Investigations on the Properties of Object-Oriented Software Metrics

Thomas Fetcke

Abstract

This paper presents basic methods to analyze the properties of object-oriented software metrics. The metrics are characterized with several concatenation operations on different levels of abstraction. Metrics can thereby be interpreted above the ordinal scale level. The result of this investigation is that a large set of object-oriented metrics have properties that are completely different from properties of metrics for procedural languages. This set of metrics follows the Dempster-Shafer measure of belief.

1 Introduction

Software metrics were introduced in order to analyze the complexity of programs and to predict the effort for development. To determine the properties of software metrics, we use measurement theory as a tool that allows us to relate numerical properties of the metrics back to the underlying empirical objects. A metric constitutes a mapping from the empirical domain, i. e. the software objects, into a numerical domain, normally the real numbers $\Re$. Under the assumption that the metric can be used as an ordinal scale, this mapping gives a preference relation on the empirical objects.

If we add concatenation operations on the empirical objects to our model, we can obtain information above the ordinal level. Zuse showed in [8] that many metrics in the procedural domain assume the extensive structure, which makes it possible to use these metrics as a ratio scale. In the object-oriented domain, the properties of a large set of metrics are completely different from those of metrics for procedural software.

Section 2 describes the empirical software objects in the object-oriented domain and introduces concatenation operations on them. Section 3 discusses the OO metrics in conjunction with the concatenation operations, that can be described with the Dempster-Shafer measure of belief.

2 Empirical Software Objects

In order to discuss the properties of object-oriented software metrics, we first have to specify the target of the measurement. The object-oriented model allows measurement on different levels of abstraction. The following four levels will be considered.

- **Classes**: A class describes the properties of objects with attributes (often called instance variables) and methods. A class has a set of attributes and a set of methods defined in it. A large set of metrics apply to classes.
- **Methods**: Methods characterize the behaviour of an object. In many object-oriented models, methods are constructed as subroutines. Metrics from the procedural domain that are defined on the control flow can be applied to these methods.
- **Inheritance Hierarchies**: Classes inherit the properties of superclasses along the inheritance relationship. The inheritance relationship forms a graph on classes. An OO system can contain a couple of unconnected subgraphs of the inheritance relationship. For the measurement theoretic analysis we consider only the subset of all connected graphs with a single root class, multiple inheritance is allowed.
- **Uses Hierarchies**: The uses relationship forms similarly to the inheritance relationship possibly unconnected subgraphs on the set of classes. Again, we consider the subset of those graphs constituted by the uses relationship, that have a single main class, i. e. a class that is not used by another class, and that are acyclic and connected.

The restrictions for the last two levels are necessary for the completeness of the concatenation operations defined below. These restrictions do not limit the expressiveness of the properties found. In real OO systems, we can find several of these hierarchies. The application of metrics on these hierarchies can still be useful.

2.1 Concatenation Operations

2.1.1 Concatenation on Class Level

Chidamber and Kemerer [1] introduce a concatenation operation for classes in order to study the properties of their measures. However, the authors base their investigation not on the extensive structure. They consider the Weyuker [6] properties. It should be mentioned here, that the Weyuker properties are not compatible (see [8, chapter 6]).

We will use the operation defined by Chidamber et. al. for our studies, naming it CUNI to distinguish it from the additional concatenation operations defined below.

CUNI: Class Unification The unification of two classes is no means of expression in object-oriented models. The result of combining two classes is a single new class. This new class combines all the properties of the two single classes, i. e. all attributes and methods of either class, where properties both classes have in common only appear once in the combined class (see fig. 1).

Many metrics proposed in literature also use the inheritance and uses relationships to characterize a class. Therefore, the definition of the concatenation operation is extended to these relationships. The new class inherits all the properties that the separate classes inherited and all classes inheriting from one of the separate classes inherit form the combination. All the classes used by either one or both of the classes combined will be used by the unified class. This lies already in the unification of the properties defined in the class. Analogously, we define all classes that used any single or both separate classes to use the combined class.

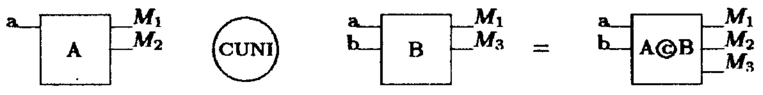

Figure 1: Concatenation of classes with CUNI

The properties of CUNI are very similar to the properties of a unification of sets, as CUNI is defined upon those unifications. CUNI is commutative and it is also associative. CUNI is also idempotent, i. e. for a class A

$$CUNI(A, A) = A$$

holds, which is an interesting property. In consequence, no measure can assume an extensive structure and we cannot construct a ratio scale this way.

