On Using Metrics in the Evaluation of Aspect-Oriented Programs and Designs

Katharina Mehner[*]
Software Engineering Group
Technical University of Berlin
Germany
mehner@cs.tu-berlin.de

ABSTRACT

Metrics are an important technique in quantifying desirable software and software development characteristics of aspect-oriented software development (AOSD). Currently, metrics proposed for AOSD have rarely been validated. We give an overview on the necessary steps to validate definitions and applications of metrics. We also compare definitions for proposed metrics.

1. INTRODUCTION

In order to study the impact of aspect-oriented software development (AOSD) on evolution one has to study its impact on software characteristics such as evolvability, maintainability, understandability, and quality. To evaluate these characteristics not only qualitative but also quantitative techniques are of interest.

The key question is how we can quantify when applying AOSD is beneficial. *Metrics* are an important techique in quantifying software and software development characteristics. However, metrics have to be used knowledgeably and carefully. Theoretical and empirical validation of metrics and of their relation to software attributes is a cumbersome and long process. It is of paramount importance that we validate the *utility* of metrics we use in order to enable others to use them, too. Until now, the metrics used and proposed for AOSD are rarely validated. It is not sufficient to prove their definitions correct but also their usefulness to describe software characteristics has to be validated. In most cases, this can only be achieved through controlled experiments or through analysing large amounts of data, e.g., from case studies. In both cases, statistical evaluations are a key technique to examine hypotheses.

In the remainder of this position paper, we first make our problem statement more concrete. We give an overview on the underlying theory of metrics and their validation. We focus on product metrics but much of what will be said equally applies to process, resource, or project metrics. Here, we cannot provide a thorough theoretical discussion but we want to raise the level of awareness concerning the issues involved in using metrics. We then discuss and compare some concrete metric examples, which are often influenced by OO metrics. We conclude by calling for more empirical research in relation with metrics in AOSD.

[*]The author has been supported by the German Federal Ministry for Education and Research under the grant 01ISC04A (Project TOPPrax).

2. PROBLEM STATEMENT

The benefit of AOSD is often studied in two ways. One is the comparison with an OO software, which is already existing or which sometimes is developed in paralled. The other one is the independent evaluation of AOSD, often by assessing evolution scenarios.

In both cases, the evaluation can employ design and program metrics to support and to quantify their observations. It has already noted by Chidamber and Kemerer that because of the abstractions introduced in OO, one can distinguish between design and code metrics [4]. The benefit of this distinction is that design metrics, although applicable also to code, can already be applied to the design when the coding has not yet begun. Apart from the difference, the validation requirements are the same. In the following, we discuss design and program metrics together. As already mentioned in the introduction, validation is twofold.

The *theoretical* or internal validation addresses the proper definition of a metric and guarantees that a metric fulfils generally accepted (axiomatic) properties (for an overview see [5]). Often, metrics are defined using the representational theory of measurement. In this theory, measurement is regarded as "the correlation of numbers with entities that are not numbers". For an empirical attribute, there should be an empirical observation of a relation in order to consider defining a metric. Then one can try to establish a mapping to a numerical presentation which preserves empirical relations. To ensure preservation of relations and proper definition a number of axioms or properties have to be fulfilled [5]. Using this method, it can be avoided that for a metric we desperately seek an empirical counterpart. There exist competing and sometimes contradictory sets of axioms or properties for validating metric definitions, for instance the axioms by Weyuker [12] or the validation framework by Kitchenham et al. [8].

The *empirical* or external validation of a metric addresses its utility in describing desirable properties or predicting desirable properties of software (for an overview see [5]). For a metric to be useful in assessing a software product or process quality, i.e., a characteristic, we need a validation that shows that this metric has a de facto influence on a characteristic or that it serves in predicting characteristics of a process. Such a cause-effect relation can not be validated by merely statistically proving a correlation, but a correlation can be an indicator as to what should be examined. Instead, an experiment with a falsifiable hypothesis has to be carried out, or data gathered from case studies or real projects have to be evaluated. Empirical validation thus can be based either on controlled experiments or on case studies which produce enough data or data collected from real projects. The use of statistical tests is helpful to discover correlations between variables of controlled experiment [8, 5, 3].

