangement				2
Link	arity				
Message	size				
Method	abstractness			1	
	interaction	5	4	3	2
	interface				1
	performance	2			
	position	2		2	1
	size	5	1	1	2
	structure				4
Package	abstractness				1
	interaction	1		2	
	structure	1			
Parameter	size				
Scenario	size	1	2		
Subsystem	structure				2
System	change			1	
	dynamics				12
	effort		9		1
	interface				1
	performance	1			
	requirements	2			3
	reuse	10	2	2	13
	size	9	3	2	5
	structure	21	13	7	16
	unknown		1		
Use Case	interface	1			
	size	2			

artifacts are structured, for example, how methods and attributes "aggregate" to become classes. Aggregation across metrics reflects the use of a summative operator (sum, average, etc.) to define a metric for an entity from the corresponding metrics for its constituent entities. For example, the metrics avg_inherit_depth() and max_inherit_depth() for the entity System represent different aggregations of the metric depth_from_root() for the entity Class. As individual Classes are aggregated into a System, properties of (i.e., metrics about) Classes can be aggregated into properties of (i.e., metrics about) Systems. A number of existing metrics can be mapped against one another in this manner. In fact, a number of metrics for the entity System do represent aggregations of metrics proposed for the entity Class. Table VIII shows a mapping of existing metrics that are related to each other via such aggregation. It shows the two metrics related to each other in this manner, and shows the computation of one that involves aggregating the other. As can be seen from the results, a number of metrics do, in fact, represent aggregations of other metrics. Such aggregation, however, assumes that

Table VIII. Aggregation Across Metrics

Metric 1	Metric 2	Metric 1 as Aggregation of <Metric 2>
System/structure/sum_association_arity()	Association/size/arity()	sum (<arity(association_i)>) s.t. association_i in Associations(s)
System/structure/sum_association_arity_and_classes()	Association/size/arity()	sum (<classes(s)>, sum <arity(association_i)> s.t. association_i in Association(s))
System/structure/avg_instance_methods()	Class/size/instance_methods()	avg (<instance_methods(c_i)>) s.t c_i in C(s)
System/structure/avg_instance_attributes()	Class/size/attributes()	avg (<attributes(c_j)>) s.t. c_i in C(s)
Class/size/sum_attributes_and_methods_defined()	Class/size/attributes()	sum (<attributes(c)>, <methods(c)>)
Class/reuse/avg_global_usage_per_method()	Class/size/methods()	(sum |ReferencesToGlobal(s(c))|, <class_attribs(c)>)/(<methods(c)>)
Class/size/sum_attributes_and_methods_defined()	Class/size/methods()	sum (<attributes(c)>, <methods(c)>)
Class/interface/sum_methods_in_and_methods_called()	Class/size/methods()	sum (<methods(c)>, |M_calledBy(c)|)
Class/interaction/avg_nonlocal_refs()	Class/size/methods()	(<sum_method_nonlocalrefs(c)/(<methods(c)>)
System/structure/avg_inherited_methods()	Class/size/methods()	avg (<methods_inherited (c_i(s))>) s.t. c_i in C(s)
Class/interaction/sum_methods_sending_messages_to()	Class/size/methods()	sum (<methods_sending_messages_to(m_i)>) s.t. m_i in M(c)
System/structure/avg_class_methods()	Class/size/class_methods()	avg (<class_methods (c_i)>) s.t c_i in C(s)
Class/interaction/sum_message_exchanges()	Class/size/messages_received_by_class()	sum (<messages_sent (O(c))>, <messages_received_by_class (O(c))>)
System/structure/sum_syntax_halstead_complexity()	Class/structure/sum_method_complexity()	sum ((Halstead_complexity(m_i) * <sum_method_complexity(m_i)>)/sum (usability(s))) s.t. m_i in M(C(s)), no computable formulation of usability() provided
System/structure/avg_class_responsibilities()	Class/structure/sum_class_responsibilities()	avg (<sum_class_responsibilities(c_i)>) s.t. c_i in C(s)
Class/interaction/sum_external_interaction()	Class/structure/public_methods()	sum (sum (<public_methods(c)>, <public_attribute(c)>), sum (|CallsTo(M(c_i))|, |UsageOf(A(c_i)|))) s.t. c_i in C(s(c)), c_i not equal to c
System/structure/avg_inherit_depth()	Class/position/depth_from_root()	avg (<depth_from_root (c _i)>) s.t. c_i in C(s)
Class/position/max_inherit_depth()	Class/position/depth_from_root()	max (<depth_from_root (c)>)
System/structure/sum_inheritance_complexity()	Class/position/depth_from_root()	sum (internalComplexity(M(c_j)*(1-relativeOrder(c_j))) * <depth_from_root(c_i)>, <children(c_i)> s.t. c_j in C_ancestor(c_i), c_i in C(s) where computable formulations of internalComplexity() and relativeOrder() are not provided.
System/dynamics/sum_interaction_complexity()	Class/position/ancestors()	sum (complexity(c_i)*<ancestors(c_i)>, RFCmetricNumber(c_i), |P(M(c_i))|, |T_sendMessage(c_i)|) s.t c_i in C(s) where no computable formulation for complexity(c_i) is provided and conflicting formulation is provided for |P(M(c))|
System/structure/avg_inherited_methods()	Class/position/methods_inherited()	avg (<methods_inherited (c_i(s))>) s.t. c_i in C(s)
System/structure/sum_exponential_in_non_root_parents()	Class/position/parents()	sum (k^ (<parents(c_nonRoot_i)>-1)) s.t. c_nonRoot_i in (C(s)-C_root(s)), k >1

System/structure/sum_inheritance_complexity()	Class/position/children()	sum (internalComplexity(M(c_j)*(1-relativeOrder(c_j))) * <depth_from_root(c_i)>, <children(c_i)> s.t. c_j in C_ancestor(c_i), c_i in C(s) where computable formulations of internalComplexity() and relativeOrder() are not provided.
Class/interaction/sum_methods_sending_messages_to()	Class/interaction/methods_sending_messages_to()	sum (<methods_sending_messages_to(m_i)>) s.t. m_i in M(c)
Class/interaction/sum_message_exchanges()	Class/interaction/messages_sent()	sum (<messages_sent (O(c))>, <messages_received_by_class (O(c))>)
System/dynamics/sum_attrib_param_interaction_for_system()	Class/interaction/sum_attrib_param_interaction()	sum (<sum_attrib_param_interaction(c_i)>) s.t. c_i in C(s)
Repository/reuse/avg_reuse_frequency_per_class	Class/reuse/reuse_frequency()	avg (<reuse_frequency(c_i)>) s.t. c_i in C(repository)
System/interface/avg_weighted_sum_method_params()	Class/interface/weighted_sum_method_params()	avg (<weighted_sum_method_params(c_i)>) s.t. c_i in C(s)
Class/structure/avg_method_mccabe_complexity()	Method/structure/method_mccabe_complexity()	avg (<method_mccabe_complexity (M(c))>)
Class/structure/sum_class_length_params_interaction()	Method/structure/method_length_params_interaction()	sum <method_length_params_interaction(m_i)> s.t. m_i in M(c)
Method/interaction/sum_input_output_params()	Method/interaction/arguments_sent()	sum (<arguments_received(m)>, <arguments_sent(m)>)
System/dynamics/sum_attrib_param_interaction_for_system()	Method/interaction/sum_attrib_param_interaction()	sum (<sum_attrib_param_interaction(c_i)>) s.t. c_i in C(s)
Method/interaction/sum_input_output_params()	Method/interaction/arguments_received()	sum (<arguments_received(m)>, <arguments_sent(m)>)
Class/interaction/sum_methods_sending_messages_to()	Method/interaction/methods_sending_messages_to()	sum (<methods_sending_messages_to(m_i)>) s.t. m_i in M(c)
Class/interaction/sum_message_exchanges()	Method/interaction/messages_sent()	sum (<messages_sent (O(c))>, <messages_received_by_class (O(c))>)
Class/interaction/avg_nonlocal_refs()	Method/interaction/sum_method_nonlocalrefs()	(<sum_method_nonlocalrefs(c)/(<methods(c)>)
System/structure/avg_method_size()	Method/size/lines_of_code()	avg (<lines_of_code(m_i)>) s.t m_i in M(C(s))
System/structure/avg_inter_package_class_usage()	Package/interaction/sum_usage_of_classes_in_other_packages()	avg (<sum_usage_of_classes_in_other_packages(package_i)>) s.t. package_i in Packages(s)
System/structure/sum_association_arity_and_classes()	System/size/classes()	sum (<classes(s)>, sum <arity(association_i)> s.t. association_i in Association(s))
Class/interaction/sum_message_exchanges()	System/size/messages()	sum (<messages_sent (O(c))>, <messages_received_by_class (O(c))>)
System/size/sum_scenario_sizes()	Scenario/size/weighted_objects_and_interactions()	sum (<weighted_objects_and_interactions(scenario_i)>) s.t. scenario_i in Scenario(s)