Of course, this property of CUNI is not by an accident. CUNI has been defined as an ***unification*** and unification of sets is idempotent. This can be seen as an argument for the unification of similar classes, so that the effort for maintenance can be reduced.

Chidamber et. al. check, whether their measures agree with the Weyuker properties. If we keep in mind that CUNI is idempotent, it is not surprising that none of their six metrics agree with Weyuker's property of wholeness, which is defined as

$$\mu(A \circ B) \geq \mu(A) + \mu(B).$$

If $A = B$, we get

$$\mu(CUNI(A, A)) = \mu(A) \not\geq 2\mu(A).$$

As mentioned above, this is due to CUNI being a kind of unification. The suggestion is to define a second concatenation operation similar to the intersection of sets.

CINT: Class Intersection The result of an intersection of two classes is that only those properties common to both classes are reduced to one property in the resulting class (see fig. 2). As for CUNI, we have to consider both inheritance and uses relationships for CINT. The new class inherits the intersection of properties that both classes inherit. That requires the new class to inherit from all classes that are direct ***or indirect*** superclasses of both classes, possibly leaving out intermediate classes. And any subclass common to both classes will be a subclass of the new class.

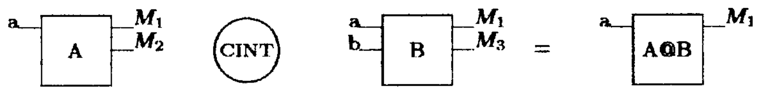

Figure 2: Class intersection with CINT

The new class will use all classes used by both classes and shall be used by classes using both separate classes. Unlike inheritance, classes using indirectly do not use the new class.

It is important to say that CINT is not intended to be used to build new stand alone classes, but it can be used as a tool to express properties of classes and metrics.

The Empty Class We define the empty class $\emptyset$ to be a class with no properties, i. e. empty sets of attributes and methods. Furthermore, the empty class may not inherit any property and no class inherits from it. It is not used by any class and of course does not use any class.

We now can say that two classes A and B are disjunct, iff

$$CINT(A,B) = \emptyset.$$

2.1.2 Concatenation of Methods

As stated above, methods can be seen as subprograms in most OO systems. This allows us to use traditional procedural metrics. Under these conditions, concatenation operations for methods can be found similar to the concatenation of procedural subroutines, using e. g. the Dijkstra-Structures sequence and alternative.

It comes up that McCabe's cyclomatic complexity and the metric Lines of Code (LOC) together with sequential concatenation assume the extensive structure. The same applies for these metrics with the concatenation of methods via alternative. At this level of the OO model, the properties of metrics are similar to the procedural model.

2.1.3 HAGG: Hierarchical Aggregation

This concatenation operation is defined on the uses hierarchies defined above. Two such independent hierarchies are combined to form a new hierarchy. As defined, both hierarchies have a single main class. Both these classes will be used by the new main class that is added to the hierarchy. Therefore, the new object is again a hierarchy as defined.

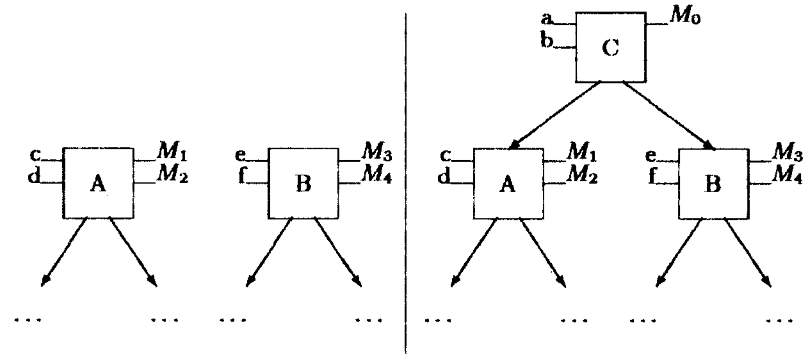

Figure 3: Concatenation of uses hierarchies with HAGG

HAGG is commutative as the order of uses is irrelevant, but HAGG is not associative. It is possible to find metrics that agree with the axioms of the extensive structure together with HAGG, e. g. the association complexity metrics defined in [3].

2.1.4 HGEN: Hierarchy Generalization

Two independent inheritance hierarchies as defined above are combined to form a new inheritance hierarchy. As defined, these hierarchies have a single root class being a superclass of all classes in the hierarchy. Let A and B be these root classes, HGEN builds a new class G by generalization of the classes A and B. This new class becomes the new root of the resulting hierarchy.

