A Multi-Layered System of Metrics for the Measurement of Reuse by Inheritance

Radu Marinescu
"Politehnica" University of Timişoara
marinesc@fzi.de

Abstract

In spite of the intense efforts of metrics researches, the impact of object-oriented software metrics is for the moment still quite reduced. The cause of this fact lies not in an intrinsic incapacity of metrics to help assessing and improving the quality of object-oriented systems, but in an unsystematic, dispersed and ambiguous manner of defining and using the metrics. In this paper we define a multi-layered system of metrics that measures the inheritance-based reuse, and propose a number of metric definitions for the layers of this system. By defining and using such systems of metrics, we obtain a unitary approach of related measures and a systematic, yet flexible manner of defining new measures as part of a particular metrics system. Organising metric definitions in systems of metrics will contribute to that strongly needed order among object-oriented metrics, increasing thus their reliability and usability.
Keywords: *Measurement, Metric Definitions, Inheritance, Reuse, Reengineering.*

1 Introduction

The need of analysing and assessing object-oriented software has dramatically increased in the last years. This has lead to a large amount of research effort being invested in the area of object-oriented software measurement, both for predicting and evaluating the quality of software [2]. In spite of the great need for proper measurements and of the intense research efforts on object-oriented metrics, the large majority of the proposed measurements were not assimilated in the industry [4].

We think there are three main causes for this. First, there is a constant lack of following a rigourously specified process for defining software measures [4]. Both Fenton [9] and Whitemire [17] emphasised this aspect proposing such rigourous processes for defining metrics, yet these processes are still not followed. The use of a standard terminology or formalism would eliminate the ambiguities found in most metrics definitions and would increase therefore their usefulness and trustworthiness [2] [10]. Second, metrics definitions are not only quite ambiguous, but also very dispersed. In other words, there is a lack of order among metric definitions. This order would permit to compare and group together similar metrics offering in the same time a frame in which further contributions could be easily added [2]. Finally, the use of software metrics is reduced because of lack of consistent experimental results for the proposed metrics. In the end, these results are the on which indicate if it is really useful to apply a particular measurement.

The goal of this paper is to define in a rigourous manner a system of object-oriented measures, proving that it represents an adequate frame for defining further measurements that evaluate the different aspects of reuse by inheritance, and showing how this system can be applied to the re-engineering process.

In order to reach this goal, we first define a multi-layered system of measures, in which the layers reflect the different levels of abstraction for which measures can be defined. The layers, together with the metrics proposed for each layer, are formally defined and informally described. The contribution of this paper is not only the definition of a particular system of metrics, but also the general methodology for further definitions of metric systems.

The paper is organised as follows. Section 2 introduces the concepts and the notation that will be used to define the system of metrics. In Section 3 we define the multi-layered system of metrics together with a number of concrete metric definitions for each layer. Section 4 presents the results of an experimental study that applies this metrics system for reengineering purposes. In Section 5 we place our study in the context of similar researches from the literature. The conclusions and the directions for future work come in Section 6.

2 Definitions and Notation

Definition 2.1 Set of Classes. *Structurally, an object-oriented system consists of a set of classes, C.* Thus, for every class c in the system, $c \in C$.

At the class-level we define following concepts:

Definition 2.2 Declared and Implemented Methods *[2]. For each class $c \in C$ let*

- *$M(c)$ be the set of the methods from class;*
- *$M_D(c)$ be the set of* **declared methods** *in c, i.e. the virtual methods of c and the methods that c inherits but does not override;*
- *$M_I(c)$ be the set of* **implemented methods** *in c, i.e. the inherited methods that c overrides and the non-virtual, non-inherited methods of c.*

Thus, $M(c) = M_D(c) \bigcup M_I(c)$ and $M_D(c) \bigcap M_I(c) = \emptyset$.