Note: s.t. = such that.

the property being measured can, in fact, be aggregated. Some aggregate metrics, however, violate this assumption, for example, when sum_halstead_complexity() is computed for the entity System using sum_method_complexity() for each class as the input.

The notion of aggregation also presents an opportunity. Several metrics may be identified for whom such aggregations have not been proposed by other researchers. The inverse is also true. Based on aggregate measures suggested, the corresponding elemental measures can be derived. The natural aggregation relationships between entities, such as Method-Class and Class-System, therefore, clearly suggest opportunities for aggregating or decomposing metrics across these entities along the dimensions of artifact granularity and mathematical aggregation akin to those suggested by Talbi et al. [2001].

4.2.2. Usage Across Metrics. Usage across metrics represents the use of one or more existing metric definitions for defining a new metric. It represents a notion similar to the idea of composite metrics discussed earlier (see Figure 3). Often, such usage can occur when one metric definition (e.g., "number of methods") is used by another metric definition (e.g., "fraction of public methods"). The usage therefore, typically, occurs within the properties for a single entity as opposed to across entities. Several metrics use other metrics in a number of ways. Table IX shows this usage across metrics. It shows the computation that exploits another metric, indicated by <angular brackets>. For instance, the metric average_method_attribute_usage() utilizes the metrics <attributes(c)> and <methods(c)> in its computation. From Table IX, it is apparent that some metrics that represent basic counting such as number of attributes <attributes(c)> or number of methods <methods(c)> are used repeatedly in the computation of other metrics. This points to the need for accurate counting practices. For example, it is necessary to clearly specify the manner of counting number of methods by addressing nuances such as inherited methods, overridden methods, virtual methods, get and set methods, etc. Without clear application of such practices, these metrics are likely to lead to widely different results. A second observation from Table IX is the existence of several composite metrics that use more than one elementary metrics. For example, consider the metric relative_position(). It uses as many as four other metrics including <parents(c)>, <depth_from_root(c)>, <local_methods(c)>, and <methods_inherited(c)>. We contend that such usage of multiple elementary metrics is likely to be against the basic tenets of measurement principles. The example cited here, for instance, attempts to add the values from the above component metrics. The resulting metric, therefore, may not be able to measure a meaningful property of interest.

Analysis of metrics in this manner—usage in computation in other metrics—therefore, underscores the importance of some elementary metrics that are repeatedly used and suggests opportunities for investigating the appropriateness of other metrics that utilize multiple component metrics.

4.2.3. Equivalent Formulations of Related Metrics. A few metrics represent similar proposals by different researchers. The evident popularity of these metrics (similar proposals from multiple researchers) suggests that the metrics are either (a) important and recognized as such by multiple researchers, or (b) easy to compute and represent straightforward opportunities for counting. In general, we find that the latter has dictated the identification of similar metrics by multiple researchers. Table X shows these metrics. For each metric that is suggested by multiple researchers, we show the generic metric name, and the number of researchers who have proposed the metric. A qualification for inclusion in this table is that the related metrics follow identical or at least substantially similar formulations. The analysis highlights the importance of a common language for expressing the metric and a common repository for metrics.

Table IX. Usage Across Metrics

<table>
<tr><th>Metric 1</th><th>Metric 2</th><th>Computation of Metric 1 using <Metric 2></th></tr>
<tr><td>Class/behavior/attrib_and_method_polymorphism()</td><td>Attribute/position/classes_defined_in()</td><td>normalized_sum (<classes_defined_in (a)>, <classes_defined_in(m)>) - - “normalized_sum()” not defined</td></tr>
<tr><td>System/structure/fraction_attrib_hiding()</td><td>Class/size/attributes()</td><td>fraction (sum (|A_hidden (c_i)|), sum (<attributes(c_j)>)) s.t. c_i, c_j in C(s)</td></tr>
<tr><td>System/structure/fraction_attribute_inherit()</td><td>Class/size/attributes()</td><td>fraction (sum (|A_inherited (c_i)|), sum (<attributes (c_j)>)) s.t. c_i, c_j in C(s)</td></tr>
<tr><td>System/size/weighted_sum_attributes_methods()</td><td>Class/size/attributes()</td><td>weighted_sum (<attributes(c_i)>, <methods(c_i)>) s.t. c_i in C(s)</td></tr>
<tr><td>System/reuse/fraction_reusesavings_generalizationcosts()</td><td>Class/size/attributes()</td><td>fraction ((weighted_sum (A(c_i), M(c_i), negative(reuseCost(c_i)))) + weighted_sum (reuseFraction(c_j)*<attributes(c_j)>, reuseFraction(c_j)*<methods(c_j)>, negative(reuseCost(c_j)))), weighted_sum (<attributes(c_k)>, <methods(c_k)>) * generalizationCost(c_k))) s.t. c_i in C_reusedFromRepository(s), c_j in C_inheritedAndUsed(s), c_k in C_reusableProducedFrom(s)</td></tr>
<tr><td>Class/structure/weighted_sum_attribute_association_methods_rule_for_class()</td><td>Class/size/attributes()</td><td>sum (weight1*(<attributes(c)> + |Associations(c)|), weight2 * avg(<lines_of_code(M(c))>)/STDLOC * <methods(c)>, weight3 * |RulesPerRuleSet(c)| * avg(|AntecedentClausesPerRule(c)|) * |RuleSets(c)|) where STDLOC means agreed language-dependent standard LOC per method, other terms not fully defined, weights decided by designer</td></tr>
<tr><td>Class/structure/method_attribute_usage_function()</td><td>Class/size/attributes()</td><td>fraction (sum (|M_containing_m_i(c)|), (<attributes(c)> *|X|)) s.t. X = M_added(c) union M_inherited Rodefined(c), m_i in X.</td></tr>
<tr><td>Class/structure/average_method_attribute_usage()</td><td>Class/size/attributes()</td><td>fraction (sum (|M_using(a_i)|/<methods(c)>), <attributes(c)>) s.t. a_i in A(c)</td></tr>
<tr><td>Method/structure/method_length_params_interaction()</td><td>Class/size/attributes()</td><td>length(m) * square (coupling(m)) s.t. length(m) = |T_if(m)| + | T_case(m)| + <attributes(m)>; coupling(m) = sum (ip(m), op(m)) s.t. ip(m) = (1 + <arguments_received(m)> + void(m)) s.t. void(m) = 1 if m returns value and 0, otherwise; op(m) = (sum (|ReferencesTo_m_i (m)| * (1 + |P_referencedBy_m(m_i)| + void (m_i)) + sum (|ReferencesTo_a_j(m)|)) s.t. m_i in M_referencedBy (m), a_j in A_referencedBy (m)</td></tr>
<tr><td>Class/structure/class_length_interaction()</td><td>Class/size/attributes()</td><td>length(c) * square (coupling(c)) s.t. length(c) = sum (length(m_i)) s.t. m_i in M(c); coupling(c) = sum (ip(m_j) + (sum (r_j_k) + sum (r_j_l))) s.t. m_j in M(c), k = 1..NMR(c), l = NMR(c) + 1..NMR(c) + NVR(c), NMR(c) = |unique (M_referencedBy(M(c)))|, NVR(c) = |unique (A_referencedBy (M(c)))|, r_j_k = |ReferencesTo_m_k (m_j)| * (1 + |R_ referencedBy_m_j(m_k) + void(m_k), r_j_l + |ReferencesTo_a_ l(m_j)| s.t. length(m) = |T_ys(m)| + |T_case(m)| + <attributes(m)>; coupling(m) = sum (ip(m), op(m)) s.t. ip(m) = (1 + <arguments_received(m)> + void(m)) s.t. void(m) = 1 if m returns value and 0, otherwise; op(m) = (sum (|ReferencesTo_m_i (m)| * (1 +</td></tr>
</table>

