On the Application of Software Metrics to UML Models

Jacqueline A. McQuillan and James F. Power

Department of Computer Science, National University of Ireland, Maynooth, Co. Kildare, Ireland
{jmcq,jpower}@cs.nuim.ie
http://www.cs.nuim.ie/research/pop/

Abstract. In this position paper we discuss a number of issues relating to model metrics, with particular emphasis on metrics for UML models. Our discussion is presented as a series of nine observations where we examine some of the existing work on applying metrics to UML models, present some of our own work in this area, and specify some topics for future research that we regard as important. Furthermore, we identify three categories of challeges for model metrics and describe how our nine observations can be partitioned into these categories.

Keywords: software metrics, object-oriented systems, UML, metamodels.

1 Introduction

Many object-oriented metrics have been proposed specifically for the purpose of assessing the design of a software system. However, most of the existing approaches to measuring these metrics involve the analysis of source code. As a result, it is not always clear how to apply existing metrics at the early stages of the software development process. With the increasing use of the Unified Modelling Language (UML) to model object-oriented systems at the early stages of the software development process, research is required to investigate how the metrics can be measured from UML models and prior to the implementation of the system.

Being able to measure the metrics accurately from both UML models and source code is important for several reasons:

- The quality of the system can be assessed in the early stages of the software life-cycle when it is still cost effective to make changes to the system.
- The implementation can be assessed to determine where it deviates from its design. This can be achieved by applying metrics to both the UML and source code and comparing the results. Variations in the metric values may help to identify parts of the implementation that do not conform to its design.
- Evaluation of the correctness of round trip engineering tools can be performed. Again, applying the same metrics to both the UML and source code

T. Kühne (Ed.): MoDELS 2006 Workshops, LNCS 4364, pp. 217–226, 2007.

Appears in Models in Software Engineering
Springer Lecture Notes in Computer Science, Vol. 4364

may help in identifying parts of the system that have been incorrectly forward or reverse engineered.

In this position paper we review some of the existing work on applying metrics to UML models, present some of our own work in this area, and outline some topics for future research that we regard as important. However, in order to serve as a basis for discussion, we have chosen to present this position paper as a series of nine observations.

2 General Observations

In this section we present three basic observations regarding the nature of metric definitions and calculation at the model level. The observations themselves are hardly contentious, but they serve as a framework for discussing related work in the area.

Observation 1. Defining model metrics is a metamodelling activity.

Many metrics for object-oriented software have been proposed in the literature [1,2]. However, one of the difficulties with comparing and evaluating these metrics is in interpreting and understanding their exact definition. For example, when counting methods in a class, should constructors, finalisers/destructors and accessor methods count as ordinary methods? Should methods that are inherited but not defined in a class be included? Should abstract methods count as empty methods, or not at all? In order to answer these questions, it is necessary to model the entities being measured, and to then define the metrics in terms of this model. In standard terminology, metrics are defined on the metamodel of the entities being measured.

Several attempts have been made to address the problem of ambiguous metric definitions. Briand *et al.* propose an integrated measurement framework, based on a model of object-oriented systems, for the definition, evaluation and comparison of object-oriented coupling and cohesion metrics [3,4]. Harmer and Wilkie have developed an extensible metrics analyser tool for object-oriented programming languages based on a general object-oriented programming language metamodel in the form of a relational database schema [5]. Reißing defines metrics over a formal model called ODEM (Object-oriented DEsign Model) which consists of an abstraction layer built upon the UML metamodel [6].

Our own work uses a middle level model to define metrics over Java programs [7]. By defining metrics on this metamodel i.e. at the meta-level, we were able to quickly specify and implement a number of different versions of *cohesion* within a class, and evaluate the metrics over a number of large software systems.

Observation 2. Implementing metrics that are defined at the meta-level is (almost) free.

Using a clearly defined metamodel is important for facilitating unambiguous definitions of metrics, but it also has clear advantages in terms of implementation.

