Metrics in Method Engineering

Matti Rossi[1] and Sjaak Brinkkemper[2]

[1] University of Jyväskylä
Department of Computer Science and Information Systems
P.O. Box 35
SF-40351 Jyväskylä
Finland
Internet: mor@jyu.fi

[2] University of Twente
Centre for Telematics and Information Technology
P.O. Box 217
NL-7500 AE Enschede
Netherlands
Internet: sjbr@cs.utwente.nl

Abstract. So many software development methods have been introduced in the last decade, that one can talk about a "methodology jungle". To aid the method developers and evaluators in fighting their way through this jungle we propose a systematic approach for measuring properties of methods. We describe two sets of metrics, which measure the complexity of diagrammatic specification techniques on the one hand, and of complete systems development methods on the other hand. Proposed metrics provide a relatively fast and simple way to analyse the technique (or method) properties, and when accompanied with other selection criteria, can be used for estimating the cost of learning the technique and the relative complexity of a technique compared to others. To demonstrate the applicability of the metrics, we have applied them to 34 techniques and 15 methods.

1. Introduction

Recent years have witnessed the appearance of new software development paradigms and methods. Examples of these are object-oriented analysis and design methods, and business process reengineering methods. However, we feel that there is room for improvement in the analysis of these methods and in understanding their use and functionality. Though, some attempts have been made to compare existing methods (e.g. see [Cha91]), the studies lack rigor and sound conceptual foundation, and they are mostly based on ad hoc feature analysis techniques. Some recent attempts [Son92, Hon93] have proposed more systematic approaches, based on a common formal metamodel of method, that hold the promise of a more systematic and analytic way to compare methods. Yet, they are still mainly used for making tabular comparisons of method's parts and properties.

Simultaneously there is a lack of CASE tools to support these methods. Brinkkemper and Tolvanen [Bri89, Tol93] have tackled the problem of adaptation of

methods by metamodelling. The rapid growth of the number of both methods and their support environments has led to the proposition of a new area called Computer Aided Method Engineering, or CAME for short [Slo93, Kum92]. *Method engineering* is defined here, according to Heym & Österle, as the disciplined process of building, improving or modifying a method by means of specifying the method's components and their relations [Hey93].

We claim that by using a metamodel and a CAME environment for method engineering, we can achieve two goals simultaneously: first we can compare the methods analytically, and second, we can try out these methods in a situation, where methods have a platform that supports the storage and representation of descriptions made with these tools. Earlier work in this area has mainly concentrated on constructing method modelling (or metamodelling) languages [Bri90, Tol93] or building support environments for them [Che91, Sor88, Smo91b]. Earlier attempts to use metamodels for method comparison [Oei94] have mainly concentrated upon mapping metamodels into some "supermethod" or comparing metamodels by identifying their common parts [Hon93]. Instead we try to find quantitative measures, that can be computed without human judgement.

In this paper, we try to establish an approach which is systematic, automatic and easy to use for method measurement. We propose a metric approach and present a suite of metrics for methods. These metrics measure the complexity of the method or technique. Complexity is here classified into two categories: the complexity of learning and understanding the method, which is caused by the number of different constructs used in a technique or method [Tei80], and the complexity of the products, which is caused by the number of describing properties of objects and relationships.

The metrics can be used at least in two purposes, first, by method engineers to check the method properties and second, by method users to aid in the selection of methods, based on their measurable properties. The first aspect should be emphasized at the current status of method development, as we see a rapid appearance of new method categories, such as object-oriented [Boo94, Coa91] or business engineering methods [Dav90]. There is a clear push to develop new methods and consequently the method developers are in a hurry to come up with their own developments and variants of methods in the "fashionable" categories. The second aspect is more problematic, because the metrics by themselves cannot be used to judge the "goodness" or the appropriateness for the task of the method, instead they should be used together with approaches such as metamodel hierarchies [Oei94] and classification frameworks [Iiv83].

To test our claims we apply proposed metrics over a variety of well known methods. As a by-product, a set of tools for analysing methods within the MetaEdit CAME tool are introduced. The adaptation of the metrics into other CAME (and CASE) environments should be straightforward.

This paper is organized as follows. In the next section, we present the metamodelling technique used to describe techniques. In section 3, we present the proposed metrics and in section 4 these metrics are applied to a number of techniques and methods in the MetaEdit environment. The last section discusses the results and proposes some future directions for research.

2. Methods and the method engineering environment

Techniques of interest here consist of traditional graphical formalisms, such as Data Flow Diagrams, Entity Relationship Diagrams or Object Diagrams. Thus these techniques describe the object systems by objects and their relationships. In most cases the relationships and objects can have attributes or properties. The techniques usually describe only one aspect of an object system (such as data flows or state changes etc.). There is also a need to apply multiple views to describe the object system. In those cases we use organised sets of techniques, called *methods*. Methods contain several techniques, their interconnections and the use process of these techniques, but we currently limit our investigation to meta data models of the techniques and methods. An example of a method is Object Modelling Technique [Rum91] which consists of several techniques such as Class Diagrams, Data Flow Diagrams and Object State Diagrams. We chose those because their popularity and generality in the class of OO methods.

To be able to compare and analyse techniques, we need a way to describe them. This systematic way is called here a metamodelling technique. We use here the OPRR metamodelling technique, proposed by Welke [Wel92] and enhanced by Smolander [Smo91] to model the techniques and methods. The use of one metamodelling language gives us a neutral and non-biased basis to compare the properties of techniques, and it provides a common background for the formulation of metrics. Because we have also a CAME and a CASE shell environment, MetaEdit, which incorporates the OPRR model [Smo91b], we can develop automatically OPRR models and generate prototype support environments for a specified technique. In the following sections we describe the CAME environment, the static structure and the concepts of the OPRR technique and develop a model of OMT Class Diagrams [Rum91] using the OPRR technique.