Continue

Table IX. *Continued*

<table>
<tr><th>Metric 1</th><th>Metric 2</th><th>Computation of Metric 1 using <Metric 2></th></tr>
<tr><td></td><td></td><td>|P_referencedBy_m(m_i)| + void (m_i)) + sum (|ReferencesTo_a_j(m)|)) s.t. m_i in M_referencedBy (m), a_j in A_referencedBy (m)</td></tr>
<tr><td>Class/reuse/interface_attribute_method_meaning_and_access()</td><td>Class/size/attributes()</td><td>reusability (c) * ((sum (reusability (m_i) s.t. m_i in M(c)))/ <methods(c)>) + ((sum (reusability (a_j) s.t. a_j in A(c)))/ <attributes(c)>), s.t. reusability (c) = (description_meaningfulness (c) + name - meaningfulness (c))/(|C_coupledTo (c)| + <depth_from_root(c)> + 1), reusability (m_i) = (name_meaningfulness (m_i) + P_usedBy (m_i) + ((sum (reusability (p_k)) s.t. p_k in P(m_i))/ |p(m_i)|))/(|functions_performedBy (m_i) + callsTo - foreignClasses (m_i)|), reusability (p_k) = (name_meaningfulness (p_k))/ (|subTypesOf (typesOf(p_k)| + 1) or = 5 if|p(m_i)| = 0, reusability (a_j) = (name_meaningfulness (a_j))/(|subTypesOf (a_J)| + 1) s.t. name_meaningfulness() and description_meaningfulness() are decided on a scale 1..5 by the developer.</td></tr>
<tr><td>Class/structure/fraction_method_attribute_usage()</td><td>Class/size/attributes()</td><td>fraction ((sum (|M_accessing_a_i(c)|/<attributes(c)>) -|M|), (1 -|M|)) s.t. a_i in A(c) fraction((<methods(c)> - sum (|M_accessing_a_i(c)|)), |M(c)|) s.t. a_i in A(c)</td></tr>
<tr><td>Class/structure/fraction_private_attributes()</td><td>Class/size/attributes()</td><td>fraction (|A_private(c)|, <attributes(c)>)</td></tr>
<tr><td>Class/structure/fraction_object_valued_atrributes()</td><td>Class/size/attributes()</td><td>fraction (|A_objectValued(c)|, <attributes(c)>)</td></tr>
<tr><td>Class/structure/fraction_simple_value_attributes()</td><td>Class/size/attributes()</td><td>fraction (|A_value(c)|, <attributes(c)>)</td></tr>
<tr><td>System/structure/fraction_method_hiding()</td><td>Class/size/methods()</td><td>fraction (sum (|M_hidden(c_i)|), sum (<methods(c_j)>)) s.t. c_i, c_j in C(s)</td></tr>
<tr><td>System/structure/fraction_method_inherit()</td><td>Class/size/methods()</td><td>fraction (sum (<methods_inherited (c_i)>), sum (<methods(c_j)>)), s.t. c_i, c_j in C(s)</td></tr>
<tr><td>System/structure/fraction_overridden_methods_new_methods()</td><td>Class/size/methods()</td><td>fraction (sum (<methods_overridden (c_i)>), sum (|M_new (c_j)|) * sum (<descendants (c_j)>)) s.t. c_i, c_j in C(s)</td></tr>
<tr><td>Class/comments/fraction_commented_methods()</td><td>Class/size/methods()</td><td>fraction (|M_commented (c)|, <methods(c)>)</td></tr>
<tr><td>Class/position/relative_position()</td><td>Class/size/methods()</td><td>sum (<depth_from_root (c)>, <parents (c)>, <methods_inherited (c)>, |C_sub (c)|, <local_methods(c)>)</td></tr>
<tr><td>Class/position/fraction_methods_overridden()</td><td>Class/size/methods()</td><td>fraction((<methods_overridden (c)> * <depth_from_root (c)>), (<methods(c)>))</td></tr>
<tr><td>Class/position/fraction_overloaded_methods()</td><td>Class/size/methods()</td><td>fraction (<methods_Overloaded_from_repository (c)>, <methods_inherited (c)>)</td></tr>
<tr><td>System/size/weighted_sum_attributes_methods()</td><td>Class/size/methods()</td><td>weighted_sum (<attributes(c_i)>, <methods(c_i)>) s.t. c_i in C(s)</td></tr>
<tr><td>System/reuse/fraction_reusesavings_generalizationcosts()</td><td>Class/size/methods()</td><td>fraction ((weighted_sum (A(c_i), M(c_i), negative(reuseCost(c_i))) + (weighted_sum (reuseFraction(c_j)*<attributes(c_j)>, reuseFraction(c_j)*<methods(c_j)>, negative(reuseCost(c_j))), (weighted_sum (<attributes(c_k)>, <methods(c_k)>) *</td></tr>
</table>