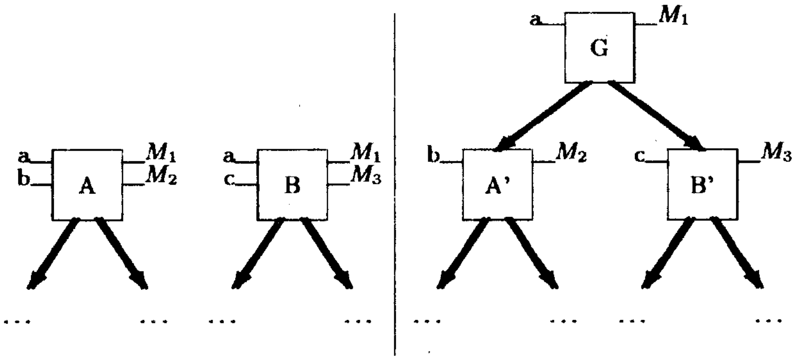

Figure 4: Concatenation of inheritance hierarchies with HGEN

Here, we meet CINT again. The new root comprises all properties that the classes A and B have in common. It holds

$$G = CINT(A, B).$$

The classes A and B need no more define the properties contained in G as they inherit those. Therefore, the properties defined in classes A and B respectively are those lying outside the intersection. We denote this case by creating two new subclasses of G named A' and B' instead of A and B. This denotes the change in properties defined ***in the class itself***, although the actual sets of properties remain unchanged under consideration of inheritance.

With CUNI and CINT, we can describe these classes

$$A = CUNI(G, A'), B = CUNI(G, B') \text{ and}$$
$$CINT(G, A') = CINT(G, B') = \emptyset.$$

HGEN is commutative and not associative. It is possible to construct metrics that assume the extensive structure with HGEN, although this is normally not the case for metrics proposed in literature.

3 Properties of Metrics on Class Level

Together with the two operations CUNI and CINT, a couple of metrics on class level hold the following equation. Let μ be a metric and A, B some classes, then

$$\mu(CUNI(A, B)) = \mu(A) + \mu(B) - \mu(CINT(A, B)).$$

HAGG is commutative as the order of uses is irrelevant, but HAGG is not associative. It is possible to find metrics that agree with the axioms of the extensive structure together with HAGG, e. g. the association complexity metrics defined in [3].

2.1.4 HGEN: Hierarchy Generalization

Two independent inheritance hierarchies as defined above are combined to form a new inheritance hierarchy. As defined, these hierarchies have a single root class being a superclass of all classes in the hierarchy. Let A and B be these root classes, HGEN builds a new class G by generalization of the classes A and B. This new class becomes the new root of the resulting hierarchy.

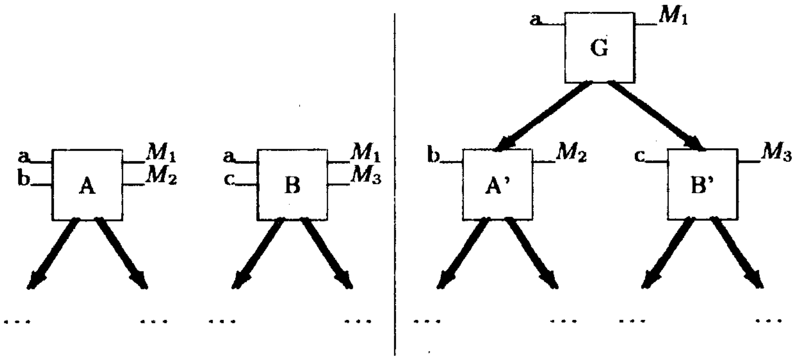

Figure 4: Concatenation of inheritance hierarchies with HGEN

Here, we meet CINT again. The new root comprises all properties that the classes A and B have in common. It holds

$$G = CINT(A, B).$$

The classes A and B need no more define the properties contained in G as they inherit those. Therefore, the properties defined in classes A and B respectively are those lying outside the intersection. We denote this case by creating two new subclasses of G named A' and B' instead of A and B. This denotes the change in properties defined ***in the class itself***, although the actual sets of properties remain unchanged under consideration of inheritance.

With CUNI and CINT, we can describe these classes

$$A = CUNI(G, A'), B = CUNI(G, B') \text{ and}$$
$$CINT(G, A') = CINT(G, B') = \emptyset.$$

HGEN is commutative and not associative. It is possible to construct metrics that assume the extensive structure with HGEN, although this is normally not the case for metrics proposed in literature.

3 Properties of Metrics on Class Level

Together with the two operations CUNI and CINT, a couple of metrics on class level hold the following equation. Let μ be a metric and A, B some classes, then

$$\mu(CUNI(A, B)) = \mu(A) + \mu(B) - \mu(CINT(A, B)).$$

These metrics can be described with the theory of belief functions[4, 5, 7], which is an extension of probability theory. It is important to say that software metrics are not meant to be uncertain or probabilistic measures. The theory of belief functions is treated here as a representation theorem. Given a function of belief, a corresponding qualitative belief relation can be constructed, that totally agrees with that belief function. This belief relation can be interpreted as a preference relation in the area of software metrics. The axioms for the relation of belief give certain properties that give more information than the ordinal scale. We thus have additional characteristics of these metrics. We also know that these metrics cannot assume an extensive structure.