2.1. The metrics environment

We have earlier developed tools for method engineering in Metaedit, one of which is a tool for developing methods with OPRR graphically (see [Ros94b]). It has been used to develop new methods for MetaEdit itself. We have currently implemented a collection of nearly forty development techniques [Ros94].

To test the metrics proposed in this paper, and to demonstrate the applicability of automating metrics computation procedures, we have implemented a metrics calculation package by using MetaEdit's report definition capabilities. List of techniques and methods, with the obtained metrics values is in Appendixes 1 and 3. The metrics computations and graphical outputs were produced using SPSS for Windows statistical package [SPS94].

2.2. The definition of the OPRR technique

The acronym OPRR comes from the words Object, Property, Role, and Relationship which are the *meta-types* in OPRR [Smo91]. Welke [Wel88] defines the meta-types in the following way:

- *Object* is a "thing" which exists on its own. Examples of objects are process, flow, store, source, module, etc.

flow, store, source, module, etc.

- *Properties* are the describing[qualifying characteristics associated with the other meta-types. Typical properties include name,description, etc.
- *Relationship* is an association between two or more objects. For example, there may be a relationship between a source and a process meaning that the process *uses* the source.
- *Role* is the name given to the link between an object and its connection with a relationship. From the example above, the process would be the *user* and the source would be the *origin* of the data.

Formally, a meta-model of a technique can be defined in OPRR as a six-tuple $M = \langle O, P, R, X, r, p \rangle$, where

O is a finite set of object types

P is a finite set of property types

R is a finite set of relationship types

X is a finite set of role types

r is a mapping $r: R \rightarrow \bigcup_{i=2}^{\infty} \left\{ \bigcup_{j=1}^{i} \{ X \times \wp(O) \} \right\}$

In other words, r is a total mapping from the relationship types to sets of cartesian products. This mapping shows the bindings of the relationships to their roles and of the roles to object types, that can act in those roles.

p is a mapping $p: NP \rightarrow \wp(P)$, where $NP = \{O \cup R \cup X\}$ is the set of *non-property types*. In other words, NP is a partial mapping from the non-property types to all subsets of property types. The mapping defines the property types associated with the non-property types.

In the sequel we will use indexes, e.g. O_T and M_T to indicate the meta model of a particular technique T. As said earlier we consider a method M to be a set of techniques. The meta model of the method is thus $M_{\mathcal{M}} = \bigcup_{T \in \mathcal{M}} M_T$, because we omit here the interconnections between methods. The metamodel of a method has been added to the original OPRR definition in [Smo91] to allow a simple handling of methods, which contain sets of techniques.

2.3. OPRR Definition of an example technique

We use here the definition of OMT method's Class Diagram technique [Rum91] as a source of metric values. OMT is an object-oriented method, which uses extensively graphical diagrams to describe information systems. The Class Diagrams are used for analysing and modelling class hierachies and the associations between classes. Their representations are depicted in Figure 3. Classes are connected to each other by Inheritance, Aggregation or Association relationships. Classes are connected to Objects by Instantiation relationships.

The Class Diagrams have been formally specified using the OPRR model. The result is the graphical OPRR model depicted in figure 2. (See [Smo91, Ros94b] for the definition of the graphical OPRR technique).

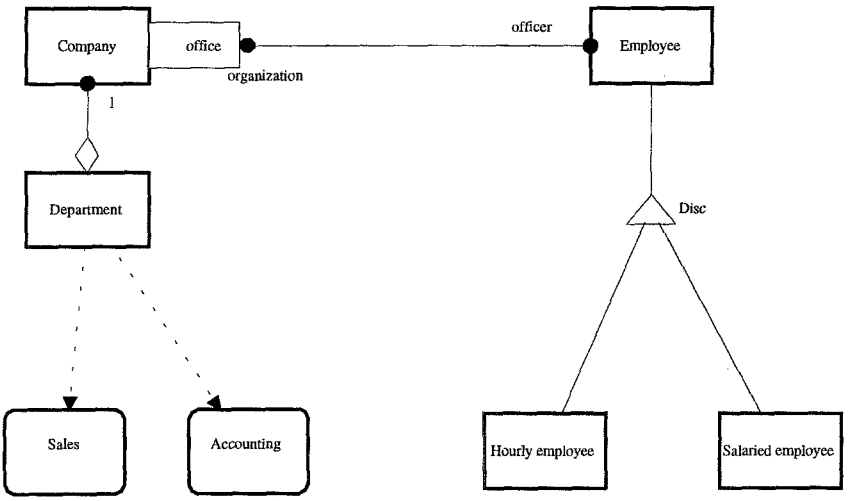

Fig. 1. Basic concepts of Class Diagrams.

The model contains some "dummy" objects in addition to those mentioned above to handle ternary relationships of the method with the binary OPRR relationships. The same method definition can be expressed in a sixtuple $M_{CD} = \langle O, P, R, X, r, p \rangle$ shown in table 1.