<table>
<tr><td></td><td></td><td>generalizationCost(c_k))) s.t. c_i in C_reusedFromRepository(s), c_j in C_inheritedAndUsed(s), c_k in C_reusableProducedFrom(s)</td></tr>
<tr><td>Class/structure/weighted_sum_attribute_association_methods_rule_for_class()</td><td>Class/size/methods()</td><td>sum (weight1*(<attributes(c)> + |Associations(c)|), weight2 * avg(<lines_of_code(M(c))>)/STDLOC * <methods(c)>, weight3 * |RulesPerRuleSet(c)| * avg(|AntecedentClausesPerRule(c)|) * |RuleSets(c)|) where STDLOC means agreed language-dependent standard LOC per method, other terms not fully defined, weights decided by designer.</td></tr>
<tr><td>Class/structure/relative_param_usage_by_methods()</td><td>Class/size/methods()</td><td>(sum (|x_fraction(m_i)|))/(|x| * <methods(c)>) s.t. m_i in M(c), x = unique (union(typesOf (P_usedBy (m_i)))) s.t. m_i in M(c), x_fraction (m_i) = intersect (x, unique (typesOf (P_usedBy (m_i)))</td></tr>
<tr><td>Class/position/fraction_methods_inherited_function()</td><td>Class/size/methods()</td><td>fraction ((|M_virtualAdded(c)| + |M_virtualInherited(c)| + 2 * (<methods_added(c)> + |<methods_inherited(c)>), 2 * <methods(c)>)</td></tr>
<tr><td>Class/position/methods_specialized_function()</td><td>Class/size/methods()</td><td>avg (P + O + N) s.t. P = fraction (|M_inheritedRedefinedWithNewBehavior(c)|, |M_inheritedRedefined(c)|) O = fraction (|M_inheritedRedefined(c)|, <methods_inherited (c)>) N = fraction(level(c) * <methods_added(c)>, (<methods_added(c)> + 1) * (level(c) + 1))</td></tr>
<tr><td>Class/structure/average_method_attribute_usage()</td><td>Class/size/methods()</td><td>fraction (sum (|M_using(a_i)|/<methods(c)>), <attributes(c)>) s.t. a_i in A(c)</td></tr>
<tr><td>Class/reuse/interface_attribute_method_meaning_and_access()</td><td>Class/size/methods()</td><td>reusability (c) * ((sum (reusability (m_i) s.t. m_i in M(c)))/ <methods(c)>) + ((sum (reusability (a_j) s.t. a_j in A(c)))/ <attributes(c)>), s.t. reusability (c) = (description_meaningfulness (c) + name - meaningfulness (c))/(|C_coupledTo (c)| + <depth_from_root(c)> + 1), reusability (m_i) = (name_meaningfulness (m_i) + P_usedBy (m_i) + ((sum (reusability (p_k)) s.t. p_k in P(m_i))/ |p(m_i)|))/(|functions_performedBy (m_i) + callsTo - foreignClasses (m_i)|), reusability (p_k) = (name_meaningfulness (p_k))/ (|subTypesOf (typesOf(p_k)| + 1) or = 5 if|p(m_i)| = 0, reusability (a_j) = (name_meaningfulness (a_j))/(|subTypesOf (a_J)| + 1) s.t. name_meaningfulness() and description_meaningfulness() are decided on a scale 1..5 by the developer.</td></tr>
<tr><td>Class/structure/fraction_method_attribute_usage()</td><td>Class/size/methods()</td><td>fraction ((sum (|M_accessing_a_i(c)|/<attributes(c)>) -|M|), (1 -|M|)) s.t. a_i in A(c) fraction((<methods(c)> - sum (|M_accessing_a_i(c)|)), |M(c)|) s.t. a_i in A(c)</td></tr>
<tr><td>System/structure/fraction_functionality_distribution_in_inheritance_chain()</td><td>Class/size/methods()</td><td>fraction (<methods(c_i)>, avg(<methods(c_i)>) s.t. c_i in C(s)</td></tr>
<tr><td>Class/structure/class_versus_instance_attributes()</td><td>Class/size/class_attributes()</td><td>(<class_attributes(c)>/|A_instance(c)|)</td></tr>
<tr><td>System/structure/std_dev_class_responsibilities()</td><td>Class/structure/sum_class_responsibilities()</td><td>stddev (<sum_class_responsibilities(c_i)>) s.t. c_i in C(s)</td></tr>
</table>

Continue

Table IX. *Continued*

<table>
<tr><th>Metric 1</th><th>Metric 2</th><th>Computation of Metric 1 using <Metric 2></th></tr>
<tr><td>Class/position/relative_position()</td><td>Class/structure/local_methods()</td><td>sum (<depth_from_root (c)>, <parents (c)>, <methods_inherited (c)>, |C_sub (c)|, <local_methods(c)>)</td></tr>
<tr><td>Class/position/relative_position()</td><td>Class/position/depth_from_root()</td><td>sum (<depth_from_root (c)>, <parents (c)>, <methods_inherited (c)>, |C_sub (c)|, <local_methods(c)>)</td></tr>
<tr><td>Class/position/fraction_methods_overridden()</td><td>Class/position/depth_from_root()</td><td>fraction((<methods_overridden (c)> * <depth_from_root (c)>), (<methods(c)>))</td></tr>
<tr><td>Class/reuse/interface_attribute_method_meaning_and_access()</td><td>Class/position/depth_from_root()</td><td>reusability (c) * ((sum (reusability (m_i) s.t. m_i in M(c)))/<methods(c)>) + ((sum (reusability (a_j) s.t. a_j in A(c)))/<attributes(c)>), s.t. reusability (c) = (description_meaningfulness (c) + name - meaningfulness (c))/(|C_coupledTo (c)| + <depth_from_root(c)> + 1), reusability (m_i) = (name_meaningfulness (m_i) + P_usedBy (m_i) + ((sum (reusability (p_k)) s.t. p_k in P(m_i))/|p(m_i)|))/(|functions_performedBy (m_i) + callsTo - foreignClasses (m_i)|), reusability (p_k) = (name_meaningfulness (p_k))/(|subTypesOf (typesOf(p_k)| + 1) or = 5 if|p(m_i)| = 0, reusability (a_j) = (name_meaningfulness (a_j))/(|subTypesOf (a_J)| + 1) s.t. name_meaningfulness() and description_meaningfulness() are decided on a scale 1..5 by the developer.</td></tr>
<tr><td>System/structure/fraction_method_inherit()</td><td>Class/position/methods_inherited()</td><td>fraction (sum (<methods_inherited (c_i)>), sum (<methods(c_j)>)), s.t. c_i, c_j in C(s)</td></tr>
<tr><td>Class/position/relative_position()</td><td>Class/position/methods_inherited()</td><td>sum (<depth_from_root (c)>, <parents (c)>, <methods_inherited (c)>, |C_sub (c)|, <local_methods(c)>)</td></tr>
<tr><td>Class/position/fraction_overloaded_methods()</td><td>Class/position/methods_inherited()</td><td>fraction (<methods_Overloded_from_, <methods_ inherited (c)>)</td></tr>
<tr><td>Class/position/fraction_methods_inherited_function()</td><td>Class/position/methods_inherited()</td><td>fraction ((|M_virtualAdded(c)| + |M_virtualInherited(c)| + 2 * (<methods_added(c)> + |<methods_inherited(c)>), 2 * <methods(c)>)</td></tr>
<tr><td>Class/position/methods_specialized_function()</td><td>Class/position/methods_inherited()</td><td>avg (P + O + N) s.t. P = fraction (|M_inheritedRedefinedWithNewBehavior(c)|, |M_inheritedRedefined(c)|) O = fraction (|M_inheritedRedefined(c)|, <methods_inherited (c)>) N = fraction(level(c) * <methods_added(c)>, (<methods_added(c)> + 1) * (level(c) + 1))</td></tr>
<tr><td>Class/position/relative_position()</td><td>Class/position/parents()</td><td>sum (<depth_from_root (c)>, <parents (c)>, <methods_inherited (c)>, |C_sub (c)|, <local_methods(c)>)</td></tr>
<tr><td>System/structure/fraction_overridden_methods_new_methods()</td><td>Class/position/descendants()</td><td>fraction (sum (<methods_overridden (c_i)>), sum (|M_new (c_j)|) * sum (<descendants (c_j)>)) s.t. c_i, c_j in C(s)</td></tr>
<tr><td>Subsystem/structure/weighted_sum_ternal_class_complexity()</td><td>Class/interaction/sum_external_interaction()</td><td>sum (weight_i * <sum_external_interaction(c_i)>) s.t. c_i in C(subsystem), weight_i = LOC_estimated(c_i)/sum(LOC_estimate(c_j)), c_j in C(subsystem)</td></tr>
<tr><td>System/structure/fraction_overridden_methods_new_methods()</td><td>Class/position/methods_overridden()</td><td>fraction (sum (<methods_overridden (c_i)>), sum (|M_new (c_j)|) * sum (<descendants (c_j)>)) s.t. c_i, c_j in C(s)</td></tr>
</table>