Many metamodelling frameworks facilitate the implementation of corresponding APIs that allow for the representation and traversal of model instances. The canonical example is the XML Metadata Interchange (XMI) for OMG's MetaObject Facility (MOF) [8], but closely related frameworks include the Eclipse Modelling Framework (EMF)[1] and the NetBeans Metadata Repository project (MDR)[2].

Previous research has exploited the implementation aspect of metamodels by defining metrics as queries. El-Wakil *et al.* propose the use of XQuery as a metric definition language to extract metric data from XMI design documents, specifically UML designs [9]. Harmer and Wilkie, working from a relational schema, express metric definitions as SQL queries over this schema [5]. Baroni *et al.* have built a library called FLAME (Formal Library for Aiding Metrics Extraction) that uses the Object Constraint Language (OCL) and the UML metamodel as a mechanism for defining UML-based metrics [10]. Goulão *et al.* have utilised this approach for defining component based metrics and used the UML 2.0 metamodel as a basis for their definitions [11].

In our own work, we have specified outline metrics on UML class diagrams, using OCL queries over the UML 2.0 metamodel [12]. The scope of such metrics is somewhat limited, since many of the features they measure relate to method internals, which are not available in class diagrams. Nonetheless, a prototype tool, *dMML*, was developed as an Eclipse plug-in to implement and measure these metrics [13].

However, some issues still exist. Assumptions have to be made when specifying how to instantiate the metamodels, such assumptions will have an effect on the metric definitions. In addition, the process of creating instances of the metamodels must be verified. Errors or omissions in this process would have a fundamental impact on the correctness of the calculated metrics.

Observation 3. Defining new metrics is (almost regrettably) easy.

One of the problems with software metrics is that they can be easy to define, but difficult to justify or correlate with external quality attributes. For example, Halstead's metrics [14] are often cited, but almost equally often criticised. Working at the model level provides a whole new layer of elements and relationships that can be grouped and counted. However, it is important to avoid the trap of proposing metrics that count these elements without offering evidence that such counts are really useful in evaluating the model. Much of the literature on the proposal of metrics for UML models has concentrated on only one or a small number of the different diagrams and views available in an overall UML specification of a software system. Furthermore, the majority of the UML metrics proposed are primarily simple counting metrics (e.g. number of use-cases in a model).

One of the earliest sets of metrics proposed for UML models are those described by Marchesi who propose metrics that can be applied to class and use

[1] `http://www.eclipse.org/emf/`
[2] `http://mdr.netbeans.org/`

case diagrams [15]. Genero *et al.* have proposed a set of metrics for assessing the structural complexity of class diagrams and have performed several experiments to empirically validate these metrics [16,17]. Various other metrics have been proposed for class diagrams and a comparison of these metrics can be found in [18]. Genero *et al.* have also developed a set of metrics for measuring the size and structural complexity of state-chart diagrams [19]. Kim and Boldyreff have defined a set of 27 metrics to measure various characteristics of a UML model [20]. However, the metrics are described informally and for some of these measures it is unclear which UML diagrams should be used to calculate the measures.

There has been relatively little work on measuring existing design metrics from *all* of the available views and diagrams of a UML model and there is as yet no convergence of opinion on the usefulness, or indeed the use, of these model level metrics.

3 Relationship with Code

The area of software metrics is reasonably well developed, and a discussion of model level metrics would be incomplete without considering what we can learn from existing lower level metrics. In particular, the relationship between models of a software system and the corresponding code can be explored through evaluation of similar metrics at each level of abstraction.

Observation 4. We can "lift" code metrics to the model level.

One of the most well known suites of object-oriented metrics is the one proposed by Chidamber and Kemerer (CK) [1]. These metrics were proposed to capture different aspects of an object-oriented design including complexity, coupling and cohesion. Several studies have been conducted to validate these metrics and have shown that they are useful quality indicators [21]. Baroni *et al.* have formalised the CK metrics using the OCL and the UML 1.3 metamodel [22]. We have also formalised the CK metrics using the OCL but have based our definitions on the UML 2.0 metamodel [12,13]. These definitions specify how to obtain the CK metrics from class diagrams but do not take any of the other UML diagrams into consideration. Tang and Chen have also attempted to specify how the CK metrics can be measured from UML diagrams [23]. They have developed an algorithm for computing the metrics from UML class, activity and communication diagrams.

