Towards the re-usability of software metric definitions at the meta level

- *Position Paper* -

Jacqueline A. McQuillan[*] and James F. Power

Department of Computer Science, National University of Ireland, Maynooth,
Co. Kildare, Ireland
{jmcq,jpower}@cs.nuim.ie

Abstract. A large number of metrics for evaluating the quality of software have been proposed in the literature. However, there is no standard terminology or formalism for defining metrics and consequently many of the metrics proposed have some ambiguity in their definitions. This hampers the empirical validation of these metrics. To address this problem, we generalise an existing approach to defining metrics that is based on the Object Constraint Language and the Unified Modelling Language metamodel. We have developed a prototype tool called DMML (Defining Metrics at the Meta Level) that supports this approach and we present details of this tool. To illustrate the approach, we present formal definitions for the Chidamber and Kemerer metrics suite.

1   Introduction

Software plays a pivotal role in many important aspects of modern daily life. In many cases, if software fails it can have catastrophic consequences such as economic damage or loss of human life. Therefore, it is important to be able to assess the quality of software. Software metrics have been proposed as a means of determining software quality. For example, studies have demonstrated a correlation between software metrics and quality attributes such as fault-proneness [1] and maintenance effort [2].

Many software metrics have been proposed in the literature [3–5]. In order for these metrics to be widely accepted, empirical studies of the use of these metrics as quality indicators are required. However, there is no standard terminology or formalism for defining software metrics and consequently many of the metrics proposed are incomplete, ambiguous and open to a variety of different interpretations [6]. For example, Churcher and Shepperd [7] have identified ambiguities in the suite of metrics proposed by Chidamber and Kemerer (CK) [3]. This makes it difficult for researchers to replicate experiments and compare existing experimental results and it hampers the empirical validation of these metrics.

---

[*] Corresponding author

Several authors have attempted to address the problem of imprecise metric definitions. Briand et al. propose two extensive frameworks for software measurement, one for measuring coupling [6] and the other for measuring cohesion [8] in object oriented systems. Other approaches include the proposal of formal models on which to base metric definitions [9] and the proposal of existing languages such as XQuery [10] and SQL [11] as metric definition languages. Baroni et al. propose the use of the Object Constraint Language (OCL) and the Unified Modelling Language (UML) metamodel as a mechanism for defining UML-based metrics [12, 13].

In this paper, we take the approach of Baroni et al. [13] and extend it in a number of ways. We decouple the metric definitions from the metamodel and thus develop a publicly available prototype tool which can be generalised to any metamodel and any set of metrics. Also, in this paper we are the first authors to provide the definitions of the CK metric suite using the OCL and the UML 2.0 metamodel.

The remainder of this paper is organised as follows. In section 2, a review of relevant research is presented. In section 3, we give details of an approach that allows for the unambiguos definition of software metrics. We have developed a prototype tool that supports this approach and present details of this tool in section 4. In section 5, we illustrate the application of the approach using the CK metrics suite. Section 6 gives a summary and discussion of future work.

2   Related Work

Several attempts have been made to address the problem of ambiguous metric definitions. Briand et al. propose an integrated measurement framework for the definition, evaluation and comparison of object oriented coupling metrics [6]. They have also developed a similar framework for cohesion [8]. These frameworks are specific to coupling and cohesion metrics and new frameworks must be developed to apply the approach to other types of metrics.

Harmer and Wilkie have developed an extensible metrics analyser tool for object oriented (OO) programming languages [11]. The tool is based on a general OO programming language metamodel in the form of a relational database schema. Metric definitions are expressed as SQL queries over this schema. The tool is extensible as it has support for incorporating new metrics and new OO programming languages. However, defining the metrics requires the additional effort of the development of C code as well as supplying the SQL queries. In addition, the tool is tied to the underlying metamodel and does not allow the interchange of metamodels.

