In: Proceedings of the 2nd Workshop on Model Size Metrics (MSM'07) at MoDELS'2007, 2007

Counts count

M. Monperrus, J. Champeau, B. Hoeltzener

ENSIETA
2, rue F. Verny
F-29806 Brest cedex 9
Contact: martin.monperrus@ensieta.fr

Abstract. Based on the 2006 edition of the Model Size Metrics workshop, we believe that counts are undervalued as useful model metrics. In this position paper, we provide arguments from the literature so as to consider counts as important metrics for the model measurement. We then state associated issues and sketch a model-driven framework to raise the abstraction level of the implementation of model metrics, starting with count metrics.

1 Introduction

According to historians [1], writing was invented by the end of the 4th millennium BC to record counts, for instance the number of animals in a herd. Indeed, the first quantitative knowledge of any system is a count.

However, in regard to the previous edition of this workshop[1], it seems that model measurement by means of counts is undervalued . The following quotations are given so as to illustrate this statement. McQuillan et al. [2] state that *the majority of the UML metrics proposed are primarily simple counting metrics*. Weil et al [3] *do not believe, however, that simply counting the element in a model to produce a single number provides a meaningful result*. And Stoerrle [4] considers that *the simplest and most general approach is to always count everything, and instrument the count by an external weight table*. The substantive "simple" is used three times: simple, simply, simplest without any counterbalancing adjectives.

In fact, we agree with the cited authors, a count itself is really simple. But replaced in a broader context, we believe that counts i.e., *number of* metrics, are more important than they first seem to be.

To set the context of our reflexion on model metrics, we consider the Goal-Question-Metric (GQM) approach [5]. The GQM approach helps to specify a measurement system targeting a particular set of issues. Since this position paper does not introduce a particular metric, one cannot use directly the GQM approach, but a kind of GQM at a higher level of abstraction. In fact, we aim to show that count metrics are relevant answers to several questions linked to a primary goal that is measuring models. To sum up this point, we now consider the standard GQM notation:

[1] Model Size Metrics workshop at Models'2006, Genova, october 2006

Goal Improve the quantitative knowledge on models from the modeler point of view.
Question #1 What model metrics can be related to development cost?
Question #2 What model metrics can be related to psychological complexity?
Question #3 What model metrics can be related to product defects?
Metrics The family of count metrics.

The paper is built as follows. The premises of our reflexion are elements of the software metrics literature. These elements are grouped into propositions that are: the software measurement literature is rich of useful count metrics; the Occam's razor principle defends counts; industrial measurement plan may include counts; complex metrics may be interestingly grounded by counts; counting is a fully-fledged modeling goal. These propositions support our initial claim: *Count counts*. We then discuss the consequences and present a model-driven framework as a preliminary answer.

2 Main arguments

2.1 Count metrics are important in software engineering

Metrics in model-driven engineering (MDE) can be compared with their software metric ancesters. Existing software metrics are defined as counts or are count-based. In this perspective, the undervaluation of counts in MDE described above constrats with the rich literature on counts in the software measurement domain.

For instance, whatever the definition (see [6] for several variations), the lines of code metric is a count. Empirical studies based on historical data have shown that the line of code metric is relevant to the correlation between the software product and the cost [7], to evaluate an attribute of the software quality (e.g. the defect rate metric), or to study the team productivity [8].

Another example is the structural complexity SC [9] (see figure 1). The structural complexity is part of the psychological complexity of software [10]. It impacts the time to understand and modify a system accordingly. The definition of SC [9], exposed in equation 1, involves the number of edges $|A|$ and the number of modules $|M|$.

$$SC = \frac{|A|}{|M|} \tag{1}$$

The goal of this paragraph is not to list all software metrics which are counts but to emphasize that known and useful software metrics are counts or count-based. This is a clue not to discard counts in the upcoming generation of accepted model metrics.