<table>
<tr><td>Class/position/fraction_methods_overridden()</td><td>Class/position/methods_ overridden()</td><td>fraction((<methods_overridden (c)> * <depth_from_root (c)>), (<methods(c)>))</td></tr>
<tr><td>Class/position/fraction_methods_inherited_ function()</td><td>Class/position/methods_added()</td><td>fraction ((|M_virtualAdded(c)| + |M_virtualInherited(c)| + 2 * (<methods_added(c)> + <methods_inherited(c)>), 2 * <methods(c)>)</td></tr>
<tr><td>Class/position/methods_specialized_function()</td><td>Class/position/methods_added()</td><td>avg (P + O + N) s.t. P = fraction (|M_inheritedRedefinedWithNew Behavior(c)|, |M_inheritedRedefined(c)|) O = fraction (|M_inherited Redefined(c)|, <methods_inherited (c)>) N = fraction(level(c) * <methods_added(c)>, (<methods_added(c)> + 1) * (level(c) + 1))</td></tr>
<tr><td>Hierarchy/arrangement/aggregate_class_ abstraction_function()</td><td>Class/position/fraction_ methods_inherited_function()</td><td>areaUnder (curveDefinedBy (<fraction_methods_inherited_ function(c_i)>, level(c_i)) s.t. c_i in C(hierarchy)</td></tr>
<tr><td>System/structure/relationship_abstraction_ function()</td><td>Class/position/fraction_ methods_inherited_function()</td><td>fraction (sum (<fraction_methods_inherited_function(c_i)>), |Associations_aggregations(c)|) s.t Associations_aggregations(c) = unique (Associations_involving(c) union Aggregations_involving(c)), (c_i, c) in Associations_aggregations©</td></tr>
<tr><td>Class/comments/comments_per_method()</td><td>Class/comments/comment_ lines()</td><td>avg (<comment_lines (M(c))>) s.t.m_i in M(c)</td></tr>
<tr><td>Method/position/fraction_method_inherited_ not_overloaded()</td><td>Method/position/inheritance_ occurrences()</td><td>fraction (difference(<inheritance_occurrences()>, <overload_ occurrences()>), <inheritance_occurrences()>)</td></tr>
<tr><td>Method/position/fraction_method_inherited_ not_overloaded()</td><td>Method/position/overload_ occurrences()</td><td>fraction (difference(<inheritance_occurrences()>, <overload_ occurrences()>), <inheritance_occurrences()>)</td></tr>
<tr><td>Method/structure/weighted_sum_mccabe_ halstead()</td><td>Method/structure/method_ mccabe_complexity()</td><td>(3.42 * log_e (Effort_halstead (m)) + 0.23 * <method_mccabe_ complexity(m)> + 16.2 * log_e (<lines_of_code(m)>)) [See Halstead[ ] for Effort_halstead, see cyclomatic complexity]</td></tr>
<tr><td>Hierarchy/arrangement/hierarchy_length_ params_interaction()</td><td>Method/structure/method_ length_params_interaction()</td><td>sum <method_length_params_interaction(m_i)> s.t. m_i in M(C(hierarchy))</td></tr>
<tr><td>System/structure/system_length_params_ interaction()</td><td>Method/structure/method_ length_params_interaction()</td><td>method_complexity (main_function(s)) + sum <method_length_ params_interaction(m_i)> s.t. m_i in M(C(s))</td></tr>
<tr><td>Method/structure/method_length_params_ interaction()</td><td>Method/interaction/arguments_ received()</td><td>length(m) * square (coupling(m)) s.t. length(m) = |T_if(m) | + |T_case(m)| + <attributes(m)>; coupling(m) = sum (ip(m), op(m)) s.t. ip(m) = (1 + <arguments_received(m)> + void(m)) s.t. void(m) = 1 if m returns value and 0, otherwise; op(m) = (sum (|ReferencesTo_m_i (m)| * (1 + |P_referencedBy_m(m_i)| + void (m_i)) + sum (|ReferencesTo_a_j(m)|)) s.t. m_i in M_referencedBy (m), a_j in A_referencedBy (m)</td></tr>
<tr><td>Class/structure/class_length_interaction()</td><td>Method/interaction/arguments_ received()</td><td>length(c) * square (coupling(c)) s.t. length(c) = sum (length(m_i)) s.t. m_i in M(c); coupling(c) = sum (ip(m_j) + (sum (r_j_k) + sum (r_j_l))) s.t. m_j in M(c), K = 1..NMR(c), l = NMR(c) + 1..NMR(c) + NVR(c), NMR(c) = |unique (M_referencedBy(M(c)))|, NVR(c) = |unique (A_referencedBy (M(c))|, r_j_k = |ReferencesTo_m_k (m_j)| * (1 + |R_ referencedBy_m_j(m_k) + void(m_k), r_j_l + |ReferencesTo_a_l(m_j) |s.t. length(m) = |T_ys(m)| + | T_case(m)| + <attributes(m)>|; coupling(m) = sum (ip(m), op(m)) s.t. ip(m) = (1 + <arguments_ received(m)> + void(m)) s.t. void(m) = 1 if m returns value and 0,</td></tr>
</table>

Continue

Table IX. *Continued*

Metric 1	Metric 2	Computation of Metric 1 using <Metric 2>
		otherwise; op(m) = (sum (|ReferencesTo_m_i (m)| * (1 + |P_referencedBy_m(m_i)| + void (m_i)) + sum (|ReferencesTo_a_j(m)|)) s.t. m_i in M_referencedBy (m), a_j in A_referencedBy (m)
Repository/reuse/params_of_method_of_parents_classes_in_repository()	Method/interaction/arguments_received()	<arguments_received(m)> s.t. m in M(C_parent_in_r (C(s)))
Class/interaction/class_inherit_noninherit_interaction()	Method/interaction/sum_method_nonlocalrefs()	sum (<sum_method_nonlocalrefs (m_i)>) s.t. m_i in M(c)
Method/structure/weighted_sum_mccabe_halstead()	Method/size/lines_of_code()	(3.42 * log_e (Effort_halstead (m)) + 0.23 * <method_mccabe_complexity(m)> + 16.2 * log_e (<lines_of_code(m)>)) [See Halstead[ ] for Effort_halstead, see cyclomatic complexity]
Class/structure/weighted_sum_attribute_association_methods_rule_for_class()	Method/size/lines_of_code()	sum (weight1*(<attributes(c)> + |Associations(c)|), weight2 * avg(<lines_of_code(M(c))>)/STDLOC * <methods(c)>, weight3 * |RulesPerRuleSet(c)| * avg(|AntecedentClausesPerRule(c)|)* |RuleSets(c)|) where STDLOC means agreed language-dependent standard LOC per method, other terms not fully defined, weights decided by designer.
Class/reuse/fraction_functional()	Method/size/statements()	fraction(|T_functionOriented(c)|, <statements(c)>)
System/reuse/fraction_source_reuse()	Method/size/statements()	fraction (|T_imported (s, r)|, <statements(s)>)
Method/size/fraction_instance_attributes_used_by_method()	Method/size/instance_attributes_used	fraction(<instance_attributes_used(m)>, |A(c(m)|)
Class/behavior/attrib_and_method_polymorphism()	Method/position/classes_defined_in	normalized_sum (<classes_defined_in (a)>, <classes_defined_in(m)>) - - "normalized_sum()" not defined
System/structure/syntax_inh_interact_complexity()	System/structure/sum_inheritance_complexity()	sum(<sum_syntax_halstead_complexity(s)>, <sum_inheritance_complexity(s)>, <sum_interaction_complexity(s)>)
System/structure/syntax_inh_interact_complexity()	System/structure/sum_syntax_halstead_complexity()	sum(<sum_syntax_halstead_complexity(s)>, <sum_inheritance_complexity(s)>, <sum_interaction_complexity(s)>)
Package/abstractness/fraction_abstract_classes()	System/structure/abstract_classes()	fraction (<abstract_classes(package)>, |C(package)|) where no computable formulation of C_abstract() is provided
System/reuse/inverse_reuse_versus_new_cost()	System/reuse/reuse_versus_new_cost()	1/relative cost <reuse_versus_new_cost(c)>
System/size/fraction_non_root_classes()	System/reuse/root_classes()	1 - fraction (<root_classes(s)>, <classes(s)>)
Class/position/fraction_overloaded_methods()	System/reuse/methods_overloaded_from_repository()	fraction (<methods_Overloaded_from_repository (c)>, <methods_inherited (c)>)
System/requirements/use_cases_actors_interactions_others()	System/size/use_cases()	k1 (|square(<use_cases(s)>) + <use_case_actor_interactions_without_extends_uses()> + k2 (<use_case_actor_interactions()>-<use_case_actor_interactions_without_extends_uses()>), k1 and k2 are empirically determined
System/structure/subsystens_and_links()	System/size/subsystems()	(k1 * <subsystems(s)>, k2 *|Arc_acrossSubsystems(s)|) s.t. k1, k2 constants
System/reuse/fraction_classes_leveraged()	System/size/classes()	fraction (|C_leveraged (s)|, <classes(s)>)
System/reuse/fraction_superclasses()	System/size/classes()	fraction (|C_super (s)|, <classes(s)>)