The CK metrics suite consists of six metrics: Weighted Methods Per Class (WMC), Depth of Inheritance Tree (DIT), Number of Children (NOC), Coupling between Object Classes (CBO), Response For a Class (RFC), and Lack of Cohesion in Methods (LCOM). Each of the metrics refer to the individual class in the software system and not to the whole system. Figure 1 reviews each of these metrics and briefly discusses which UML diagrams need to be examined in order to gain accurate measures of the metrics.

In addition, it may be possible to obtain further information for the calculation of these metrics, e.g. method invocations and variable usages of methods and classes, by inspecting OCL constraints of the system. Interpreting such information requires further research.

Weighted methods per class (WMC): This metric is concerned with the complexity of the methods within a given class. It is equal to the sum of the complexities of each method defined in a class. If we consider the complexity of each method to be unity then the WMC metric for a class is equal to the number of methods defined within that class, we refer to this as WMC_1. The WMC_1 metric for a class can be obtained from the class diagrams of a UML model by identifying the class and counting the number of methods in that class. Alternatively, we can consider the complexity of each method to be McCabe's Cyclomatic complexity [2], which we refer to as WMC_{cc}. The activity, sequence and communication diagrams clearly contain information relevant to WMC_{cc}, but it is equally plausible that the state machine diagram could be used to compute this value for the class as a whole.

Number of children (NOC): This is the number of immediate descendants of a given class, that is the number of classes which directly inherit from the class. Again, this metric can be measured for a class by taking the union of all the class diagrams in a UML model and examining the inheritance relationships of the class.

Response for a class (RFC): This is a measure of the number of methods that can potentially be invoked by an object of a given class. The number of methods for a class can be obtained from a class diagram, but the number of methods of other classes that are invoked by each of the methods in the class requires information about the behaviour of the class. This information can be derived by inspecting the various behavioural diagrams, such as sequence and collaboration in order to identify method invocations.

Coupling between object classes (CBO): Two classes are coupled to each other if a method of one class uses an instance variable or method of the other class. An estimate for this metric can be obtained from the class diagrams by counting all the classes to which the class has a relationship with and counting all the reference types of the attributes and parameters of the methods of the class. To obtain a more precise value, information from the behavioural diagrams can be taken into account in order to get information about the usage of instance variable and invocation of methods. For example, a sequence diagram gives direct information about the interactions between methods in different classes.

Depth of inheritance tree (DIT): This is a measure of the depth of a class in the inheritance tree. It is equal to the maximum length from the class to the root of the inheritance tree. This metric can be computed for a class by taking the union of all the class diagrams in a UML model and traversing the inheritance hierarchy of the class.

Lack of cohesion in methods (LCOM): Calcuating the LCOM for a given class involves working out, for each possible pair of methods, whether the sets of instance variables accessed by each method have a non-empty intersection. In order, to compute a value for this metric, information on the usage of instance variables by the methods of a class is required. This information cannot be obtained from a class diagram. However, an upper bound for this metric can be computed using the number of methods in the class. Diagrams that contain information about variable usages, e.g. sequence diagrams can be used to compute this metric.

Fig. 1. An overview of applying the CK metrics to UML models. In this figure we review the diagrams in a UML model that can contribute to calculating the CK metrics.

Observation 5. Models can represent partial and/or overlapping information.

In the latest version of UML (2.0), there are 13 different basic diagrams that can be used to specify a software system. Existing object-oriented metric suites, such as the CK suite, are mainly relevant to class diagrams, since they measure structural elements of the design. Source code provides this information in the same format at the same level of abstraction but UML models can represent many different kinds of information. For example

- A single class may appear in a number of different class diagrams, with different degrees of elaboration of its attributes, methods and associations in each. This information needs to be merged in a consistent way before metric calculation.
- Some UML diagrams represent a view of a system, rather than a single overview. For examples, sequence diagrams are typically used to provide details of a usage scenario. It is not obvious how we should calculate metrics across such diagrams, and how we should merge the information from different diagrams with the same elements.