2.2 The application of the law of parsimony to metrics

> *Using more sophisticated measures do not necessarily guarantee better results.* Zuse [9, 410]

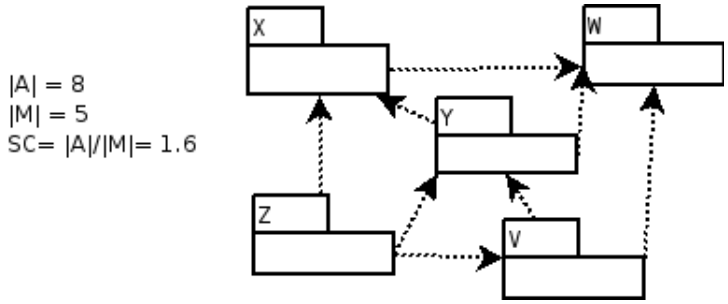

Fig. 1. The structural complexity [9] is based on the counts of dependency edges and modules

The Occam's razor, also known as the law of parsimony [11], is a principle most commonly stated as follows: *Entities are not to be multiplied beyond necessity.* It is a polymorphic principle and one of its instance is the *Keep It Short & Simple* (or K.I.S.S) engineering rule. This principle applied to model metrics is a strong argument in favor of counts compared to more complex metrics. Counts are very simple metrics, but may give model users the appropriate information.

Considering that the law of parsimony has been proven as a powerful principle in both science and philosophy (e.g. [11]), counts should be considered with a positive a priori judgement.

2.3 Counts ground more complex metrics

Counts are simple but can ground more complex metrics. For instance, in [12] (cited in [13]) a linear relationship is found with a high statistical confidence between defects and some language constructs in a given technical context (# denotes the *number of*):

$$Field\ defects = 0.11\#ifthen + 0.03\#calls \tag{2}$$

In this case, a defect measure is derived from some counts.

In a similar manner, the Cocomo model [7] outputs a cost with several inputs. Inputs are cost drivers and a size parameter. In a similar technical and management context i.e. for similar cost drivers, the main parameter is a count of lines of code.

Since counts are able to ground more complex metrics, they must be carefully considered. The first way to build more complex metrics is to define a weighted sum of counts (as in equation 2 and emphasized in [4] and [3]). It is not the unique solution, non linear combinations in a bigger space of functions can be explored.

2.4 Counts are part of industrial development processes

Counts are valuable enough to be adopted in an industrial context. Cheng et al. [14] examine the automated metrics-based analysis and detection of design

Error Description	Model A	Model B
SEVERE ERRORS		
Abstract class not inherited	8	82
Circular association	0	8
Circular dependency	102	0
Abstract class inherits from concrete class	6	34
Class inherits from one or more non-base classes	0	5
Interface to class expected but defined improperly	1	53
Two methods exist in the model with the same signature	1	0
Two objects exist in the model with the same name	5	23
Parent accessing attributes/operations of child class	0	2

Table 1. Counts of violations of a given pattern from [14]

guideline violations in the Siemens context. In fact, the metrics explored in this paper are counts of violations of a given pattern (see table 1) and ground the whole analysis. In another company, Motorola [3], a list of around 150 counts of UML elements is used to control the model-driven development process.

2.5 Counting is a fully-fledged modeling goal

Modeling goals are multiple [15]: execution, test generation, transformation, etc. In domains where textual artifacts are historically done, modeling is a way to isolate and count concepts. For instance, expressing software and system requirements with models allows a precise count of explicited elements [16]. These counts are difficult to retrieve with textual requirements with a similar validity and reliability. In such cases, counting is part of the primary modeling goals.

3 Consequences

This section discusses the consequences and challenges raised by the fact that *counts count*.

3.1 What artefacts in a model need to be counted?

The first challenge is to determine what needs to be counted in a model. A goal, a metamodel, and a corpus of models set the context of an empirical answer. Effectively, according to the GQM approach, a goal precises the question. A model space is identified with a metamodel or a part of a metamodel. For instance, what are the elements of UML2 models to be counted and given as inputs of a cost estimation model? Furthermore, since the quality of a metric resides on its empirical validation [17], a validation must be done on a statistically big enough corpus of data, with as less as possible of uncontrolled variables.

Unfortunately, our community does not yet have such a corpus of data. By analogy with the non model-based database, maintained by the ISBSG [18], a

database of count metrics of models and their associated dependent variables would be of great use to select count metrics and validate count based derived metrics. It is to be noted that the confidentiality of industry data obstruct the making of database. Contrary to whole models, count metrics have the advantage to be almost completely free of business information, yet are a kind of signature of models.

3.2 How are these artefacts counted?

As stressed in [19] the natural language is not enough to precisely define metrics. For instance, in the figure 1, do signatures include parameters name? To that extent, an issue is to rigorously define what to count.

We also consider that the implementation of count metrics should satisfy the following requirements:

1. The model measurement tool is accessible via a command-line interface.
2. The model measurement tool integrates into a modeling environment.
3. The code includes an API to implement estimation models on top of count metrics.
4. The model measurement tool let the modeler to tailor implemented count metrics.
5. The model measurement tool let the end-user to define its own business relevant metrics.
6. The model measurement tool is independent of a specific persistence format.