System/reuse/fraction_classes_importedverbatim()	System/size/classes()	fraction (|C_importedVerbartim (s)|, <classes(s)>)
Class/behavior/attrib_and_method_polymorphism()	System/size/classes()	normalized_sum (<classes_defined_in (a)>, <classes_defined_in(m)>) - - "normalized_sum()" not defined
System/effort/classes_per_developer()	System/size/classes()	<classes(s)>/|Developers_involvedIn(s)|
System/reuse/fraction_classes_reused_as_is()	System/size/classes()	fraction (|C_reusedAsIs(s)|, <classes(s)>)
System/reuse/fraction_classes_reused_modified()	System/size/classes()	(sum (|C_reusedAsIs(s)|, |C_modifiedDuringReuse(s)|))/(<classes(s)>)
System/size/fraction_non_root_classes()	System/size/classes()	1 - fraction (<root_classes(s)>, <classes(s)>)
System/structure/syntax_inh_interact_complexity()	System/dynamics/sum_interaction_complexity()	sum(<sum_syntax_halstead_complexity(s)>, <sum_inheritance_complexity(s)>, <sum_interaction_complexity(s)>)
System/requirements/use_case_actor_interactions_without_extends_uses()	System/requirements/use_case_actor_interactions()	<use_case_actor_interactions()> -|Interactions (Actors(s), UseCasesRelatedTo (UseCasesDirectlyInteractingWith(Actors(s))))|
System/requirements/use_cases_actors_interactions_others()	System/requirements/use_case_actor_interactions()	k1 (|square(<use_cases(s)>) + <use_case_actor_interactions_without_extends_uses()> + k2 (<use_case_actor_interactions()>-<use_case_actor_interactions_without_extends_uses()>), k1 and k2 are empirically determined
System/requirements/use_cases_actors_interactions_others()	System/requirements/use_case_actor_interactions_without_extends_uses()	k1 (|square(<use_cases(s)>) + <use_case_actor_interactions_without_extends_uses()> + k2 (<use_case_actor_interactions()>-<use_case_actor_interactions_without_extends_uses()>), k1 and k2 are empirically determined

Note: s.t. = such that.

Table X. Equivalent Formulations of Metrics from Different Researchers

Entity	Attribute	Metric	Proposed by*
Class	interaction	sum_method_import_interactions_with_friends()	2
	interface	public_local_methods()	2
	position	ancestors()	4
	position	children()	3
	position	depth_from_root()	4
	position	descendants()	5
	position	max_inherit_depth()	2
	position	sum_method_import_interactions_with_ancestors()	2
	reuse	fraction_functional()	2
	size	attributes()	4
	structure	method_attribute_usage_in_class()	3
	structure	public_attributes()	2
	structure	sum_method_complexity()	2
Method	interaction	methods_sending_messages_to()	2
	position	classes_inheriting()	2
	size	executable_statements()	2
	size	instance_attributes_used	2
	size	statements()	2
System	reuse	fraction_classes_importedverbatim()	2
	reuse	fraction_classes_leveraged()	2
	size	atttributesin_system()	2
	size	classes()	5
	size	messages()	2
	size	methods_in_system()	3
	size	subsystems()	4
	size	use_cases()	2
	structure	abstract_classes()	2
	structure	avg_instance_attributes()	2
	structure	avg_messages_sent()	2
	structure	avg_method_size()	2
	structure	avg_methods()	2
	structure	inheritance_associations()	2
	structure	multiple_inheritance()	2
	structure	sum_association_arity()	2

*Number of researchers.

4.2.4. Metrics That Use Functions Proposed for Traditional Metrics. A few metrics make use of functions that were suggested for traditional metrics. The most telling example in this category is the metrics that use the McCabe [1976] cyclomatic complexity function. Table XI shows these metrics. For example, the cyclomatic complexity function is used by several researchers to compute complexity. This demonstrates the researchers' biases in applying traditional functions to object-oriented systems. Prima facie, such applications suggest a possible misuse of the traditional functions, following the arguments presented in Section 2.1. We see only a few such uses, with cyclomatic complexity being the most glaring. Others include Boehm's complexity number, which is used for some metrics by different researchers. Such analysis can force researchers to rigourously evaluate the appropriateness of traditional metrics for object-oriented systems.

Additional analyses of metrics are certainly possible beyond the three possibilities discussed above. Consider, for example, using the dimensions of artifact granularity (method $\subset$ class, class $\subset$ subsystem, and so on), and mathematical aggregation (set theoretic operations on elements of a set, e.g., class $\in$ set of classes). Talbi et al. [2001] suggest a taxonomy that can be useful in this regard. Other analyses may include investigating the use of statistical summary operators (such as average) or dispersion oprators (such as min, max, or variance) across metrics that apply to elements versus sets. We leave these, along with several others, as further extensions to this research.

Table XI. Metrics Built on Traditional Metrics

Entity	Attribute	Metric	Researcher	Year
Class	size	sum_attribute_boehm_size()	Sharble and Cohen	1993
Class	structure	avg_method_mccabe_complexity()	Etzkorn et al.	1999
Class	structure	class_mccabe_complexity()	Karlsson	1995
Class	structure	sum_distance_times_method_and_attributes_halstead_complexity()	Etzkorn et al.	1999
Method	size	halstead_length()	De Champeaux	1997
Method	size	halstead_volume()	Abreu and Carapuca	1994
Method	structure	method_mccabe_complexity()	Abreu and Carapuca	1994
Method	structure	weighted_sum_mccabe_halstead()	De Champeaux	1997
Subsystem	structure	class_group_mccabe_complexity()	Karlsson	1995
System	structure	assoc_mccabe_complexity()	Kolewe	1993
System	structure	sum_syntax_halstead_complexity()	Kim et al.	1994

Table XII. Number of Metrics Proposed over Time for Entities

Entity	1989	1991	1992	1993	1994	1995	1996	1997	1998	1999
Association								1		
Attribute						4				
Class	2		7	17	49	28	9	45	19	4
Event			1							
Hierarchy				1					1	
Link										
Message										
Method			4	3	16	10		7		
Package					3				2	
Parameter										
Scenario								3		
Subsystem						2				
System	3	5	22	4	43	34	1	24	14	
Use Case			2					1		

Note: Only entities for which metrics have been proposed are shown.

4.3. Trends

Based on the compilation of metrics, we further present a historical perspective on available object-oriented metrics that can provide a sense of the direction in which the field appears to be moving. Tables XII and XIII show the number of metrics proposed over time, mapped against the entities and attributes. The tables suggest that intense level of activity witnessed during the mid-1990s (1994 to 1997) appears to be tapering off as more researchers are shifting their agendas away from measurement concerns.

This is a disturbing finding because a number of concerns still remain, as seen from our analysis in Sections 4.1 and 4.2, where we have identified a number of gaps in the present state of the research. Our discipline has been criticized for changing focus too quickly to meet the vagaries of the market. As key issues dealing with measurement of object-oriented development processes remain unresolved, we see a similar trend. We hope that the present analysis will provide researchers specific avenues that they may consider for advancing the field.