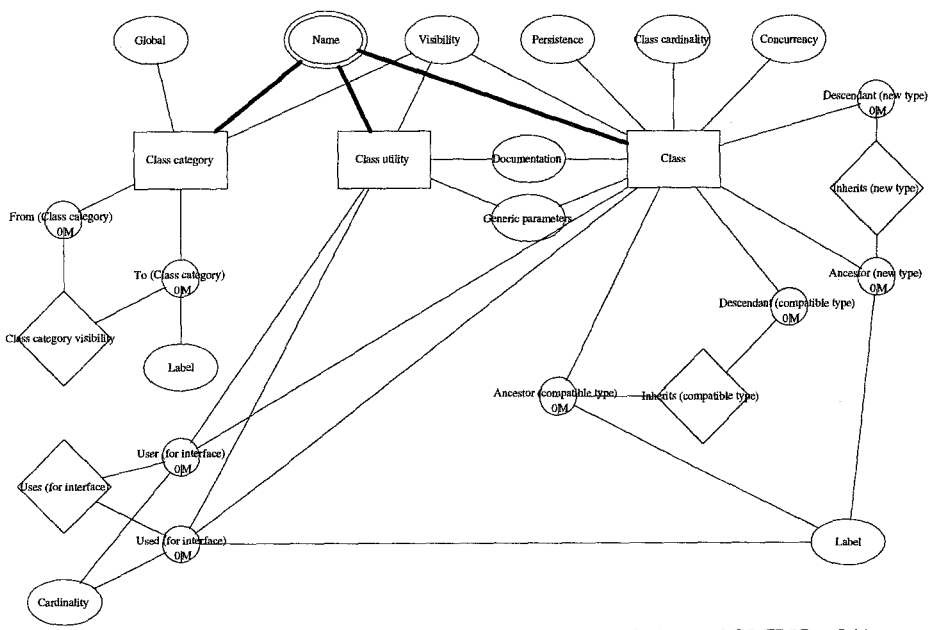

Fig. 2. Part of the OPRR Model of the Class Diagram technique of OMT]Ros94/

O= {Class, Disjoint divider, Divider, Object, Subclass group}

P= {Class name, Group name, ID, Object name, Aggregation name, Association name, Attributes, Discriminator, Link Attributes, Operations, Ordered, Qualifier, Role name, Cardinality}

R= {Aggregation 1 to 1, Aggregation 1 to M, Association (optional to 1), Association (optional to many), Association (optional to optional), Association 1 to 1, Association 1 to many, Association many to many, Generalisation, Generalisation/Specialisation, Instantiation, Qualified association 1 to 1, Qualified association 1 to many, Qualified association many to many, Specialisation}

X= {Ass 1 part, Ass M part, Ass opt part, Assembled to, Gen_from, Gen_to, Generalisation part, Instantiates, Part of, Qualified 1 part, Qualified M part, Specialisation_part, Superclass_part, is instance of}

r= {<Aggregation 1 to 1, {<Assembled to, {Class }>, < Part of, {Class}>}>
<Aggregation 1 to M, {<Ass M part, {Class, Class }>, < Assembled to, {Class}>}>,
<Association (optional to 1), {<Ass 1 part, {Class }>, < Ass opt part, {Class}>}>,
<Association (optional to many), {<Ass opt part, {Class }>, < Ass M part, {Class, Class}>}>,
<Association (optional to optional), {<Ass opt part, {Class }>, < Ass opt part, {Class}>}>,
<Association 1 to 1, {<Ass 1 part, {Class }>, < Ass 1 part, {Class}>}>, <Association 1 to many, {<Ass M part, {Class, Class }>, < Ass 1 part, {Class}>}>, <Association many to many, {<Ass M part, {Class, Class }>, < Ass M part, {Class, Class}>}>,
<Generalisation/Specialisation, {<Specialisation_part, {Class }>, < Generalisation part, {Class}>}>, <Instantiation, {<is instance of, {Class }>, < Instantiates, {Object}>}>,
<Qualified association 1 to 1, {<Ass 1 part, {Class }>, < Qualified 1 part, {Class}>}>,
<Qualified association 1 to many, {<Qualified 1 part, {Class }>, < Ass M part, {Class, Class}>}>; <Qualified association many to many, {<Ass M part, {Class, Class }>, < Qualified M part, {Class}>}>}

p= {<Class, {Class name, Operations, Attributes}>, <Disjoint divider, {ID, Discriminator}>, <Divider, {ID, Discriminator}>, <Object, {Object name, Attributes}>, <Subclass group, {Group name}>, <Ass 1 part, {Role name}>, <Ass M part, {Qualifier, Cardinality, Role name, Ordered}>, <Ass opt part, {Role name}>, <Assembled to, {Role name}>, <Qualified 1 part, {Qualifier, Role name}>, <Qualified M part, {Qualifier, Role name, Cardinality}>}

Table 1. OPRR definition of Class Diagrams

This example will be used in the following chapters as a source to define metric values.

3. Metrics for techniques and methods

This chapter outlines a number of metrics and their purpose. The metrics are derived and enhanced from metrics proposed in the earlier literature on the complexity of specification techniques [Tei80]. We describe the metrics on two levels: the technique level, which describes the characteristics of one technique and on the method level, which describes the complexity of a set of techniques.

Complexity is here classified into two categories: the complexity of learning and understanding the method, which is caused by the number of different constructs used in the technique or method [Tei80], and the complexity of the products, which is caused by the number of describing properties of objects and relationships. We are not trying to derive normative values such as "quality" or "learnability" from the measures, because these are not direct numerical attributes of the models [Fen94].

To avoid reported problems of design and specification metrics [Alb83, Hen81] i.e. poor theoretical foundations, being hard to analyse, and being flawed derivatives of code measures [Kit91, Roc94], we try to present the thing to be measured, a formal mathematical basis of the metric for the measure and guidelines for the interpretation

of the obtained values. Also the metrics are defined so that they are directly computable from the properties of the methods [Fen94].

For each metric the following is described: The formula for computing the metric, a brief explanation of the metric and the expected values of the metric and their interpretation. The expected values are given as box-plots, that have been obtained from the collection of techniques. The collection is represented in appendix 1. A box-plot is a five number summary of (minimum, lower quartile, median, higher quartile, maximum) [Tuk77]. The box-plot gives an interval, where the values should locate. If tighter intervals are needed, one could use for example box-plots or medians and variances from a category of techniques. For example in case of Class Diagrams we could use a group of object description methods. The box-plots can be found in Appendix 2.