Marinescu et al. [20] underline the issue of the implementation complexity of object-oriented metrics. By analogy, and regarding to the above requirements, we assume that the implementation of model metrics, including count metrics, introduces a similar complexity. Consequently, it raises the development and maintenance cost of a modeling environment and may hamper the code quality of the whole tool. In section 3.3 below, this issue is adressed with an abstraction level to define model metrics from which implementation and integration into a modeling tool are automatically derived.

3.3 Model-driven implementation of count metrics for DSMLs

Motivation On the one hand, counts count. On the other hand, the implementation of count and count-based metrics can be costly. The development of a domain-specific modeling language (DSML) is usually done with tight time-to-market and cost constraints [21]. Thus, several frameworks and techniques try to generate as much as possible of the DSML modeling environment and associated tools (eg. [22]). Our motivation is to leverage code generation for the DSML metric implementation, starting with counts, to eventually give an enhanced editor with metric capabilities.

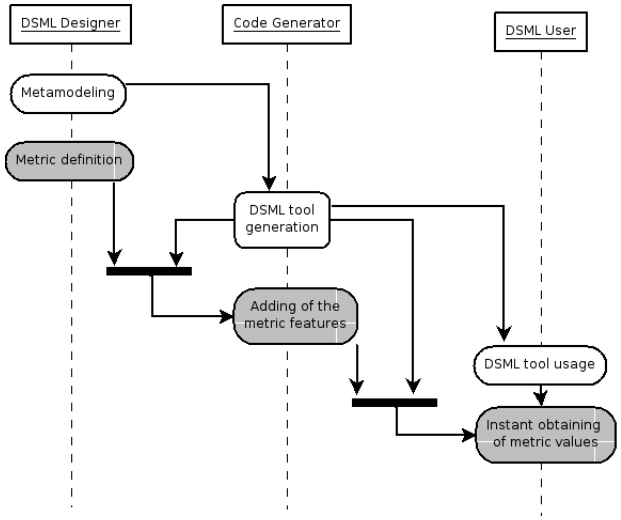

Fig. 2. Model-driven engineering of counts

Sketch In figure 2, we sketch the whole process as an UML activity diagram. The framework is involved in the three activities highlighted in gray.

The design of a DSML environment starts with the metamodeling activity. The metamodel identifies the concepts of the domain and their relationships. Tools are implemented to create, edit and browse models as well as to check the conformity to the metamodel. Frameworks exist to generate such tools from a metamodel (e.g. EMF/GMF[2]).

Then, the DSML designer defines the metrics at a higher level of abstraction than code. The family of count metrics defines an abstraction level. Each metric of this family is specified by a predicate [**?**]. The value of the metric is the number of elements that satisfy the predicate. For sake of brevity, we only make a short presentation of predicates. A predicate is a function that takes an object as input and makes tests on it to determine if it has to be counted. Predicates refer to the concepts expressed in the metamodel. The tests are made on type, attribute values and/or referenced objects. A predicate can be as long and complex as needed. A predicate can be expressed in a metric specific language or in an existing query language (e.g. OCL). Two examples of predicates in an OCL-like language are shown in figure 3.

A toolchain generates the DSML modeling environment, then compiles the declarative description of metrics directly in the modeling tool to augment it with counting capabilities (the two central activities of the figure 2).

Finally, the DSML user sees its model changes instantly reflected in the metric values. This approach has three major advantages:

[2] Eclipse/Graphical Modeling Framework, see www.eclipse.org/emf

```
-- metric #1: number of leaf abstract classes
self.isTypeOf(Class) and self.abstract = true and self.children->size() = 0

-- metric #2: number of too big use cases
self.isTypeOf(UseCase) and self.includedUsesCases->size() > 10
```

Fig. 3. Examples of predicates expressed in OCL

- metrics are defined at a higher level of abstraction without caring about how this can be implemented in a DSML modeling environment;
- the implementation of the enhanced editor is generated;
- DSML users can edit and measure models in the same environment.

Implementation We are developing a prototype of this framework where the metamodel is expressed in Ecore[3], and the predicates in Kermeta[4], which has good query capabilities. EMF/GMF generate editors in Java and the prototype compiles the predicates inside the editor Java code.