Another approach put forward by Reißing involved the proposal of a formal model on which to base metric definitions [9]. This model is called ODEM (Object-oriented DEsign Model) and consists of an abstraction layer built upon the UML metamodel. However, this model can only be used for the definition of design metrics and does not solve the ambiguity problem as the abstraction layer consists of natural language expressions.

2

El-Wakil et al. propose the use of XQuery as a metric definition language [10]. They propose extracting metric data from XMI design documents, specifically UML designs. XQuery is a language that can be used to extract information from XMI documents. Again this approach has only been used to define metrics at the design level, specifically for UML designs. There is no information available on how it extends to other languages.

Baroni et al. propose the use of the OCL and the UML metamodel as a mechanism for defining UML-based metrics [12]. They have built a library called FLAME (Formal Library for Aiding Metrics Extraction) [14] which is a library of metric definitions formulated as OCL expressions over the UML 1.3 metamodel [15]. Goulão et al [16] have utilised this approach for defining component based metrics and used the UML 2.0 metamodel [17] as a basis for their definitions. We believe that this approach provides a useful mechanism for the precise definition of software metrics and we build upon it in this paper.

3   Software Metrics at the Meta Level

As we are examining the use of metamodels and the OCL as a basis for the definition of software metrics, we will begin by presenting a short explanation of these concepts.
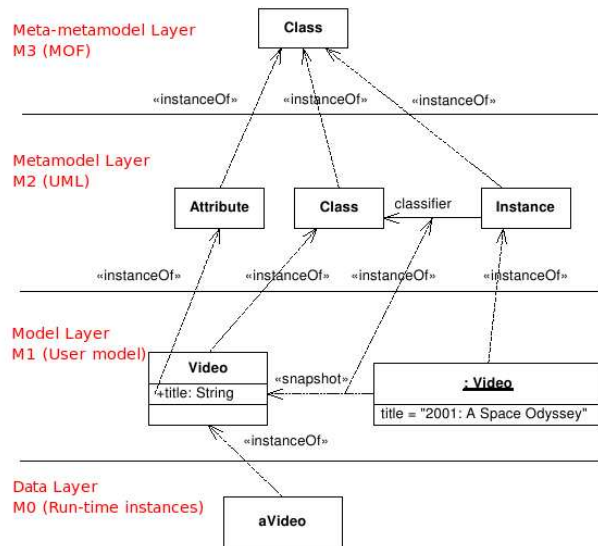

Fig. 1. The Four Layer Metamodel Architecture. *This diagram shows the standard four-layer hierarchy depicting the relationships between various levels of modelling and is taken from [18].*

3

3.1 Metamodels

A metamodel is a model of a model describing a software system [19]. It specifies the elements that may be used in a model of a software system and the relationships between these elements. Thus, it defines the language used to model a software system. Metamodels are important as they provide a way to unambiguously define languages.

When dealing with metamodels to define languages there are generally four levels or layers to be taken into account. These four layers are depicted in Fig. 1. At the data layer or $M0$ layer the entities are run-time instances of model elements. At the model layer also referred to as the $M1$ layer, the entities are models describing a software system. In Fig. 1 the model at layer $M1$ is a user model specified in UML. The metamodel or $M2$ layer is the next abstraction layer and it defines the language that can be used to describe the entities at the model layer. In Fig. 1 the UML metamodel is found at this layer. Finally, there is the meta-metamodel or $M3$ layer defining the language for the metamodel. We do not make further use of this layer here.

3.2 The Object Constraint Language

The Object Constraint Language (OCL) is a standard language that allows constraints and queries over object oriented models to be written in a clear and unambiguous manner [20]. It offers the ability to navigate over instances of object oriented models, allowing for the collection of information about the navigated model.