The quartiles and median give the range of expected values for a given metric and most of the methods will fall into the range between a lower and upper quartile. If we find a significantly lower or higher value, there will be a need to analyse the reasons for that, and either change the method model or accept the the model as it is.

The metrics suggested here are not expected to produce nice regressions, and to infinitely come closer to a certain numerical value, as in traditional software science [Hal77]. The obtained metric values have the usual properties of metric data, i.e. the distributions are discrete, heavily skewed and there are a lot of outliers [Kit91], which make the usual statistical techniques unsuitable. Thus we apply data analysis and outlier analysis to unterstand the reasons for abnormal values. As Kitchenham points out, [Kit91] the interpretation of the results makes metric values meaningful, not their comparison with some arbitrarily given values. Yet, to make the judgements easier, we have derived some guiding values from the available material. The comparison of metric values between methods of similar species should be particularly fruitful.

3.1. Technique level

We assume a technique $M_T = \langle O_T, P_T, R_T, X_T, r_T, p_T \rangle$. We use the function $n(A)$ to denote the number of elements in the set of A. As all sets are considered to be finite (see section 2), this function always yields finite numbers.

Independent measures. The first measure is the number of object types used per technique. This measure, and the following two, are used while analysing the complexity of the technique on the basis of the number of concepts to be learned. They were suggested already by Teichroew et al. [Tei80]. The greater the number, the more complicated the technique. On the other hand, the technique with more concepts should also be able to capture more precise or detailed information about the object system, as claimed by Oei and Falkenberg [Oei94].

1. $n(O_T)$

The *count of object types per technique*. This metric shows the number of individual object types used to specify object systems. In the case of Class Diagrams, we find out that $n(O_{\text{Class Diagram}}) = 5$. This can be compared to the box-plot in appendix 2, and this shows, that the value is somewhat big, in the maximum line.

2. $n(R_T)$

The *count of relationship types per technique*. This is the number of concepts, which are used for describing connections between objects. The value $n(R_{Class\ Diagram}) = 15$ is marked as extreme value in the box-plot. The reason for the big number is partially the way of modelling the particular method in OPRR, because all the subtypes of relationships with different cardinalities have been modelled with separate relationship types. The reason for this choice is that the technique has been modelled for use with a CASE tool and the relationships with different graphical appearance have been modelled as different types.

The reader should notice, that the lower quartile and the minimum have the same value (1), and thus the number of relationship types tends to be quite low, between 1 and 5 for most techniques.

3. $n(P_T)$

The *number of property types per technique*. The value $n(P_{Class\ Diagram}) = 14$, is in the upper quartile line, but most CASE tools allow the specification of various properties per object or relationship type, so $n(P_T)$ can be rather high in comparison to $n(O_T)$ and $n(R_T)$. The reader should notice, that this is not the construct used directly in the following metrics, because the next metrics count properties per object, or a relationship type.

The following three metrics (formulae 5, 7 and 9) suggest metrics, that measure the complexity of the description of the object or relationship types.

4. $P_O(M_T, o) = n(p_T(O))$, where $o \in O_T$

5. $P_O(M_T, o) = \frac{1}{n(O_T)} \sum_{o \in O_T} P_O(M_T, o)$

The fourth formula is the *number of properties for a given object type*. It is defined separate by, in order to be able to define the method level summaries later. The fifth formula is the *average number of properties per object type*. This metric shows the average internal complexity of the object types in technique. The value $P_o(M_{Class\ Diagram}) = 2$ is quite typical, and it seems that most of the techniques fall into the range between one to three properties per object type.

6. $P_R(M_T, e) = n(p_T(e)) + \sum_{x \in r_T(e)} n(p(x \cdot i))$, where $i = (1,0)$ and $e \in R_T$

7. $\overline{P}_R(M_T) = \frac{1}{n(R_T)} \sum_{e \in R_T} P_O(M_T, o)$

The sixth formula is the *number of properties of a relationship type and its accompanying role types*. The inner sum of the equation counts the number of properties for all the role types associated with the current relationship type. Formula seven counts the *average number of properties per relationship type*. This metric shows the complexity of the interface between object types. The value $\overline{P}_R(M_{ClassDiagram}) = 4.0$ is quite normal.

8. $R_O(M_T, o) = n\left(\left\{e \in R_T : o \in \bigcup_{x \in e_T(e)} (x \cdot j)\right\}\right)$, where $j = (0,1)$ and $o \in O_T$

9. $\overline{R}_o(M_T) = \frac{1}{n(O_T)} \sum_{o \in O_T} n(R_O(M_T, o))$

Formula 8 gives the *number of relationship types that can be connected to a certain object type.* Formula 9 gives the *average number of relationship types that can be connected to a given object type.* This metric measures, how complicated it is to select the right connection between object types. For example a requirements analysis technique can just use one connection type, whereas a detailed design technique can present a large number of slightly different relationship types.

This metric was chosen instead of, for example the average number of object types, that can be connected by a given relationship type, because in the usual use the developers are faced with the selection of a relationship type between objects instead of making first a relationship and then selecting object types for the relationship. The value for $R_O(M_{\text{Class Diagrams}}) = 4.4$, which is in the upper quartile line and shows, that the method has quite simple descriptions of the interfaces between objects (formula 7 above), but a high number of relationship types in the interface. The result can be interpreted that the complexity of using the relationships in this method is in the selection of the right relationship type and not in describing further the connection the relationship represents.

Aggregate metrics. The independent metrics above described the individual characteristics of techniques. In this section we propose some aggregate metrics, that can be used to measure the overall complexity of the technique.