Empirical validation is also required when a *quality model* is used to define relationships between metrics and software characteristics. For instance, the Goal-Question-Metric (GQM) model is used to identify what must be measured in order to answer the question from which the goal of an empirical study can be determined [2]. This model helps to define and select appropriate metrics. However, especially the relation of the questions to the metrics has to be empirically validated for each new model derived.

Given these steps for validating metrics, the state-of-the-art in metrics for AOSD can be assessed. Metrics related to desirable design and program characteristics have been proposed amongst others by Lopes [9], Zhao [14], Garcia [10], and Ceccato [11]. Individual examples therof will be shortly discussed in Sect. 3. The theoretical validation has only been addressed in some of these publications. Empirical validation is even harder to achieve and has been rarely addressed. The research by [10] has included an empirical validation which indicates the usefulness of the proposed metrics through two case studies. This is a first step towards an empirical validation of the metrics. The authors acknowledge that this does not yet provide a complete validation. It would be desirable to accomplish their studies by planned, controlled experiments to validate their hypotheses. Note that controlled experiments in AOSD have been used to gather qualitative data as in [7, 1].

Some of these approaches use metrics in the comparison of AO and OO design. Here, the challenge is to validate that metrics which have been extended from OO to AO permit the direct comparison. Without empirical validation it is particularly dangerous to interpret values of metrics as better or worse than other values with respect to software characteristics. One has to be careful with taking such statements for granted.

3. METRICS FOR AOSD

The examples discussed in the following present by no means a complete overview. We have selected them to show that different ideas exist on how to define AOSD metrics.

For defining AOSD metrics it is obvious to build on existing general software metrics and especially on existing OO metrics [4]. Thus, it is a key issue, how to extend metrics and how to define corresponding metrics besides defining completely new metrics. Most existing metrics cannot be applied straightforwardly to aspect-oriented software, since AOSD introduces new abstractions and new composition techniques. Consequently, there are new kinds of coupling and cohesion. Not only new metrics but also extended metrics to cover a new programming language have to be validated.

Some characteristics and attributes have been widely acknowledged as playing a key role in the evolability of software, both for OO and AO, such as size, coupling, cohesion and separation of concern, and because they are obviously related to aspect-oriented modularisation of crosscutting concerns. We can observe two approaches with respect to defining metrics for these characteristics, *extending* metrics and *defining new* metrics. Here we shortly sketch the different ideas behind the two approaches.

Separation of concern metrics

Separation of concern metrics are new ones. In [9] the first separation of concern metrics were proposed. In [10, 6] these metrics have been refined. Until now, these metrics requires manually identifying concerns.

Coupling metrics

Coupling is interesting because it is treated both ways. Initially, the coupling-between-objects metric [4] is defined as counting all classes (once) to which a class is coupled. This is in turn defined by counting the classes of the objects on which a given object/class "acts upon". This refers to access or method calls on instance variables, local variables or formal parameters. Although the metric relies on different kinds of variables or parameter, essentially the same kind of access is counted, i.e., that an instance of another class is accessed via a reference and potentially via its attributes or methods. Also note that in the OO context, high numbers of coupling are considered as undesirable.

In [13] no new metrics are defined but the preservation of the existing definitions is assumed. They focus on the effects aspects typically have on these metrics, e.g., they discuss that OO coupling may be decreased. Thus, their work implies the necessity to be able to understand the OO part on its own, e.g. by preserving OO metrics.

In [10, 6], this original metric is extended to cover also coupling between aspects. This means that the metric still counts the same kinds of coupling as before, i.e., coupling to classes used in declarations of attributes, parameters and local variables, but now includes declarations that belong to aspects.

Other authors [11] have decided to provide separate metrics for the new kinds of couplings found in AO. The metric can be used as an indicator for different effects of the different kinds of coupling in a validation experiment. Although the different metrics mentioned so far, coupling for OO and coupling for AO, are all related to coupling, it should be avoided to compare them directly as they describe coupling related to two different paradigms.