5. CONCLUDING REMARKS

We have presented a survey of existing metrics proposed for object-oriented systems focusing on product metrics. In particular, we have analyzed the metrics for the coverage of entities, attributes, and development stages (indicated as states). The classification of metrics is also based on the dimensions of elementarity and directness. Our analysis has included the investigation of computation across metrics such as aggregation (Table VIII) or usage across metrics (Table IX), equivalent formulations of metrics by multiple researchers (Table X), and exploitation of traditional metrics for object-oriented metrics (Table XI). We have also analyzed activity in this area over time,

Table XIII. Number of Metrics Proposed over Time for Entity-Attributes

Entity	Attribute	1989	1991	1992	1993	1994	1995	1996	1997	1998	1999
Association	size								1		
Attribute	interaction						2				
	position						2				
Class	behavior	1				1	1		5		
	comments					2			1		
	interaction	1		5	6	3	10		17	3	
	interface					3	1			2	
	performance					5					
	position				1	13	8		8	6	
	reuse					6				1	
	size				3	7			3		
	structure			2	7	9	8	9	11	7	4
Event	size			1							
Hierarchy	arrangement				1					1	
Link	arity										
Message	size										
Method	abstractness						1				
	interaction			2	1	3	6		2		
	interface				1						
	performance					2					
	position			1		3	2				
	size			1		6	1		4		
	structure				1	2			1		
Package	abstractness					1					
	interaction					2				1	
	structure									1	
Parameter	size										
Scenario	size								3		
Subsystem	structure						2				
System	change					1					
	dynamics					2	1		8		
	effort					9					
	performance					1					
	requirements			2						3	
	reuse	2	5	3	1	2	12		3		
	size	1		9		9	4		5	3	
	structure			8	3	18	17	1	8	8	
	indeterminate					1					
Use Case	interface			1							
	size			1					1		

and commented on the trends (Tables XII and XIII). The survey provides a historical view of existing research, uncovers gaps in existing research, and suggests opportunities for future research.

The presented analysis is based on a uniform representation of existing metrics and is aided by storing this information in a structured repository for ease of processing; all the analysis tables in Section 4 are generated from this repository. The complete data with the uniform representation is available online in the ACM Digital Library, in a format that is easily amenable to conversion to a standardized representation using the eXtensible Markup Language. This uniformity in representation (Table IV) is achieved by applying a mathematical formalism (Table III), which in turn is driven by a framework for object-oriented metrics (Figure 1), the relational model for measurement (Figure 2), and the dimensions for classifying metrics (Figure 3). Further support for the framework is provided by use of the object-oriented system development process as the underlying phenomenon, which suggests the distinction between the entities products, processes, and resources, and, specifically, the states of products (Figure 4). Based on these concepts, a bottom-up analysis of the metrics proposed for object-oriented systems has resulted in the entities

Table XIV. Application of Metrics

Possible Steps for Identifying Apprppriate Metrics			
Step	Selection	Example	Source
1	Entity	Class	Table I
2	Goal/Question	Maintainability	Decision-Maker
3	State(s)	Specifications	Decision-Maker, Choice from Figure 4
4	Attribute(s)	Position, Structure	Decision-Maker, Choice from Table II
5	Metric Type	Elementary and Direct	Decision-Maker, Choice from Figure 3, Table VII
6	Metrics	Results below	Metrics Repository

Application of Metrics

Entity	Attribute	Metric	State	Elementary	Direct
Class	position	methods_added()	2 or 3	y	y
Class	position	children()	2 or 3	y	y
Class	position	methods_inherited()	2 or 3	y	y
Class	position	parents()	2 or 3	y	y
Class	position	methods_overridden()	2 or 3	y	y
Class	position	max_inherit_depth()	2 or 3	y	y
Class	structure	avg_method_params()	2 or 3	y	y
Class	structure	sum_attrib_chenlu_value()	2 or 3	y	y
Class	structure	attrib_combinations_ constraints()	2 or 3	y	y
Class	structure	local_methods()	2 or 3	y	y
Class	structure	classes_thisclass_depends_on()	2 or 3	y	y
Class	structure	methods_longer_than_N_loc()	2 or 3	y	y
Class	structure	public_attributes()	2 or 3	y	y
Class	structure	public_methods()	2 or 3	y	y
Class	structure	method_params()	2 or 3	y	y
Class	structure	bidirectional_class_usage()	2 or 3	y	y
Class	structure	classes_used()	2 or 3	y	y
Class	structure	abstract_classes_used()	2 or 3	y	y
Class	structure	classes_dependent_on()	2 or 3	y	y

*Legend: 1 = Requirements, 2 = Design, 3 = Implementation, 4 = Operational; y = yes.

and attributes presented in Tables I and II.

Finally, we suggest one plausible use of a metrics repository compiled using this formalism: use of the metrics for specific goals or for answering questions. Determining specific instantiaition of "goals" or "questions" is a rather daunting task. A goal/question relates to either quantifiably assessing some attribute of an existing product, process, or resource, or quantifiably predicting some attribute of a future product, process, or resource. Mylopoulos et al. [1992] also suggest alternatives for understanding and interpreting nonfunctional requirements, which can be adapted for this purpose. Along with these works, the GQM paradigm suggests that available entity-attribute pairs or entity-attribute-metric tuples may be used to answer questions arising from predetermined goals.

In this regard, researchers have suggested approaches to relate metrics to quality characteristics (see, for example, Mylopoulos et al. [1992]; Shepperd [1992]; Gillibrand and Liu [1998]). It is difficult, however, to reach a genuine consensus about the definitions of exernal attributes that are sometimes referred to as *ilities* [Fenton 1994]. ISO 9126 [ISO 9126; ISO/IEC FCD 9126-2nd], for example, defines Quality as consisting of: Functionality, Reliability, Usability, Efficiency, Maintainability, and Portability. McCall et al.'s [1977] hierarchy of ilities indicates hierarchies of Maintainability, which consists of Simplicity, Modularity, and of Conciseness; and of Dependability, which consists of Reliability, Safety, Security, Availability, Maintainability, Usability, and Extendability. These terms are notoriously difficult to operationalize with universal applicability.

We acknowledge these problems, which have been reported elsewhere. However, in spite of these problems, it may be possible to derive broad guidelines for using the metrics. Table XIV shows one possible approach to using our representation to select appropriate metrics in the context provided by the development lifecycles. It shows a possible question ("assessment of maintainability") for the goal ("improving maintainability"), and the progression from a goal/question to an attribute, selection of state(s), metric type(s), and the resulting metrics extracted from the metrics repository.

Clearly, the above analysis does not extend to expressing judgments about individual metrics, such as their trustworthiness or relevance. These concerns are the topic of research on specific metrics and their mapping against specific concerns. Analysis like that presented in Table XIV merely serves as the first step in identifying possibilities, which may be informed by other research that takes into account issues such as trustworthiness. Our contribution has been mainly in providing an insightful survey of object-oriented product metrics using the internal perspective (Figure 1) and a mathematical formalism.

ACKNOWLEDGMENTS

We acknowledge the help of the reviewers and the associate editor, whose comments helped shape the paper through the revisions.

REFERENCES

ABREU, B. F. AND CARAPUCA, R. 1994. Candidate metrics for object-oriented software within a taxonomy framework. *J. Syst. Softw. 26*, 87–96.

ABREU, B. F., GOULAO, M., AND ESTEVES, R. 1995. Toward the design quality evaluation of object-oriented software systems. In *Proceedings of the Fifth International Conference on Software Quality*. 44–57.

BAKER, A. L., BIEMAN, J. M., FENTON, N. E., GUSTAFSON, D., MELTON, A., AND WHITTY, R. W. 1990. A philosophy for software measurement. *J. Syst. Softw. 12*, 277–281.

BANDI, R. K. 1998. Using object-oriented design complexity metrics to predict maintenance performance. Ph.D. dissertation, Georgia State University, Atlanta, GA.

BARNES, G. AND SWIM, B. 1993. Inheriting software metrics. *J. Obj.-Orient. Program. 6* (November–December), 27–34.

BASILI, V. R. 1992. Software modeling and measurement: The Goal/Question/Metric paradigm. Tech. Rep. CS-TR-2956. Department of Computer Science, University of Maryland, College Park, MD.

BASILI, V. R., BRIAND, L. C., AND MELO, W. L. 1996. A validation of object-oriented design metrics as quality indicators. *IEEE Trans. Softw. Eng. 22*, 751–761.