Definition 2.3 Set of Methods. *The set of all implemented methods of a set of classes C is named set of methods, notated M(C).* Formally,

$$M(C) = \bigcup_{c \in C} M_I(c)$$

Definition 2.4 Defined Class Member. *A defined class member (dcm_c) of a class $c \in C$, is a non-inherited attribute or a implemented method of that class. The set of defined class members is denoted by $DCM(c)$ ($dcm_c \in DCM(c)$).*

Definition 2.5 Accessible Class Member. *An accessible class member (acm_c) of a class $c \in C$, is a defined class member that is accessible to other classes. The set of accessible class members is denoted by $ACM(c)$ ($acm_c \in ACM(c)$). Between the set of defined class members and the set of accessible class member exists an inclusion relation: $\forall c \in C, ACM(c) \subset DCM(c)$.*

Definition 2.6 Set of Ancestors. *The set of classes from which a class c is directly or indirectly derived is called set of ancestors of class c, notated $A(c)$.*

3 The "Reuse of Ancestors" System of Metrics

3.1 Defining a Multi-Layered System of Metrics

Definition 3.1 Multi-Layered System of Metrics (MLSM). *A multi-layered system of metrics* (MLSM) *is a set of interdependent metric definitions organised in layers, in which the definition of each layer – excepting the lowest – is based upon the definitions of the lower layers in the system.*

The motivation behind the definition of multi-layered systems of metrics stays in the fact that most internal measurement attributes are too rich conceptually to be measured by using a single metric. This fact has caused in the past a lot of confusion.

This multi-layered approach is not new. In fact, it follows the same divide-and-conquer principle implemented as standard to the measurement of software quality [12]. The idea of a decompositional approach stays also behind the quality models of Boehm [1] and McCall [6]. From this point of view, MLSM should be understood as a new way of applying the decompositional approach.

In contrast to the quality models, MLSM does *not* refer to the decomposition of higher-level quality factors in lower-level criteria [6], but it refers to the decomposition of *metric definitions* based on the different levels where a software quality factor can be evaluated (e.g. reuse can be measured at the level of a method, but it can be also measured at the class level or system level). The measurement results at each level of measurement can and have to be interpreted differently, increasing thus the potential usefulness of the metric.

There is three-step method by which a MLSM deals with this problem:

1. **Identifying the Layers.** The different levels at which the attribute can be measured are identified and named.
2. **Defining the Layers.** The general form of each layer is defined. The identification and definition of the layers must capture the general characteristics of the attribute.
3. **Defining Layer-specific Metrics.** In order to use the system of metrics, for every layer a concrete metric definition has to be provided. The definition of these metrics is determined by the particular definition used for the attribute.

3.2 The Layers for "Reuse of Ancestors"

The name of this MLSM speaks already about the attribute that we want to measure: the reuse of ancestors in the derived classes. The reuse of ancestors can be evaluated at one of the three levels:

- *Method-Ancestor level.* In fact the real reuse of ancestors takes place at this level, therefore this is the base (bottom) level of the system.
- *Class-Ancestor level.* When speaking about reuse we usually think in terms of *classes* reusing other classes.
- *Class level.* Often we need to assess the total reuse for a class (e.g. in order to evaluate its self-sufficiency). For this we need a higher-level definition of reuse.

The Bottom Layer – Reuse of Ancestor in Methods (RAM)

Definition 3.2 The RAM Layer. *Any metric that expresses the measure of reuse of an ancestor class $a \in A(c)$ at the level of the methods $m_c \in M_I(c)$ from class c, belongs to the RAM layer.* Formally, a metric belongs to this layer if it is defined as follows:

$$RAM : D \subset M(C) \times C \rightarrow \Re$$

where $D = \{(m_c, a) \mid c \in C \ \wedge \ m_c \in M_I(c) \ \wedge \ a \in A(c)\}$

The Middle Layer – Class-Ancestor Reuse (CAR)

Definition 3.3 The CAR Layer. *Any measurement that quantifies the reuse relation between a class $c \in C$ and an ancestor class, $a \in A(c)$, belongs to the CAR layer of this metrics system.* Rigourously, a metric belongs to this layer if it conforms to following definition:

$$CAR : D \subset C \times C \rightarrow \Re$$

where $D = \{(c, a) \mid c \in C \ \wedge \ a \in A(c)\}$

The Top Layer – Total Reuse of Ancestors (TRA)

Definition 3.4 The TRA Layer. *Any metric that measures the total reuse of all the ancestors for a given class $c \in C$, belongs to the TRA layer.* Formally, a metric belongs to this layer if it is defined as:

$$TRA : C \rightarrow \Re$$

Metrics defined in this layer offer a global view of the reuse in a class, allowing comparisons among different classes in the system.