4 Conclusion

We presented a set of arguments in order to rehabilitate counts as first class citizens for the model measurement:

- software measurement history involves important count metrics;
- the law of parsimony prefers simple metrics;
- counts ground more complex metrics;
- counts are part of industrial measurement processes;
- counting is a fully-fledged modeling goal.

We hope that these arguments can provide a fertile basis for the discussion during the upcoming workshop. We then discussed the consequences and sketched a model-driven framework for count metrics. The contribution of this approach is to give an instant, reliable and low cost implementation of count metrics seamlessly integrated into modeling tools.

Acknowledgments The authors would like to thank B.Baudry and the anonymous reviewers for helpful comments. This work is supported by a DGA grant.

References

1. J. Goody, *The Domestication of the Savage Mind.* Cambridge University Press, 1977.

[3] XMI format of EMF

[4] a model-driven language and workbench, see www.kermeta.org and [23]

2. J. McQuillan and J. Power, "Some observations on the application of software metrics to UML models," in *Proceedings of the Model Size Metrics workshop at Models'2006*, 2006.
3. F. Weil, A. Neczwid, and K. Farbelow, "Model size metrics research in motorola," in *Proceedings of the Model Size Metrics workshop at Models'2006*, 2006.
4. H. Störrle, "On different notions of model size," in *Proceedings of the Model Size Metrics workshop at Models'2006*, 2006.
5. V. R. Basili, G. Caldiera, and H. D. Rombach, "The goal question metric approach," in *Encyclopedia of Software Engineering*, Wiley, 1994.
6. C. Jones, *Programming Productivity.* McGraw-Hill, 1986.
7. B. Boehm, B. Clark, E. Horowitz, R. Shelby, and C. Westland, "An Overview of the COCOMO 2.0 Software Cost Model," in *Software Technology Conference*, Apr. 1995.
8. C. Jones, *Software Productivity and Quality Today: The Worldwide Perspective.* Carlsbad, Calif.: IS Management Group, 1993.
9. H. Zuse, *Software Complexity.* Berlin: Walter de Gruyter, 1991.
10. B. Henderson-Sellers, *Object-Oriented Metrics, measures of complexity.* Prentice Hall, 1996.
11. A. Baker, "Simplicity," in *Stanford Encyclopedia of Philosophy*, The Metaphysics Research Lab, Stanford University, 2004.
12. B. Lo, "Syntactical construct based apar projection," tech. rep., IBM Santa Teresa Laboratory Technical Report, California, 1992.
13. S. H. Kan, *Metrics and Models in Software Quality Engineering.* Reading, MA: Addison Wesley, 1995.
14. B. H. C. Cheng, R. Stephenson, and B. Berenbach, "Lessons learned from automated analysis of industrial uml class models (an experience report).," in *Proceedings of MODELS' 2005*, pp. 324–338, 2005.
15. D. C. Schmidt, "Model-driven engineering," *IEEE Computer*, vol. 39, pp. 25–31, February 2006.
16. R. J. Costello and D.-B. Liu, "Metrics for requirements engineering," *J. Syst. Softw.*, vol. 29, pp. 39–63, Apr. 1995.
17. V. R. Basili and D. M. Weiss, "A methodology for collecting valid software engineering data.," *IEEE Trans. Software Eng.*, vol. 10, no. 6, pp. 728–738, 1984.
18. The International Software Benchmarking Standards Group, "ISBSG repositories," 2007.
19. A. Baroni, S. Braz, and F. Abreu, "Using OCL to formalize object-oriented design metrics definitions," in *ECOOP'02 Workshop on Quantitative Approaches in OO Software Engineering*, 2002.
20. C. Marinescu, R. Marinescu, and T. Gîrba, "Towards a simplified implementation of object-oriented design metrics.," in *IEEE METRICS*, p. 11, 2005.
21. J. White and D. Schmidt, "Simplifying the development of product-line customization tools via model driven development," in *MODELS'2005 workshop on MDD for Software Product-lines: Fact or Fiction?*, 2005.
22. A. Lédeczi, A. Bakay, M. Maroti, P. Vőlgyesi, G. Nordstrom, J. Sprinkle, and G. Karsai, "Composing domain-specific design environments," *IEEE Computer*, vol. 34, pp. 44–51, Nov. 2001.
23. P. A. Muller, F. Fleurey, and J. M. Jézéquel, "Weaving executability into object-oriented meta-languages," in *Proceedings of MODELS/UML 2005*, 2005.