BAUDOIN, C. AND HOLLOWELL, G. 1996. *Realizing the Object-Oriented Lifecycle*. Prentice Hall, Englewood Cliffs, NJ.

BERARD, E. V. 1993. *Essays on Object-Oriented Software Engineering*. Prentice Hall, Englewood Cliffs, NJ.

BIEMAN, J. M. AND KARINATHINI, S. 1995. Measurement of language-supported reuse in object-oriented and object-based software. *J. Syst. Softw. 30*, 271–293.

BLUM, B. A. 1994. Taxonomy of software development methods. *Commun. ACM 37*, 82–94.

BOOCH, G. 1994. *Object-Oriented Analysis and Design with Applications*. Benjamin/Cummings Pub. Co., Redwood City, CA.

BRIAND, L. C., DALY, J. W., AND WUST, J. K. 1999. A unified framework for coupling measurement in object-oriented systems. *IEEE Trans. Softw. Eng. 25*, 91–121.

BROOKS, F. 1987. No silver bullet: Essence and accidents of software engineering. *IEEE Comput. 20*, 4, 10–19.

BUSH, M. E. AND FENTON, N. E. 1990. Software measurement: A conceptual framework. *J. Syst. Softw. 20*, 223–231.

CHEN, J. Y. AND LU, J. F. 1993. A new metric for object-oriented design. *Inform. Softw. Tech. 35* (April), 232–240.

CHIDAMBER, S. R. AND KEMERER, C. F. 1991. Towards a metric suite for object oriented design. *SIGPLAN Not. 26*, 197–211.

CHIDAMBER, S. R. AND KEMERER, D. F. 1994. A metrics suite for object-oriented design. *IEEE Trans. Softw. Eng. 20*, 476–493.

DE CHAMPEAUX, D. 1997. *Object-Oriented Development Process and Metrics*. Prentice Hall, Englewood Cliffs, NJ.

ETZKORN, L., BANIYA, J., AND DAVIS, C. 1999. Design and complexity metrics for OO classes. *J. OOP 12*, 1, 35–40.

FENTON, N. E. 1991. *Software Metrics, A Rigorous Approach*. Chapman & Hall, New York, NY.

FENTON, N. E. 1994. Software measurement: A necessary scientific basis. *IEEE Trans. Softw. Eng. 20*, 199–206.

FENTON, N. E. AND PFLEEGER, S. L. 1997. *Software Metrics, A Rigorous and Practical Approach*, 2nd ed. International Thomson Computer Press, Boston, MA.

FINKELSTEIN, L. 1984. A Review of the Fundamental Concepts of Measurement. *Measurement*. *2*, 1, 25–34.

GILLIBRAND, D. AND LIU, K. 1998. Quality metrics for OO design. *J. Obj.-Orient. Program. 17*, 175–184.

HENDERSON-SELLERS, B. 1996. The mathematical validity of software metrics. *Softw. Eng. Notes 21*, 89–94.

INCE, D. 1990. Software metrics: Introduction. *Inform. Softw. Tech. 32*, 297–303.

ISO 9126. 1991. Information technology—Software product evaluation—quality characteristics and guidelines for their use, ISO, Geneva, Switzerland.

ISO/IEC FCD 9126-2ND. 1998. Software quality characteristics and metrics—part 1: Quality characteristics and sub-characteristics. ISO, Geneva, Switzerland.

JACOBSON, I., CHRISTERSON, M., JONSSON, P., AND OVERGAARD, G. 1992. *Object-Oriented Software Engineering. A Use Case Driven Approach*. Addison-Wesley, Reading, MA.

KARLSSON, E. 1995. *Software Reuse: A Holistic Approach*. John Wiley & Sons, New York, NY.

KIM, E. M., CHANG, O. B., KUSUMOTOS, S., AND KIKUNO, T. 1994. Analysis of metrics for object-oriented program complexity. In *Proceedings of the Eighteenth Annual International Computer Software and Applications Conference* (COMPSAC '94). IEEE Computer Society, Press, Los Alamitos, CA, 201–207.

LI, W. 1998. Another metric suite for object-oriented programming. *J. Syst. Softw. 44*, 155–162.

LI, W. AND HENRY, S. 1993. Object oriented metrics that predict maintainability. *J. Syst. Softw. 23* (Nov.), 111–122.

LORENZ, M. AND KIDD, J. 1994. *Object-Oriented Software Metrics*. Prentice Hall, Englewood Cliffs, NJ.

MARCHESI, M. 1998. OOA metrics for the Unified Modeling Language. In *Proceedings of the Second Euromicro Conference on Software Maintenance and Reengineering*. IEEE Computer Society Press, Los Alamitos, CA.

MARCINIAK, J. (ED.) 1994. *Encyclopedia of Software Engineering*. John Wiley & Sons, New York, NY.

MC CABE, T. J. 1976. A complexity measure. *IEEE Trans. Softw. Eng. SE-2*, 4, 308–320.

MCCALL, J. A., RICHARDS, P. K., AND WALTERS, G. F. 1977. Factors in software quality, RADC TR-77-369, I, II, III. U.S. Rome Air Development Center Reports NTIS AD/A-049 014, 015, 055. Rome, NY.

MCGREGOR, J. AND KORSON, T. 1994. Integrated object-oriented testing and development process. *Commun. ACM 37*, 59–77.

MILI, H., MILI, F., AND MILI, A. 1995. Reusing software: Issues and research directions. *IEEE Trans. Softw. Eng. 21*, 528–562.

MYLOPOULOS, J., CHUNG, L., AND NIXON, B. 1992. Representing and using nonfunctional requirements: A process-oriented approach. *IEEE Trans. Softw. Eng. 18*, 6 (June), 483–497.

NAKANISHI, K. T. ARANO, K. T., AND IMASE, M. 1995. A matric for evaluating class library interfaces and its application to library upgrades. In *Proceedings of the International Conference on Software Maintenance* (ICSM '95, Nice, France). IEEE Computer Society Press, Los Alamitos, CA.

RATIONAL 2001. The Rational Unified Process. Available online at `http://www.rational.com/products/rup/index.jsp`.

ROBERTS, F. S. 1979. *Measurement Theory with Applications to Decision Making, Utility and the Social Sciences*. Addison-Wesley, Reading, MA.

ROMBACH, H. D. 1990. Design measurement: Some lessons learned. *IEEE Softw. 7* (March), 17–25.

SHARBLE, R. C. AND COHEN, S. S. 1993. The object-oriented brewery: A comparison of two object-oriented development methods. *ACM SIGSOFT Softw. Eng. Notes 18*, 2, 60–73.

SHEPPERD, M. 1992. Products, processes and metrics. *Inform. Softw. Tech. 34*, 674–680.

TALBI, T., MEYER, B., AND STAPF, E. 2001. A metric framework for object-oriented development. In *Proceedings of the 39th IEEE International Conference and Exhibition on Technology of Object-Oriented Languages and Systems* (TOOLS). 164–172.

TEGARDEN, D. P., SHEETZ, S. D., AND MONARCHI, D. E. 1995. A software complexity model of object-oriented systems. *Decis. Supp. Syst*. *13*, 241–262.

WAND, Y. AND WEBER, R. 1995. On the deep structure of information systems. *Inform. Syst. J. 5*, 203–223.

ZUSE, H. 1991. *Software Complexity—Measures and Methods*. De Gruyter, Berlin, Germany.

ZUSE, H. 2001. Zuse/Drabe Measurement Information System. Available online at `http://home.t-online.de/home/horst.zuse/zd-www.html`.

ZUSE, H. AND BOLLMAN, P. 1989. *Software Metrics: Using Measurement Theory to Describe the Properties and Scales of Static Software Complexity Metrics*. Technische Universitat Berlin, Berlin, Germany.

Received July 2001; revised October 2002, March 2003; accepted April 2003