3.3 Layer-Specific Metric Definitions

As we have seen in the Section 3.1, the layers of a MLSM define the skeleton of the system, by rigourously specifying what types of metrics can be part of the system. The next step is to define concrete metrics for each layer. We state earlier that the definition of these metrics is determined by the particular definition used for the attribute. There are mainly two kinds of reuse relation (from an ancestor) that can be measured by using this system of metrics:

- *Implementation reuse*, i.e. to take advantage in a derived class of the structure and/or behaviour implemented in an ancestor class.
- *Interface reuse*, i.e. to implement the interface defined by an ancestor class. (usually the ancestor is abstract)

Metrics for the RAM Layer. Because the bottom layer defines a direct measurement, this is the place where the distinction between the two kinds of reuse will be made. Although, in this paper we decided to define only metrics for the *implementation reuse*, metrics that measure the interface reuse can be defined in the same manner. (The reasons for this decision are exposed at the beginning of Section 4)

Metric 3.1 Unitary RAM Metric *The metric defines the reuse of an ancestor class $a \in A(c)$ in a method mt_c of class $c \in C$ as follows:*

$$RAM_{unit}(mt_c, a) = \begin{cases} 1 & \text{if } \{acm_a \mid acm_a \in ACM(a) \wedge mt_c \text{ uses } acm_a\} \neq \emptyset \\ 0 & \text{otherwise} \end{cases}$$

The RAM_{unit} metric identifies the methods that use a member of an ancestor, but it does neither express the intensity of the reuse, nor does it differentiate between using the interface or the implementation of the ancestor class. These two aspects are covered in the next definition.

Metric 3.2 Percentage RAM Metric. *The metric defines the reuse of an ancestor class $a \in A(c)$ in a method mt_c of class $c \in C$ as follows:*

$$RAM_{perc}(mt_C, a) = \frac{\sum_{i=1}^{n_a^{imp}} uses(acm_i^{imp}, mt_c) + (1 - k_a) \cdot \sum_{j=1}^{n_a^{int}} uses(acm_j^{int}, mt_c)}{n_a^{imp} + (1 - k_a) \cdot n_a^{int}}$$

where acm_i^{imp} $(i = \overline{1, n_a^{imp}})$ are the accessible class members of the ancestor class a that belong to the implementation of the class, acm_j^{int} $(j = \overline{1, n_a^{int}})$ are the accessible class members of ancestor a that belong to the interface of the class, and *uses* is defined as:

$$uses(acm_a, mt_c) = \begin{cases} 1 & \text{if } mt_c \text{ uses } acm_a \\ 0 & \text{otherwise} \end{cases}$$

In spite of the complicated formal aspect the metric is quite simple. The metric is mainly composed of two sums: the first sum is counting how many *distinct* members, belonging to the implementation of an ancestor class $a \in A(c)$, are used in a method of class c, while the second sum counts the members from the interface of the ancestor that are used in that method. Because reuse of implementation is in fact a form of coupling and because the interface of a class is considered being much stabler than the implementation, a *stability factor* $k_a \in [0, 1]$ has been introduced in order to consider this difference when computing the metric. The bottom part of the fraction is included only for the normalisation of the metric.

Metrics for the CAR Layer

Metric 3.3 CAR Metric. *The CAR metric defined between a class $c \in C$ and one of its ancestor classes $a \in A(c)A$ is formally expressed as:*

$$CAR(c, a) = \frac{\sum_{i=1}^{n_c} RAM(mt_i, a)}{n_c}$$

where $mt_i \in M_I(c)$, $i = \overline{1, n_c}$ and $n_c = cardM_I(c)$.

The CAR metric defined above quantifies the reuse from an ancestor by summing the reuse in all the implemented methods of the class. The way reuse is calculated at the method-level is encapsulated in the bottom layer, RAM. Thus, the CAR metric can be computed using any of the two previously defined RAM metrics, or any other metric that belongs to the RAM layer. Notice that the values of this metric are again *relative* numbers, like in the metrics defined for implementation reuse in the RAM layer. We define the CAR metric this way because we considered them more adequate for the measurement of implementation reuse. If we were to define the metric thinking on the interface reuse, we would have probably defined it using an absolute range of values. We should also remark that this metric respects the definition given for a MLSM, as its definition is based on the lower layer of the system.