3.3 Extensions to the approach of Baroni et al.

Baroni et al. propose expressing design metrics as OCL queries over the UML 1.3 metamodel [15]. This approach involves modifying the metamodel by creating the metrics as additional operations in the metamodel and expressing them as OCL conditions [12]. We extend this approach in a number of ways. We decouple
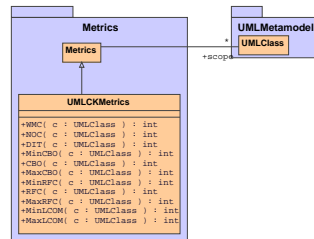

Fig. 2. Extension to the UML 2.0 metamodel. *This UML package diagram shows the definition of the CK metrics as a separate package, with a dependency on classes from the UML metamodel.*

4

the metric definitions from the metamodel by extending the metamodel with a seperate metrics package (see Fig. 2). This metrics package contains a single class, `Metrics`. To define a metric set a class is created that extends the `Metrics` class. For each metric, an operation in the class is declared, parameterised by the appropriate elements from the metamodel. The metrics are defined by expressing them as OCL queries using the OCL `body` expression. This approach has allowed us to develop an easily extendible tool that can, in theory, be applied to any language. Furthermore, these extensions will allow for interoperability between software metrics and will allow metric definitions to be easily re-used.

4 The DMML Tool

To facilitate the use of our approach to defining metrics at the meta level, we have developed a a prototype tool called DMML (Defining Metrics at the Meta Level). Our tool is implemented as a plug-in for the integrated development environment Eclipse [21] and operates in two stages: the *metric definition* stage and the *metric calculation* stage. An overview of these two stages is depicted in Fig. 3.
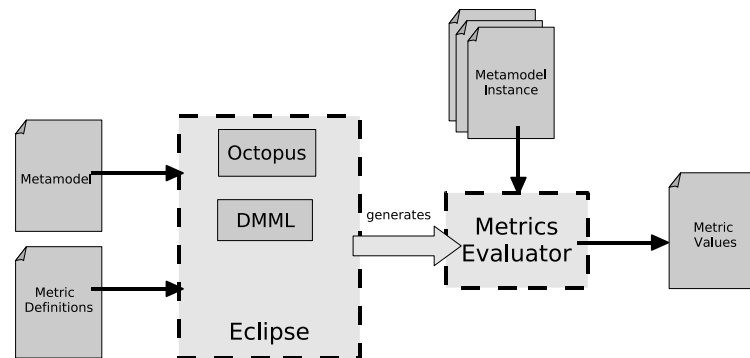

Fig. 3. DMML - An environment for the definition and calculation of software metrics. *This system overview diagram shows the main inputs to and outputs from the DMML tool, which is implemented as an Eclipse plug-in.*

At the *metric definition* stage a user can create and define a set of metrics specific to a software modelling (e.g. UML) or programming language (e.g. Java). To perform this task the user first loads the metamodel of the language and then specifies the names of the metric sets and metrics. Using this information, DMML creates an extension (i.e. a metrics package) to the metamodel. The user then creates or loads a file containing the metric definitions, expressed as queries over the language metamodel. DMML uses the Octopus [22] plug-in to syntactically and semantically check these OCL expressions.

5

At the *metric calculation* stage DMML takes the loaded metamodel and its metrics extension and uses Octopus to generate the corresponding Java classes. A metrics evaluator is created as a Rich Client Platform (RCP) to calculate the defined metrics for any instance of the loaded metamodel. When this metrics evaluator runs it displays all the metrics that have been defined during the *metric definition* stage. To calculate the metrics, an instance of the metamodel is loaded and is used to instantiate the metamodel classes. If the instance of the metamodel is created successfully, the methods corresponding to the metric definitions are called and return the metric results. These results are displayed in the RCP application window and exported in text format. Future versions will provide options to export the results in text, XML and HTML format.