Defining how to integrate these different sources of information is a significant issue in model level metrics.

Observation 6. Differences between metric values are themselves metrics.

Ideally, following a Model Driven Architecture (MDA) approach to software development, the design models and the implementation are synchronised, so that changes in one are reflected in the other [24]. In practice, UML models can represent a design stage of a project, used perhaps once to develop a prototype implementation, and then not updated as the software develops. In this context, differences between the values of similar metrics measured at the model and source code level will reflect properties of the evolution of the system, rather than its design.

Even when models and implementation are synchronised, there will be a difference between metric values. For example, internal complexity measures for method bodies may not be available in the model, but can be calculated from the code. In this context, the model could be used to specify boundary values for the implementation, or the difference between metric values at the model and implementation level can capture the level of additional complexity added by the implementation process. For example, one might expect a prototype implementation to preserve many of the model level metric values, whereas the ultimate "real" implementation might introduce significant changes in the metric values.

Identifying differences between the values of the same metric applied to the same system could also have potential use in reverse engineering. It has already been noted that a significant level of variation exists between existing tools that reverse engineer class diagrams [25]. Software metrics, measured at the model level and then compared, can be used to evaluate the correctness of reverse engineering tools, or to quantify their perspective on the abstraction of high-level concepts, such as aggregation and composition, from the source code.

4 Some Future Directions

In this section we outline some directions for future research in the area of model level metrics that we regard as important.

Observation 7. Metric definitions should be re-usable.

Standard concepts measured by metrics, such as DIT or NOC, apply equally to models and code. Ideally, it should be possible to define these concepts once, and then adapt them to each relevant metamodel in turn. This provides not only for economy of expression, but also assurance that the same concepts are being measured at each level. However, this is not as easy as it may appear. Even a relatively simple metric, such as DIT, involves traversing relationships that may be represented quite differently in different models.

The simplest approach might be to define a single model over which the metrics are defined, and then apply transformations to map other models into this canonical model. However, given the range of UML diagrams, and possible contributions from language metamodels, a single canonical model may not be realistic. Instead, we may need to examine the possibility of mapping the metric definitions across different models.

Observation 8. The relationship between behavioural models and coverage needs to be explored.

A number of the UML diagrams represent behavioural aspects of a system, for example, use case, sequence and communication diagrams. Calculating metrics for such diagrams involves measuring a particular usage of the system, rather than its design as a whole. We have previously mentioned the difficulty of merging such partial information, but there are also unexplored issues regarding how such information should be interpreted.

Previous work, including our own, has explored some of the issues relating to defining and evaluating metrics at run-time [26,27]. Such metrics can be shown to capture additional information about the program but are, of course, dependent on the context in which the program is run. Indeed it is arguable that metrics at this level represent *coverage* data, rather than metrics in the usual sense. The use of such information, or its integration into testing strategies, is still relatively undeveloped.

Observation 9. Standardisation is multi-faceted; interoperability is the key.

One of the benefits of metamodelling is that interoperability between models is facilitated; metamodel Zoos [3] represent an important contribution here. However, there are other aspects that can contribute to comparing and evaluating metric results; some of these include:

- Benchmark suites
  The importance of benchmarks in software engineering in general, and in evaluating fact extractors in particular, has been noted by Sim *et al.* [28].

[3] For example, `http://www.eclipse.org/gmt/am3/zoos/`

They note the importance of benchmark suites, such as the SPEC suite, in other areas of computer science, and argue for a similar approach to software engineering research. Similarly, a call for benchmarks for software visualisation was issued in 2003 [29], but it is not clear what level of acceptance this has gained. The selection of a number of common programs and models for use in metric studies would greatly facilitate comparison between metrics and evaluation of new metrics.