Metric for the TRA Layer

Metric 3.4 TRA Metric. *The TRA metric for a class $c \in C$ is defined as the sum of the reuses of all the ancestors of that class.* This can be formally expressed as:

$$TRA(c) = \sum_{i=1}^{n_a} CAR(c, a_i)$$

where $a_i \in A(c)$,$i = \overline{1, n_c}$ and $n_c = cardM_I(c)$.

TRA defined in this manner offers a simple way to compute a total reuse value for a class, based on the metric defined for the middle layer CAR. This is for sure not the only possibility to define TRA. For a particular measurement goal, it would be possible to calculate TRA based directly on the definition ofRAM, or to differentiate in the sum between ancestors.

4 The Experimental Study

The special goal of this experimental study is to analyse if and how the proposed MLSM can be used for the re-engineering process of legacy systems[1]. We expect that metrics would help especially during the early phases of the process, i.e. during the phases of *model capture* and *problem detection* [15].

We use for this study the two RAM definitions on implementation reuse. There was a strong reason for focusing our attention on implementation reuse: as we were expected to study legacy systems, we knew that these are mostly designed to the implementation and not to the interface, but. Thus, in such systems inheritance is mainly used to reuse the code and not the interface from the ancestors.

4.1 The Case Studies

Before presenting the experimental results we will briefly describe the case-studies, both in a descriptive and a quantitative manner (Table 1) in order to create a picture of the preconditions of the study and facilitate some correlations between the metrics results and the projects.

Site A is a part of a project that produces software that supports the development of control and command systems. The project is based on some commercial and own-developed frameworks. *Site B* is a software for graphical visualisation of measurement values. The application implements several graphical objects and additional graphical information objects. The run-time optimised display of animated diagrams and the flexibility to expand

[1]This study is part of the FAMOOS project (http://dis.sema.es/projects/FAMOOS/) whose goal is to produce a set of re-engineering techniques and tools that would help the migration of legacy systems to object-oriented frameworks. This explains the motivation behind this experiment

the programs display capacity by adding new graphical objects was one of the major design goals. *Site C* is part of a very large project, and is mainly a simulator for atomic physics experiments. While *Site A* and *Site B* were developed as *Windows NT* applications, *Site C* is designed to run on UNIX machines.

Site Name	Source code [in KB]	Number of classes	Number of inheritance trees	Maximum depth of inheritance tree
Site A	1258	228	45	9
Site B	500	69	12	3
Site C	4620	323	49	5

Table 1. Overview of the case studies

4.2 Viewpoints

1. A low average TRA value for a project might be a sign of poor object-oriented design, that does not take advantage of inheritance-based reuse.
2. If two or more classes have high CAR values in relation to a common ancestor then these classes might constitute a *subsystem* in the project.
3. The outliers with high TRA values lie deeper in the class hierarchy and are mostly *leaf-classes*. We expect the TRA values to increase from the root to the leaves of the class hierarchy.
4. The highest CAR values appear on *direct* inheritance relations.

4.3 Results Summary

Measurements Based on the RAM-Unitary Metric. The summary of the results obtained on the three case-studies when using the RAM_{unit} metric in the bottom-layer of the metrics system is presented below. The table contains the minimum the average and the maximum values for each of the three sites.

Site Name	Minimum	Average	Maximum
Site A	0.02	0.27	1.0
Site B	0.07	0.47	1.0
Site C	0.02	0.35	0.81

Table 2. Results summary for the CAR metric based on $RAM\text{-}Unitary$

Site Name	Minimum	Average	Maximum
Site A	0	0.38	1.90
Site B	0.10	0.56	1.00
Site C	0.10	0.61	1.10

Table 3. Results summary for the TRA metric based on $RAM\text{-}Unitary$

Measurements Based on the RAM-Percentage Metric. The summary of the results obtained on the three case-studies when using the RAM_{perc} metric in the bottom-layer of the metrics system is presented below. The table contains only the average and the maximum values for each of the three sites, because the minimum values can be approximated with zero.