Cohesion metrics

Cohesion has been addressed in both ways, too. [14] specifically looks at a new way for defining cohesion. Others extend existing metrics such as lack-of-cohesion in methods straightforwardly to advice [10].

Size metrics

Size metrics are often critized as being dependent on programming styles and being too simple. Size itself has an empirical counterpart in the physical length though there are numerous ways to define it, e.g., LOC. From this viewpoint it might be admissable to count a pointcut as a code line. In [10, 6], experience with an extended LOC metric has been gathered.

Size metrics can point to the fact that duplicated code is avoided. The question is, if LOC is a good indicator for maintainability and reusability when comparing AO with OO. Avoiding duplicate code might be achieved by writing difficult-to-maintain pointcuts.

4. CONCLUSION

This paper gave a short overview on the necessary steps for validating metrics that are to be used in an evaluation process. These steps are well-known in software engineering. The current state-of-the-art in AOSD is that one has started to work on the definition of apparently useful metrics. Now it is time to start with completing this research by providing empirical results. This will enable a larger to community to use AOSD metrics and more importantly, understand the benefits of AOSD. Thus, we have to strive for planning and carrying out the corresponding experiments. The results may also give hints as to for which purposes metrics extensions are useful and for which purposes separate metrics are useful.

5. REFERENCES

[1] E. L. A. Baniassad, G. C. Murphy, C. Schwanninger, and M. Kircher. Managing crosscutting concerns during software evolution tasks: An inquisitive study. In *1st International Conference on Aspect-Oriented Software Development (AOSD)*, pages pp. 120–126, 2002.

[2] Victor R. Basili. Software Modeling and Measurement: The Goal Question MetricParadigm. In *Computer Science Technical Report Series, CS-TR-2956(UMIACS-TR-92-96), University of Maryland*, 1992.

[3] Coral Calero, Mario Piattini, and Marcela Genero. Method for obtaining correct metrics. In *ICEIS (2)*, pages 779–784, 2001.

[4] Shyam R. Chidamber and Chris F. Kemerer. A metrics suite for object oriented design. *IEEE Trans. Software Eng.*, 20(6):476–493, 1994.

[5] Norman Fenton and Shari Pfleeger. *Software Metrics: A Rigorous and Practical Approach (2nd edition).* 1997.

[6] A. Garcia, C. Sant'Anna, E. Figueiredo, U. Kulesza, C. Lucena, and A. v. Staa. Modularizing design patterns with aspects: A quantitative study. In *International Conference on Aspect-Oriented Software Development (AOSD)*, pages pp. 3–14, 2005.

[7] Mik Kersten and Gail C. Murphy. Atlas: A case study in building a web-based learning environment using aspect-oriented programming. In *OOPSLA99*, pages 340–352, 1999.

[8] Barbara Kitchenham, Shari Lawrence Pfleeger, and Norman E. Fenton. Towards a framework for software measurement validation. *IEEE Trans. Software Eng.*, 21(12):929–943, 1995.

[9] Christa V. Lopes. *D: A Language Framework for Distributed Programming.* PhD thesis, College of Computer Science, Northeastern University, Nov. 1997.

[10] Claudio Sant'Anna, Alessandro Garcia, Christina Chavez, Carlos Lucena, and Arndt von Staa. On the Reuse and Maintenance of Aspect-Oriented Software: An Assessment Framework. In *XVII Brazilian Symposium on Software Engineering*, 2003.

[11] Paolo Tonella and Mariano Ceccato. Aspect mining through the formal concept analysis of execution traces. In *WCRE*, pages 112–121. IEEE Computer Society, 2004.

[12] Elaine J. Weyuker. Evaluating software complexity measures. *IEEE Trans. Software Eng.*, 14(9):1357–1365, 1988.

[13] A. Zakaria and H. Hosny. Metrics for Aspect-Oriented Software Design. In *Workshop on Aspect-Oriented Modeling AO'03*, 2003.

[14] Jianjun Zhao and Baowen Xu. Measuring aspect cohesion. In Michel Wermelinger and Tiziana Margaria, editors, *FASE*, volume 2984 of *Lecture Notes in Computer Science*, pages 54–68. Springer, 2004.