- Data sets
  An interesting recent development towards standardisation and repeatability of results is the *Promise Software Engineering Repository* [30]. This is a collection of publicly available datasets "created to encourage repeatable, verifiable, refutable, and/or improvable predictive models of software engineering". At the moment the repository is still in the early stages of development and contains relatively specialised data sets, but it represents a promising trend in software engineering research.
- Non-code artifacts
  One of the difficulties in evaluating metrics at the UML level is the relatively small supply of UML and other design level artifacts. Open source software provides a rich source of information at the code level; it would be highly desirable if design level documents could be made available in a similar fashion. One initiative is the *Repository for Model Driven Development (ReMoDD)* project [31]. The objective of this project is to develop a repository of artifacts for use by researchers and industry practioners in the area of Model Driven Engineering of software systems. Also as an approximation, UML diagrams can be reverse engineered from code, and the reverse engineering community has already provided for interoperability through formalisms such as GXL [32] and our own *g4re* artifact repository [33]. However, reverse engineering artifacts are fundamentally different from design artifacts, and can at best only serve as an approximation for the real thing.

5 Summary

In this position paper we have discussed a number of issues relating to model metrics, with particular emphasis on metrics for UML models. We have structured our discussion around nine observations, which we can also partition into three levels of challenges for model metrics:

- The *technical challenge* of defining, comparing and reusing metrics over different descriptions of the same software system (Observations 1, 6, 7)
- The *conceptual challenge* of defining how to measure metrics from partial descriptions of models, and of the change in metrics between different representations of the software (Observations 3, 5, 8)
- The *practical challenge* of gathering, comparing and interpreting new and existing metrics (Observations 2, 4, 9)

Our own work in this area, as cited above, is concentrated on addressing the technical challenges of defining reusable metrics at the meta-level.

References

1. Chidamber, S., Kemerer, C.: A metrics suite for object oriented design. IEEE Transactions on Software Engineering **20**(6) (1994) 476–493
2. Fenton, N., Lawrence Pfleeger, S.: Software Metrics: A Rigorous and Practical Approach. International Thompson Computer Press (1996)
3. Briand, L., Daly, J., Wuest, J.: A unified framework for coupling measurement in object-oriented systems. IEEE Transactions on Software Engineering **25**(1) (1999) 91–121
4. Briand, L., Daly, J., Wuest, J.: A unified framework for cohesion measurement in object-oriented systems. Empirical Software Engineering **3**(1) (1998) 65–117
5. Wilkie, F., Harmer, T.: Tool support for measuring complexity in heterogeneous object-oriented software. In: IEEE International Conference on Software Maintenance, Montréal, Canada (October 3-6 2002)
6. Reißing, R.: Towards a model for object-oriented design measurement. In: ECOOP Workshop on Quantative Approaches in Object-Oriented Software Engineering, Budapest, Hungary (June 18-19 2001)
7. McQuillan, J., Power, J.: Experiences of using the Dagstuhl Middle Metamodel for defining software metrics. In: Proceedings of International Conference on Principles and Practices of Programming in Java, Manheim, Germany (August 30 - September 1 2006) 194–198
8. The Object Management Group: UML 2.0 draft superstructure specification (2003)
9. El-Wakil, M., El-Bastawisi, A., Riad, M., Fahmy, A.: A novel approach to formalize object-oriented design metrics. In: Evaluation and Assessment in Software Engineering, Keele, UK (April 11-12 2005)
10. Baroni, A., Brito e Abreu, F.: A formal library for aiding metrics extraction. In: ECOOP Workshop on Object-Oriented Re-Engineering, Darmstadt, Germany (July 21 2003)
11. Goulão, M., Brito e Abreu, F.: Formalizing metrics for COTS. In: ICSE Workshop on Models and Processes for the Evaluation of COTS Components, Edinburgh, Scotland (May 25 2004)
12. McQuillan, J., Power, J.: A definition of the Chidamber and Kemerer metrics suite for the Unified Modeling Language. Technical Report NUIM-CS-TR-2006-03, Dept. of Computer Science, NUI Maynooth, Co. Kildare, Ireland (October 2006)
13. McQuillan, J., Power, J.: Towards re-usable metric definitions at the meta-level. In: PhD Workshop of the 20th European Conference on Object-Oriented Programming, Nantes, France (July 4 2006)
14. Halstead, M.: Elements of Software Science. First edn. Elsevier, North Holland (1977)
15. Marchesi, M.: OOA metrics for the Unified Modeling Language. In: Second Euromicro Conference on Software Maintenance and Reengineering, Florence, Italy (March 8-11 1998)
16. Genero, M., Piattini, M., Calero, C.: Early measures for UML class diagrams. L'Object **6**(4) (2000) 489–515
17. Genero, M., Jimnez, L., Piattini, M.: A controlled experiment for validating class diagram structural complexity metrics. In: International Conference on Object-Oriented Information Systems, Montpellier, France (September 2-5 2002)
18. Yi, T., Wu, F., Gan, C.: A comparison of metrics for UML class diagrams. ACM SIGSOFT Software Engineering Notes **29**(5) (2005) 1–6