	Site Name	$k_a = 0.0$	$k_a = 0.5$	$k_a = 1.0$
Average	Site A	0.02	0.03	0.06
	Site B	0.11	0.13	0.24
	Site C	0.02	0.02	0.01
Maximum	Site A	0.17	0.23	0.40
	Site B	0.33	0.42	0.58
	Site C	0.13	0.16	0.18

Table 4. Results summary for the CAR **metric based on** *RAM-Percentage*

	Site Name	$k_a = 0.0$	$k_a = 0.5$	$k_a = 1.0$
Average	Site A	0.04	0.05	0.10
	Site B	0.13	0.16	0.20
	Site C	0.04	0.04	0.03
Maximum	Site A	0.24	0.29	0.40
	Site B	0.33	0.42	0.58
	Site C	0.13	0.16	0.18

Table 5. Results summary for the TRA **metric based on** *RAM-Percentage*

4.4 Results Analysis

In all the three case studies the Top 10 outliers both for $CAR-Percentage$ and $TRA-Percentage$ do not change very much when varying the stability factor k_a between $k_a = 0$ and $k_a = 0.5$; but for *Site B* and *Site C* the top did change very much when the ancestor class was considered completely stable ($k_a = 1.0$). Comparing the values for $CAR-Percentage$ for different values of the stability factor for all the three case-studies the order was: $CAR_{perc}^{k_a=0} < CAR_{perc}^{k_a=0.5} < CAR_{perc}^{k_a=1.0}$. Thus the highest factor of code reuse (don't forget these are relative values) was obtained considering the superclass stable and ignoring its interface. The conclusion is that there are a lot of public members in the superclass, but few of them are reused through inheritance relations.

Almost all outliers for total $TRA - Percentage$ are *"light-classes"*, i.e. classes with a small number of methods. This could also be an interesting point for *model capture*.

The Top 10 for the $CAR-Unitary$ strongly differs from the $CAR-Percentage$ top (20% for *Site A*, 30% for *Site B*, 20% for *Site C*). This demonstrates that the two definitions are not correlated, indicating that classes where an ancestor is reused in many methods($RAM-Unitary$), are not the same with the classes where the reuse is more intense($RAM-Percentage$). Thus, we deduce that classes that reuse much of an ancestor class, do it in a small number of methods.

Concerning the average values for the three analysed projects, we found a correlation between these values and the maximum depth of inheritance tree: the deeper the class hierarchy, the smaller the code reuse from ancestors.

4.5 Viewpoints Validation

Analysing the results presented in Table 2 and Table 4 and based on supplementary information concerning the quality of the design, we can confirm that *Site A* is not as well designed compared with the other two sites, sustaining our first viewpoint.

Concerning our second viewpoint, on *Site A* and *Site B*, when using the $RAM-Unitary$, the first half of the top consists of classes that can be *clustered*. Discussions with the developers of the systems confirmed that those clusters are indeed *subsystems*. It still has to be studied if this can be used as a general rule for model capture. Another question that still remains open is, what kind of subsystems are those detected with these metrics.

For all case studies the very most CAR relations involve a leaf-class (*Site B* – 76.0%; *Site B* – 95.8%; *Site C* – 79.3%), confirming the third viewpoint.

The hypothesis that the highest CAR values are between classes that are in a *direct* inheritance relation proved to be true for *Site B* and *Site C* (83.3% and 73%). In *Site A* we encountered an interesting a class, having 55 descendants spread on 4 hierarchy levels. The class is reused by most of its indirect descendant classes, so that approximately the half (47.1%) of the inheritance-reuse relations are indirect.

5 Related Work

As we mentioned already in the beginning of the paper, there is a great interest in the software metrics community towards the problem of defining metrics that are rigourously defined [10] and that can also prove to be useful in the practical experiments. This paper is inspired by some of these contributions. Briand et al. define in [2] and [3] two measurement frameworks – on coupling and cohesion – that offer a systematic vision of the already defined measures. Some other researchers have also proposed in the past measurement frameworks for object-oriented systems [8] [11].

This work is also related to the papers that propose other measurements on inheritance like [7] [5] and [14], but compared to them, we propose metrics that go beyond the high-level design of classes and evaluate the real inheritance-based reuse.

The experimental study in this paper is related to our previous efforts [15] [16] to use metrics to support the re-engineering process.