An outline of how DMML works for the UML 2.0 metamodel can be seen in Fig. 4. The language metamodel here describes the domain over which the metrics are to be applied. To extend DMML to work with other language meta-models the user only needs to add the functionality to convert instances of the metamodel to the format understood by DMML. In Fig. 4 we represent this as an XSLT transformation from XMI to XML, though the user is free to use any XML-generating tool. We emphasise that the DMML tool is parameterised by both the language metamodel and the definition of the metrics set.
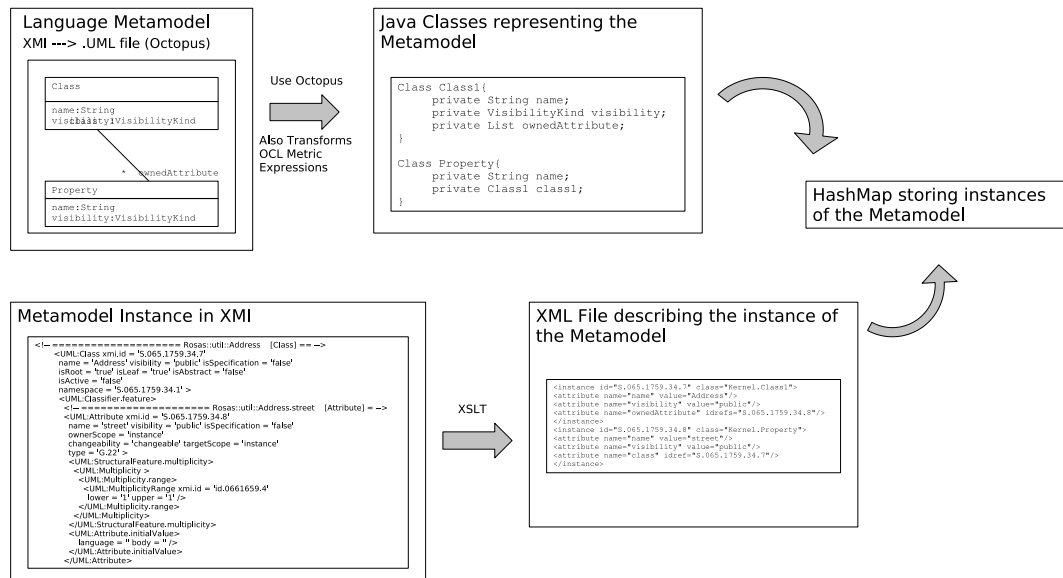

Fig. 4. An overview of using DMML with the UML 2.0 Metamodel. *This diagram gives a snapshot of some of the data involved in using the DMML tool.*

6

5 The Chidamber and Kemerer Metrics Suite

In this section we use the Chidamber and Kemerer (CK) [3] metrics suite to illustrate the use of the approach outlined in this paper. We express the CK metrics as OCL queries over the part of the UML 2.0 metamodel [17] that defines class diagrams. We are the first authors to provide such definitions using the UML 2.0 metamodel.

The UML 2.0 metamodel is a model that is used to define the UML. It specifies the constructs that may be used in a UML model and the relationships between these constructs. For example, the part of the UML metamodel that is specific to class diagrams defines the concepts of class, attribute, operation and states that a class includes attributes and operations. The structure of a UML model always conforms to the UML metamodel.

The metrics suite proposed by Chidamber and Kemerer is one of the most well known suite of OO metrics. The suite consists of the following 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)
– Lack of cohesion in methods (LCOM)

The CK metrics were proposed to capture different aspects of an OO design. However, not all of the CK metrics can be precisely measured from a UML class diagram. Implementation details, such as the code in the bodies of method definitions, are required to measure the CBO, RFC and LCOM metrics. However, we were able to provide definitions to estimate the values for these metrics based on the information in the UML diagrams. Such measures are useful as they can provide upper and lower bounds for metrics calculated at later stages in the design or implementation process.

With reference to Fig. 4, the *Language Metamodel* we used is the standard metamodel for UML class diagrams [17]. A *metamodel instance* in this case is an actual class diagram, represented in XMI, the standard output format for most UML modelling tools. The DMML tool then reads a transformed version of the class diagram, expressed in XML, and uses this to instantiate the generated Java classes that are used to represent the elements of class diagrams.