10. $C(M_T, o) = \frac{P_O(M_T, o)}{\sum_{\{R_i \in R : \exists b \in r(R_i = b \cdot i) \wedge (\exists \sigma \in (b \cdot j) : (o \in \sigma \cdot j))\}} P_R(M_T, e)}$, where $i = (1,0)$ and $j = (0,1)$

11. $\overline{C}(M_T) = \frac{1}{n(O_T)} \sum_{o \in O_T} C(M_T, o)$

The quotient (formula 10) shows the division of work in this technique, i.e. are things described by their internal properties, or external connections. The quotient will get higher values if there are many properties and a few relationship types with a few properties. The value for $\overline{C}(M_{\text{Class Diagram}}) = 0{,}91$ is quite close to the median line, and it shows that the method gives considerable importance to external connections.

12. $C'(M_T) = \sqrt{n(O_T)^2 + n(R_T)^2 + n(P_T)^2}$

The total conceptual complexity of a technique is not a straightforward measure, but we use the sum vector of the individual complexity factors of formula 1, 2 and 3. We propose to use it as the complexity vector in a xyz-coordinate system, that can be compared with other techniques.

In figure 3 we show the xy-plot of objects and relationships. It shows that Class Diagrams are more complex than average in number of relationships, but average in number of objects.

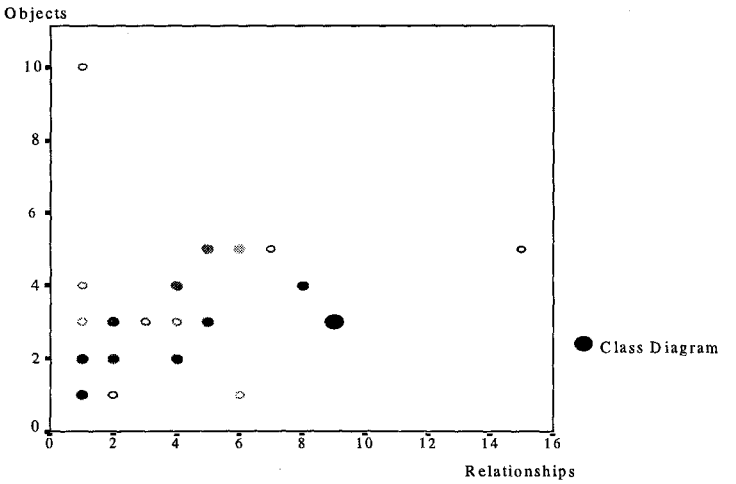

Fig. 3. Object / Relationship for Techniques

3.2. Method level metrics

Methods are here treated as collections of individual techniques, and thus we are omitting the problems related to the complexity of interconnected methods, due to the used meta modelling techniques inability to tackle with multiple techniques and their connections. This area clearly needs to be addressed in the future, but currently there is a lack of formal models of method interconnections as well as clear and unambigious decription of these interconnections in the methods [Kel94]. Thus method level complexities are simply summaries of individual technique complexities.

The cumulative complexities for method are counted first for each of the object, relationship and property types.

13. $$n(O_m) = \sum_{T \in \mathcal{M}} n(O_T)$$

14. $$n(R_m) = \sum_{T \in \mathcal{M}} n(R_T)$$

15. $$n(P_m) = \sum_{T \in \mathcal{M}} n(P_T)$$

The following are the aggregate complexity metrics for the method level.

16. $$\overline{C}(\mathcal{M}) = \sum_{T \in \mathcal{M}} \overline{C}(M_T)$$

17. $$C'(\mathcal{M}) = \sqrt{n(O_{\mathcal{M}})^2 + n(R_{\mathcal{M}})^2 + n(P_{\mathcal{M}})^2}$$

The cumulative complexity can either be defined as the cumulative value of each individual technique's complexity, or we can take the sum vector of the totals of formulae 13, 14 and 15. The cumulative complexity returns a value, that explains the total complexity of the method. The sum vector identifies the "style" of the method,

i.e. whether it describes the object systems by the properties, or relationships or objects, and whether these are used in a coherent and consistent style.

The values for OMT are the following: $n(O_{OMT}) = 13$, $n(R_{OMT}) = 19$ and $n(P_{OMT}) = 26$. The total complexity value is: $C'(M_{OMT}) = 35.92$. In Appendix 3 the values of these metrics are given for 15 methods. The reader should notice that we haven't divided the complexities by the number of techniques in a method $n(\mathcal{M})$. This would hide the overall complexity of methods with a large number of techniques.