The theory of belief functions is thus used as a tool to derive properties of the qualitative relation on software objects (here: classes). In fact, the axioms for belief functions and belief relations have been modified to the unlimited range $\Re$ of software metrics, as shown in definition 1 and 2.

Definition 1 (Modified Function of Belief) *Let X be a countable Set and Γ the set of all finite subsets of X. A measure $\mu : \Gamma \mapsto \Re$ is a* modified function of belief *iff*

$$\mu(\emptyset) = 0 \tag{1}$$
$$\forall A \in \Gamma : \mu(A) \geq 0 \tag{2}$$
$$\mu(A_1 \cup A_2 \cup \ldots \cup A_n) \geq \sum_{\substack{I \subseteq \{1 \ldots n\} \\ I \neq \emptyset}} (-1)^{|I|+1} \mu\left(\bigcap_{i \in I} A_i\right) \tag{3}$$

The set X stands for the set of all possible properties (attributes and methods) of a class. Finite subsets of X are classes, Γ is therefore the set of all classes. The unification operator $\cup$ stands for CUNI and $\cap$ represents CINT.

Definition 2 (Modified Relation of Belief) *Let $\succeq$ be a relation on Γ. $\succeq$ is a* modified relation of belief, *iff*

$$\forall A, B \in \Gamma : A \succeq B \vee B \succeq A \tag{4}$$
$$\forall A, B, C \in \Gamma : A \succeq B \wedge B \succeq C \Longrightarrow A \succeq C \tag{5}$$
$$\forall A \supseteq B \Longrightarrow A \succeq B \tag{6}$$
$$\forall (A \supset B, A \cap C = \emptyset) \Longrightarrow (A \succ B \Longrightarrow A \cup C \succ B \cup C) \tag{7}$$
$$\forall A \in \Gamma : A \succeq 0 \tag{8}$$

Given these definitions, the following theorem holds.

Theorem 1 *It exists a modified function of belief which fulfills (1-3), such that*

$$A \succeq B \Longleftrightarrow \mu(A) \geq \mu(B),$$

iff $\succeq$ fulfills axioms (4-8).

The proof for theorem 1 can be found in [2]

4 Conclusion

Object-oriented software metrics have other properties than procedural metrics. Many object-oriented metrics do not assume an extensive structure. We have seen that a set of metrics can be described with unification and intersection operations and the function of belief. These investigations should be seen as a first foundation for the analysis of properties of object-oriented metrics.

References

[1] Chidamber, Shyam R.; Kemerer, Chris F.: *A Metrics Suite for Object Oriented Design*, IEEE Transactions on Software Engineering, Vol. 20, No. 6, 1994, pp. 476–793

[2] Fetcke, Thomas: *Softwaremetriken bei objektorientierter Programmierung*, Diploma thesis, Gesellschaft für Mathematik und Datenverarbeitung (GMD), St. Augustin and TU Berlin, 1995

[3] Kolewe, Ralph: *Metrics in Object-Oriented Design and Programming*, Software Development, October 1993, pp. 53-62

[4] Shafer, Glenn: *A Mathematical Theory of Evidence*, Princeton University Press, 1976

[5] Shafer, Glenn: *Belief Functions and Possibility Measures*, In Analysis of Fuzzy Information, Vol. 1, Mathematics and Logic, CRC Press, 1987, pp. 51–84

[6] Weyuker, Elaine J.: *Evaluating Software Complexity Measures*, IEEE Transactions of Software Engineering, Vol. 14, No. 9, 1988

[7] Wong, S. K. M; Yao, Y. Y.; Bollmann, P.; Bürger, H. C.: *Axiomatization of Qualitative Belief Structure*, IEEE Transactions on Systems, Man and Cybernetics, Vol. 21, No. 4, 1991

[8] Zuse, Horst: *Software Complexity; Measures and Methods*, DeGruyter, Berlin, New York, 1991

[9] Zuse, Horst; Fetcke, Thomas: *Properties of Object-Oriented Software Measures*, Proc. 7th Annual Oregon Workshop on Software Metrics, 1995

About the Author

Thomas Fetcke was from August to December 1994 as a graduate student with the Gesellschaft für Mathematik und Datenverarbeitung in St. Augustin. There he wrote his diploma thesis on object-oriented software metrics. In May 1995, he received his diploma degree in computer science from the Technical University Berlin. Address: Thomas Fetcke, Innsbrucker Str. 33, D-10825 Berlin; e-mail: fetcke@cs.tu-berlin.de.