As an example of the format of the CK metric definitions, Fig. 5 illustrates how the NOC metric can be expressed as an OCL query over the UML 2.0 metamodel. Here, the definition is parameterised by a single `UMLClass`, and the body of the definition returns the size of the set of all children of this class. The auxiliary operation `children` traverses the elements and relationships in the UML metamodel to assemble this set. Full details of this and other metric definitions can be found in [23].

As a proof of concept, DMML has been used to calculate these metrics for an open source project, *Velocity* which is part of the Apache Jakarta project [24].

7

```
-- Returns a count of all the immediate descendants of the Class c
context umlckmetricset::NOC(c:UMLMetamodel::Kernel::UMLClass):Integer
  body: self.children(c)->size()

--Returns the set of all children of the Class c
context umlckmetricset
  def: children(c:UMLMetamodel::Kernel::UMLClass)
    : Set(UMLMetamodel::Kernel::UMLClass)
    = self.scope->select(i:Kernel::UMLClass| i.parents()->includes(c))->asSet()
```

Fig. 5. NOC Metric Definition. *This OCL code defines the NOC metrics from the CK metrics set, and is part of a larger definition of the whole CK metric set which we have implemented as part of DMML.*

We chose to use version 1.2 of *Velocity* as this is the version used in the study by Briand et al. [25]. We reverse engineered the system using Rational Rose to obtain a UML model. Using our DMML tool we calculated the CK metrics suite for the resulting UML model. Fig. 6 shows the results from the metric calculations.

Fig. 6. Metric results for classes from the Apache Jakarta *Velocity* project. *This screenshot of the DMML tool shows some of the results of evaluating the CK metrics set over the classes from the* Velocity *project [24].*

8

6   Summary and Future Work

In this paper, we have identified the need for a clear, unambiguous framework for defining metrics. This framework should provide for the comparison on different definitions of the same metrics, and for using a metric, or suite of metrics in different environments. To achieve this we exploit OCL as a specification language, and harness the UML metamodel to provide a framework for metric definitions. We have implemented a tool, DMML, as an initial demonstration of the feasibility of our approach. A final contribution of this work is that it provides a first ever definition of the Chidamber and Kemerer metrics suite using the UML 2.0 metamodel as a basis for these definitions.

While our approach to date is similar to other research in this area, particularly that of Baroni et al., it differs in a number of key areas. Our approach decouples the metrics from the underlying metamodel. While this does not provide any immediate benefit for the specification of metrics over UML class diagrams, it is key to providing a foundation for our future work. First, our approach can be generalised at the metamodel level, for example, to apply to other UML diagrams. Second, the metric definitions and their calculation procedure is highly extensible, allowing for different versions to be implemented and compared. The DMML tool is readily extendible and applicable to any metamodel and any set of metrics.

We plan to build on this foundation by developing our research in three main directions:

- We plan to extend metric definitions to other UML diagrams. While this will allow us to add breadth to our metric set, it will also be important in ensuring consistency across design documents for a single application, and in tracking the impact of design decisions from different diagrams on the application as a whole.
- We will extend the metrics to the implementation level, using programming language metamodels. This will provide a single, coherent framework within which the design and implementation process can be measured. This will provide a clear, quantitative measure of the changes that take place between design and implementation.
- We will investigate the variances between different definitions of the same metrics over both design and implementation artifacts.

We have already tested the feasibility of our approach on the Jakarta *Velocity* tool. We intend to analyse a suite of open-source software as part of our work, in order to ensure the robustness and generalisability of our results.