19. Genero, M., Miranda, D., Piattini, M.: Defining and validating metrics for UML statechart diagrams. In: 6th ECOOP Workshop on Quantitative Approaches in Object-oriented engineering, Malaga, Spain (June 11 2002)
20. Kim, H., Boldyreff, C.: Developing software metrics applicaple to UML models. In: 6th ECOOP Workshop on Quantitative Approaches in Object-oriented engineering, Malaga, Spain (June 11 2002)
21. Basili, V., Briand, L., Melo, W.: A validation of object-oriented design metrics as quality indicators. IEEE Transactions on Software Engineering **22**(10) (1996) 751–761,
22. Baroni, A., Brito e Abreu, F.: An OCL-based formalization of the MOOSE metric suite. In: Proceedings of ECOOP Workshop on Quantative Approaches in Object-Oriented Software Engineering, Darmstadt, Germany (July 22 2003)
23. Tang, M.H., Chen, M.H.: Measuring OO design metrics from UML. In: International Conference on The Unified Modeling Language, Dresden, Germany (September 30 - October 4 2002)
24. Warmer, J., Kleppe, A.: The Object Constraint Language. Addison-Wesley (2003)
25. Guéhéneuc, Y., Albin-Amiot, H.: Recovering binary class relationships: putting icing on the UML cake. In: Object Oriented Programming Systems Languages and Applications, Vancouver, BC, Canada (October 24-28 2004) 301–314
26. Arisholm, E., Briand, L., Foyen, A.: Dynamic coupling measures for object-oriented software. IEEE Transactions on Software Engineering **30**(8) (2004) 491–506
27. Mitchell, A., Power, J.: A study of the influence of coverage on the relationship between static and dynamic coupling metrics. Science of Computer Programming **59**(1-2) (January 2006) 4–25
28. Sim, S., Easterbrook, S., Holt, R.: Using benchmarking to advance research: A challenge to software engineering. In: International Conference on Software Engineering, Portland, Oregon, USA (May 3-10 2003) 74–83
29. Maletic, J., Marcus, A.: CFB: A call for benchmarks - for software visualization. In: 2nd IEEE Workshop of Visualizing Software for Understanding and Analysis, Amsterdam, The Netherlands (September 22 2003) 108–113
30. Shirabad, J.S., Menzies, T.J.: The PROMISE Repository of Software Engineering Databases. School of Information Technology and Engineering, University of Ottawa, Canada (2005)
31. Cheng, B., France, R., Bieman, J.: ReMoDD: A repository for model driven development
32. Holt, R., Schrr, A., Sim, S., Winter, A.: GXL: A graph-based standard exchange format for reengineering. Science of Computer Programming **60**(2) (2006) 149–170
33. Kraft, N., Malloy, B., Power, J.: Toward an infrastructure to support interoperability in reverse engineering. In: Working Conference on Reverse Engineering, Pittsburgh, PA (November 8-11 2005) 196–205