6 Conclusions and Future Work

There is an urgent need of finding ways to define object-oriented metrics in a rigourous and systematic manner in order to increase their utility and trustworthiness. In this paper we propose a new manner of defining metrics, by organising them in multi-layered systems of metrics (MLSM). The major advantage of applying this approach is that we obtain a unitary definition of related metrics definitions, and a systematic, yet flexible manner of defining new metrics as part of a particular metrics system. Another advantage of using MLSM's is that it dictates an an organised way of thinking about the metric definitions in a given measurement context. Applying the “Reuse of Ancestors” MLSM to re-engineering indicated us that there is a good opportunity to use the metrics for this purpose.

Rather than pretending that this is a fulfilled work, we consider it a beginning. The continuation of this work must necessarily include a theoretical validation of the proposed system of metrics together with a continuation of the experimental studies (empiricial validation). The objective of the theoretical validation will be to prove if the proposed metrics do accurately measure the attributes they were designed to measure. For this, we intend to focus our attention on the criteria proposed by Kitchenham et al. [13] and on the properties defined by Briand et al. [4]. Concerning the empirical validation we intend to conduct a rigourous experimental study, based several on real-world legacy systems, aiming to investigate the usefulness of these metrics especially for design-flaws detection in legacy systems. We believe that these experiments must rely on the principles of empirical investigation [9].

We hope that in the future further such MLSM's will be proposed, as we believe that organising metric definitions in systems of metrics, will contribute to that strongly needed order among object-oriented metrics, increasing thus their reliability and usability.

References

[1] B.W. Boehm, J.R. Kapsar, et. al *Characteristics of Software Quality* TRW Series of Software Technology, Amsterdam, North Holland, 1978

[2] L.C. Briand, J.W. Daly, J.Wuest *A Unified Framework for Coupling Measurement in Object-Oriented Systems.* Fraunhofer Institute for Experimental Software Engineering Kaiserslautern, ISERN-96-14, 1996

[3] L.C. Briand, J.W. Daly, J.Wuest *A Unified Framework for Cohesion Measurement in Object-Oriented Systems.* Fraunhofer Institute for Experimental Software Engineering Kaiserslautern, ISERN-97-05, 1997

[4] L.C. Briand, S.Morasca, V.R. Basili. *Property-Based Software Engineering Measurement.* IEEE Transactions on Software Engineering, 22(1), January 1996

[5] F.Brito e Abreu. *The MOOD Metrics Set.* Proc. ECOOP'95 Workshop on Metrics, 1995

[6] J.A. McCall et. al. *Factors in Software Quality.* RADC TR-77-369, Vols. I, II, III, 1977

[7] S.R. Chidamber, C.F. Kemerer. *A Metrics Suite for Object-Oriented Design.* IEEE Transactions on Software Engineering, 20(6), June 1994

[8] J. Eder, G. Kappel, M. Schrefl. *Coupling and Cohesion in Object-Oriented Systems.* Technical Report, University of Klagenfurt, 1994

[9] N.E. Fenton. S.L. Pfleeger. *Software Metrics, A Rigorous & Practical Approach.* PWS Publishing Company, 1997.

[10] N.E. Fenton. *Software Metrics, A Rigorous Approach.* Chapman & Hall, New York 1991.

[11] M. Hitz, B. Montazeri. *Measuring Coupling and Cohesion in Object-Oriented Sytems..* Proc. Int. Symposium on Applied Corporate Computing, Monterrey, Mexico, 1995.

[12] International Standards Organization. *Information Technology – Software Product Evaluation – Quality Characteristics and Guide Lines for their Use* ISO/IEC 9126, Geneva, Switzerland, 1991

[13] B.Kitchenham, S.L. Pfleeger, N. Fenton *Towards a Framework for Software Measurement Validation.* IEEE Transactions on Software Engineering, 21(12), December 1995

[14] M. Lorenz, J. Kidd *Object-Oriented Software Metrics.* Prentice-Hall Object-Oriented Series, Englewood Cliffs, NJ, 1994

[15] R. Marinescu. *The Use of Software Metrics in the Design of Object-Oriented Systems.* Diploma thesis, "Politehnica" University of Timisoara, October 1997.

[16] R. Marinescu. *An Object-Oriented Metrics Suite on Coupling.* Master thesis, "Politehnica" University of Timisoara, September 1998.

[17] S. A. Whitmire. *Object-Oriented Design Measurement.* Wiley, ISBN-0471134171, September 1997.