References

1. Basili, V., Briand, L., Melo, W.L.: A validation of object-oriented design metrics as quality indicators. IEEE Transactions on Software Engineering **22** (1996) 751–761
2. Li, W., Henry, S.: Object-oriented metrics that predict maintainability. Journal of Systems and Software **23** (1993) 111–122

3. Chidamber, S., Kemerer, C.: A metrics suite for object oriented design. IEEE Transactions on Software Engineering **20** (1994) 476–493
4. Lorenz, M., Kidd, J.: Object-Oriented Software Metrics. Prentice Hall Object-Oriented Series (1994)
5. Fenton, N., Lawrence Pfleeger, S.: Software Metrics: A Rigorous and Practical Approach. International Thompson Computer Press (1996)
6. Briand, L.C., Daly, J.W., Wuest, J.K.: A unified framework for coupling measurement in object-oriented systems. IEEE Transactions on Software Engineering **25** (1999) 91–121
7. Churcher, N., Shepperd, M.: Comments on 'A metrics suite for object-oriented design'. IEEE Transactions on Software Engineering **21** (1995) 263–265
8. Briand, L.C., Daly, J.W., Wuest, J.K.: A unified framework for cohesion measurement in object-oriented systems. Empirical Software Engineering **3** (1998) 65–117
9. Reißing, R.: Towards a model for object-oriented design measurement. In: Proceedings of ECOOP Workshop on Quantative Approaches in Object-Oriented Software Engineering, Budapest, Hungary (2001)
10. El-Wakil, M., El-Bastawisi, A., Riad, M., Fahmy, A.: A novel approach to formalize object-oriented design metrics. In: Proceedings of Evaluation and Assessment in Software Engineering, Keele, UK (2005)
11. Wilkie, F., Harmer, T.: Tool support for measuring complexity in heterogeneous object-oriented software. In: Proceedings of IEEE International Conference on Software Maintenance, Montréal, Canada (2002)
12. Baroni, A.: Formal definition of object-oriented design metrics. Master's thesis, Vrije Universiteit Brussel - Belgium, in collaboration with Ecole des Mines de Nantes - France and Universidade Nova de Lisboa - Portugal (2002)
13. Baroni, A., Braz, S., Brito e Abreu, F.: Using OCL to formalize object-oriented design metrics definitions. In: Proceedings of ECOOP Workshop on Quantative Approaches in Object-Oriented Software Engineering, Malaga, Spain (2002)
14. Baroni, A., Brito e Abreu, F.: A formal library for aiding metrics extraction. In: Proceedings of ECOOP Workshop on Object-Oriented Re-Engineering, Darmstadt, Germany (2003)
15. The Object Management Group: UML 1.3 specification (1999)
16. Goulão, M., Brito e Abreu, F.: Formalizing metrics for COTS. In: Proceddings of the ICSE Workshop on Models and Processes for the Evaluation of COTS Components, Edinburgh, Scotland (2004)
17. The Object Management Group: UML 2.0 draft superstructure specification (2003)
18. The Object Management Group: UML 2.0 draft infrastructure specification (2003)
19. Warmer, J., Kleppe, A., Bast, W.: MDA Explained: The Model Driven ArchitecturePractice and Promise. Addison-Wesley (2003)
20. Warmer, J., Kleppe, A.: The Object Constraint Language. Addison-Wesley (2003)
21. Eclipse Open Source Community: Eclipse. http://www.eclipse.org/ (2006)
22. Klasse Objecten: Octopus: OCL tool for precise UML specifications. Available from http://www.klasse.nl/octopus/ (2006)
23. McQuillan, J., Power, J.: A definition of the Chidamber and Kemerer metrics suite for the Unified Modeling Language. Technical Report NUIM-CS-TR-2006-04, Department of Computer Science, NUI Maynooth, Co. Kildare, Ireland (2006)
24. Jakarta: The Apache Jakarta Project. http://jakarta.apache.org/ (2003)
25. Arisholm, E., Briand, L., Fyen, A.: Dynamic coupling measurement for object-oriented software. Technical Report 2003-05, Simula Research Laboratory, Norway (2003)