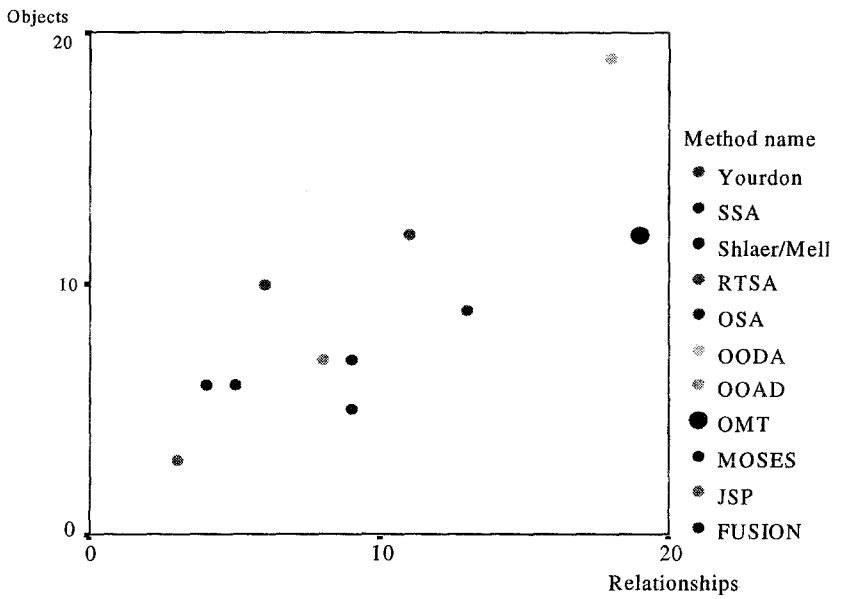

Fig. 4. Object / Relationship for Methods

On the method level it can be useful to check out the balance of individual techniques in the methods: i.e. if one of the techniques is very heavily loaded (i.e. has much more concepts than others), or the parts of the method are very different in style, this should be made explixit. In the case of OMT the Class Diagrams use 14 of the total of 19 relationship types and the other measures also have highest numbers in CD's. This means that the CD's are hard to learn and they are probably quite important for the method, because the main attention in development has been devoted to them. The checking ot the balancing can be done by counting the methods internal variances of each of the metrics and pointing out strange or extraordinary values.

4. Discussion and future research

In this paper we have proposed a set of metrics to evaluate the complexity of software development techniques and methods. By doing so we wish to guide and instruct the method developers to understand and analyse more systematically methods they suggest. Our goal is to establish one set of instrumental tests, that can be easily, and in a cost effective manner, used in evaluating methods. The proposed metrics are

relatively simple due to the fact, that they are easier to understand and there is not much point in developing more complex metrics before we know more about the nature and measuring of methods.

One interesting comparison could be made between the implementations of the same techniques or methods in different CASE tools. This could show some differences in the complexity of the use of one technique in different tool environments. Similarly the various implementations of rule checking in Entity Relationship diagramming and Data Flow diagramming in CASE tools has been compared by Vessey [Ves92]. The metrics have been applied here only with OPRR-models, but their adaptation for ER-based metamodels in other CASE tools should be straightforward.

The metrics proposed here analyse only the conceptual part of the techniques, and they should be accompanied with a set of metrics for the complexity of the resulting models. The analysis of resulting models could be used to verify the method complexity. There should be a negative correlation between the complexity of the method and the size of the produced models, if the methods conceptual complexity leads into greater expressiveness [Oei94]. We believe, that there is a balance between learnability and expressive power of the method, and the organisations selecting methods should be aware of the fact, that more powerful methods are harder to learn, but can be more effective for experienced users.

Furthermore the metrics should be tested by method developers and in that way we could reach a set of metrics that can predict the properties of techniques such as learnability and model size.

The limitations of the approach proposed here are: first, there is no way of representing constraints of the technique in OPRR and OPRR can only model the static aspects of the techniques. Secondly, OPRR is not capable of dealing appropriately with interconnected methods. Third, our values should be complemented with empirical experience from practical applications of methods in use.

In the future we'll have to consider integrated methods and derive metrics for them. In that work we'll need better understanding of integration of techniques and how that complicates, or simplifies, the methods. Also we should gather empirical material about the learnability of different techniques and their implementations and about the use of different constructs in different techniques.

Acknowledgements

We would wish to thank Steven Kelly for assistance in formalising the metrics, other members of MetaPHOR team, Design Methodology Research Group of University of Twente and anonymous reviewers.

References

[Alb83] Albrecht, A. J., J. E. Gaffney, "*Software Function, Source Lines of Code,* and Development Prediction: A Software Science Validation," IEEE Transactions on Software Engineering 9(**6**) (1983) pp.639--647.

[Boo94] Booch, G., "Object-Oriented Analysis and Design," Benjamin]Cummings, Redwood City, California (1994).

[Bri89] Brinkkemper, S., M. de Lange, R. Looman and F. H. G. C. van der Steen, *"On the Derivation of Method Companionship by Meta-Modelling,"* Imperial College, London, UK (1989).

[Bri90] Brinkkemper, S., *"Formalisation of Information Systems Modelling,"* Thesis Publishers, Amsterdam (1990).

[Cha91] Champeaux, D. de, *"A comparative study of Object Oriented Analysis Methods,"* Technical report Research Report, HP Laboratories (1991).

[Che91] Chen, M., J. F. Nunamaker Jr. and G. Mason, *"The Architecture And Design Of A Collaborative Environment For Systems Definition,"* DATA BASE (1991) pp.22--28.

[Coa91] Coad, P., E. Yourdon, *"Object-Oriented Analysis,"* Yourdon Press, Englewood Cliffs, New Jersey (1991).

[Dav90] Davenport, T. H., J. E. Short, *"The New Industrial Engineering: Information Technology and Business Process Redesign,"* Sloan Management Review (1990) pp.11--26.

[Fen94] Fenton, N., *"Software Measurement: A Necessary Scientific Basis,"* IEEE Transactions on Software Engineering 20(**3**) (1994) pp.199--206.

[Hal77] Halstead, M., *"Elements of Software Science,"* Elsevier North-Holland (1977).

[Hen81] Henry, S., D. Kafura, *"Software Structure Metrics Based on Information Flow,"* IEEE Transactions on Software Engineering 7(**5**) (1981) pp.510--518.

[Hey93] Heym, M., H. Österle, *"Computer-aided methodology engineering,"* INFORMATION AND SOFTWARE TECHNOLOGY 35(**6/7**) (1993) pp.345--354.

[Hon93] Hong, S., G. van den Goor and S. Brinkkemper, "A Comparison of Six Object-Oriented Analysis and Design Methods," in *Proceedings of the 26th Hawaiian Conference on Systems Sciences*, IEEE Computer Science Press (1993).

[Iiv83] Iivari, J., P. Kerola, "A sociocybernetic framework for the feature analysis of information systems development methodologies," in *Information Systems Methodologies: A Feature Analysis*, North--Hollland, Amsterdam (1983).

[Kel94] Kelly, S., K. Smolander, *"Evolution and Issues in MetaCASE,"* Information and Software Technology (1994).

[Kit91] Kitchenham, B., "Metrics and measurement," in *Software Engineer's reference book*, Butterworth-Heinemann, Oxford (1991).

[Kum92] Kumar, Kuldeep, Richard J. Welke, "Methodology Engineering: A Proposal for Situation Specific Methodology Construction," in *Challenges and Strategies for Research in Systems Development*, John Wiley & Sons, Washington (1992).

[Oei94] Oei, J. L. H., E. D. Falkenberg, "Harmonisation of information systems modelling and specification techniques," in *Methods and Associated Tools for the Information Systems Life Cycle*, Elsevier Science publishers (1994).

[Roc94] Roche, John M., *"Software Metrics and Measurement Principles,"* Software Engineering Notes 19(**1**) (1994) pp.77--85.

[Ros94] Rossi, M., J.-P. Tolvanen, *"Metamodeling approach to method comparison: A survey of a set of ISD methods,"* University of Jyväskylä, Jyväskylä (1994).

[Ros94b] Rossi, M., *"The MetaEdit CAME environment,"* University of Sunderland press, Sunderland (1994).

[Rum91] Rumbaugh, J., M. Blaha, W. Premerlani, F. Eddy and W. Lorensen, *"Object-Oriented Modeling and Design,"* Prentice--Hall, Englewood Cliffs, NJ, USA (1991).

[Slo93] Slooten, Kees van, Sjaak Brinkkemper, "A Method Engineering Approach to Information Systems Development," in *Procs. of the IFIP WG 8.1 Working Conference on the Information Systems Development Process*, North-Holland, Amsterdam (1993).

[Smo91] Smolander, Kari, "OPRR: A Model for Modelling Systems Development Methods," in *Next Generation CASE Tools*, IOS Press, Amsterdam, the Netherlands (1991).

[Smo91b] Smolander, Kari, Kalle Lyytinen, Veli-Pekka Tahvanainen and Pentti Marttiin, "MetaEdit --- A Flexible Graphical Environment for Methodology Modelling," in *Advanced Information Systems Engineering, Proceedings of the Third International Conference CAiSE'91, Trondheim, Norway, May 1991*, Springer-Verlag, Berlin (1991).

[Son92] Song, X., L. Osterweil, *"Towards Objective and Systematic Comparisons of Software Design Methodologies,"* IEEE Software (1992).

[Sor88] Sorenson, Paul G., Jean-Paul Tremblay and Andrew J. McAllister, *"The Metaview System for Many Specification Environments,"* IEEE SOFTWARE (1988) pp.30--38.

[SPS94] SPSS, , *"SPSS for Windows 1.0 Reference Guide,"* SPSS Inc., Chicago, USA (1994).

[Tei80] Teichroew, Daniel, Petar Macasovic, Ernest A. Hershey III and Yuzo Yamamoto, "Application of the entity-relationship approach to information processing systems modeling," in *Entity-Relationship Approach to Systems Analysis and Design*, North-Holland (1980).

[Tol93] Tolvanen, J.-P., K. Lyytinen, *"Flexible method adaptation in CASE environments - The metamodeling approach,"* Scandinavian Journal of Information Systems 5(**1**) (1993) pp.51-77.

[Tuk77] Tukey, J. W., *"Exploratory Data Analysis,"* Addison Wesley, Reading, MA (1977).

[Ves92] Vessey, I., S. L. Järvenpää and N. Tractinsky, *"Evaluation of Vendor Products: CASE Tools as Methdology Companions,"* Communications of the ACM 35(**4**) (1992) pp.90-105.

[Wel92] Welke, R. J., "The CASE Repository: More than another database application," in *Challenges and Strategies for Research in Systems Development*, Wiley, Chichester UK (1992).

Appendix 1. Values obtained from 34 techniques

This table lists the values of 34 techniques modelled by OPRR in the MetaEdit environment. The table shows the name of the method, the name of the technique and the values obtained for the formulae, which are referred by their number in the main text.

Method	Technique	1	2	3	5	7	9	11	12
Yourdon	Entity Relationship Attribute Diagram	3	2	6	2,33	1,67	,50	1,44	7
Yourdon	Data Flow Diagram	3	2	6	1,67	2	1	,42	7
Yourdon	Structure Chart	1	1	4	2	1	2	,67	4,24
Yourdon	State Transition Diagram	3	1	5	2	1	4	,40	5,92
OODA	Class Diagram	3	9	10	4,67	5	1,44	,70	13,78
OODA	Module Diagram	10	1	4	3	1	1	1,50	10,82
OODA	Object Diagram	1	6	8	5	6	4	,17	10,05
OODA	Process Diagram	2	1	6	4	1	3	1	6,40
OODA	State Transition Diagram	3	1	4	1	1	3	,25	5,10
JSP	Data Structure Diagram	1	1	3	3	1		3	3,32
JSP	Program Structure Diagram	2	2	5	2,50	1,50		1,75	5,74
OOAD	Object Oriented Analysis and Design	3	5	6	2,33	4	1	,37	8,37
OOAD	Service Chart	3	2	7	2,67	2	,50	,89	7,87
OOAD	Object State Diagram	1	1	3	2	1	1	1	3,32
OMT	Class Diagram	5	15	14	2	4,40	4	,91	21,12
OMT	Data Flow Diagram	3	3	8	1,67	3	2	,19	9,06
OMT	State Diagram	4	1	4	1,50	1	2	,50	5,74
FUSION	Object Model	4	4	8	2	2,25	,75	,54	9,80
FUSION	Object Interaction Graph	2	1	7	3	1	3	,75	7,35
MOSES	O/C Model	4	8	10	4,50	6	1,88	,67	13,42
MOSES	Event Model	1	1	6	3	1	4	,60	6,16
Shlaer/ Mellor	Information Structure Diagram	2	4	8	2,50	2	5	1,06	9,17
Shlaer/ Mellor	Action Data Flow Diagram	2	4	7	2,50	3	1,75	,30	8,31
Shlaer/ Mellor	State Transition Diagram	3	1	5	2	1	4	,40	5,92
OSA	Object Behavior Model	3	4	6	3	4	1	,38	7,81
OSA	Object Interaction Model	1	2	4	1	2	2,50	,14	4,58
OSA	Object Relationship Model	5	7	11	1,80	2,20	1,57	,75	13,96
Goldkuhl	Activity model	5	5	8	1,60	2,80	,20	,55	10,68
Demeter	Demeter	3	2	5	4	2	,50	1,33	6,16
Express	EXPRESS-G	5	6	13	2,20	4	2,67	,14	15,17
SSA	Structured Systems Analysis	3	2	10	3	2	2	,50	10,63
SSA	Entity Relationship Attribute Diagram	3	2	6	2,33	1,67	,50	1,44	7

RTSA	Real-Time Structured Analysis	5	7	4	2,20	4,20	1	,30	9,49
RTSA	Entity Relationship Attribute Diagram	3	2	6	2,33	1,67	,50	1,44	7
RTSA	Structure Chart	1	1	4	2	1	2	,67	4,24
RTSA	State Transition Diagram	3	1	5	2	1	4	,40	5,92

Appendix 2. Boxplots for Techniques

The 5-point box-plots can be read as (from left to right): bar representing minimun, box starting from lower quartile, in the box there is the median bar and end of the box is the upper quartile, fifth bar is the maximum. The outliers are indicated by a marker with the name of the technique (objects number with the * indicating Booch module diagrams is an example). **Notice**, that the scales differ from figure to figure to ease the reading of the boxes.

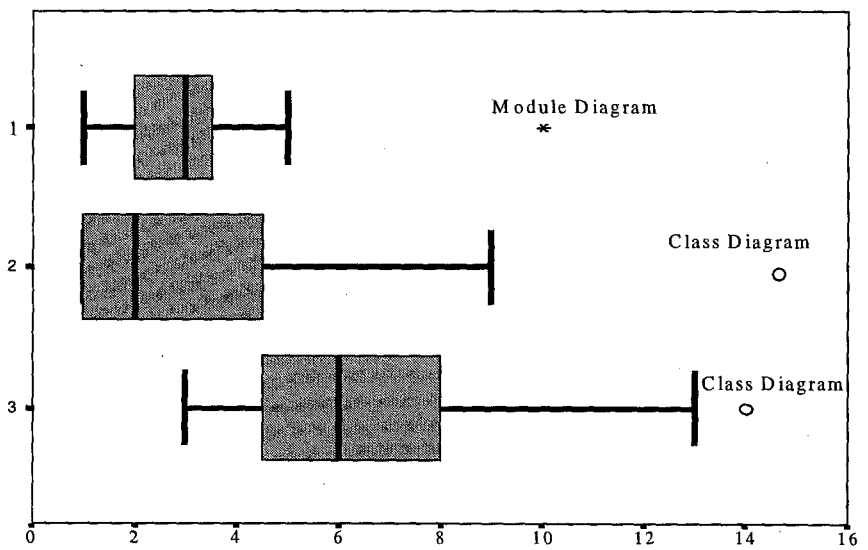

Fig. 5. Boxplots for formulae 1, 2 and 3 for techniques

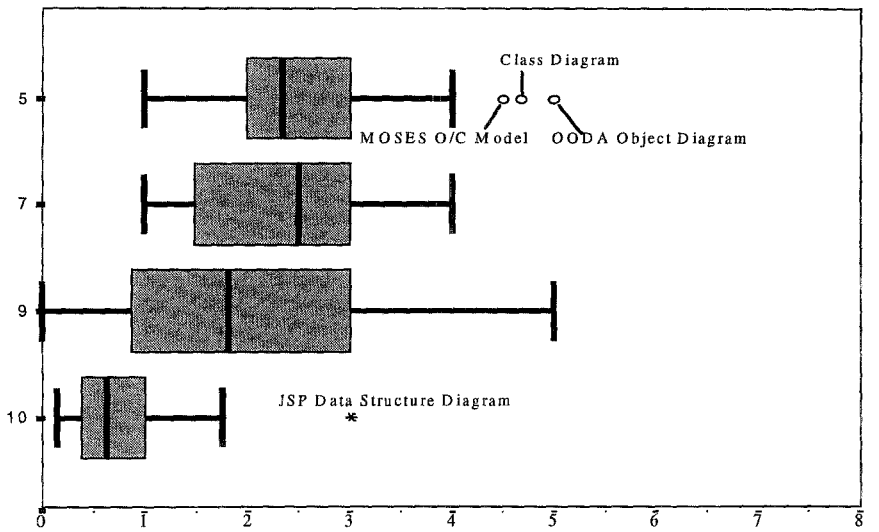

Fig. 6. Boxplots for formulae 5, 9, 7, 10 for techniques

Appendix 3. Values obtained from 15 methods

Method	n*	13	14	15	16	17
FUSION	2	6	5	15	1,20	17,15
JSP	2	3	3	8		9,06
MOSES	2	5	9	16	2,11	19,58
OMT	3	12	19	26	3,58	35,92
OOAD	3	7	8	16	,88	19,56
OODA	5	19	18	32	2,44	46,15
OSA	3	9	13	21	1,54	26,36
RTSA	4	12	11	19	1,27	26,65
Shlaer/Mellor	3	7	9	20	3,44	23,39
SSA	2	6	4	16	1,25	17,63
Yourdon	4	10	6	21	1,50	24,16

* n is the number of techniques in the method