Pergamon

Information Systems Vol. 21, No.2, pp. 209-227, 1996

S0306-4379(96)00012-9 0306-4379/96 $15.00 + 0.00

COMPLEXITY METRICS FOR SYSTEMS DEVELOPMENT METHODS AND TECHNIQUES

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

[2] University of Twente, Department of Computer Science, P.O. Box 217, NL-7500 AE Enschede, the Netherlands

(Received 10 August 1995; in final revised form 13 March 1996)

Abstract — So many systems 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 single diagram techniques, and of complete systems development methods. The proposed metrics provide a relatively fast and simple way to analyse the descriptive capabilities of a technique or method. When accompanied with other selection criteria, the metrics can be used for estimating the relative complexity of a technique compared to others. To demonstrate the applicability of the metrics, we have applied them to 36 techniques and 11 methods.

Key words: Method Evaluation, Metrics, Metamodel

1. INTRODUCTION

Recent years have witnessed the appearance of new systems development paradigms and methods. Examples of these are object-oriented analysis and design methods, and business process re-engineering methods. However, we feel that there is a need for improvement in the analysis of these methods and in understanding their use and functionality. Although some attempts have been made to compare existing methods (e.g. object-oriented methods [10, 17, 25] and a number of general comparisons in the CRIS papers [32, 33]), the studies lack rigour and a sound conceptual foundation, and are mostly based on ad hoc feature analysis techniques. Some recent attempts [23, 40] have proposed more systematic approaches, based on a common formal metamodelling language to describe methods. These hold the promise of a more systematic and analytic way to compare methods; however, they are still mainly used for making tabular comparisons of methods' parts and properties.

Alongside these patterns there is a lack of CASE tools to support these methods. Brinkkemper et al. [6] and Tolvanen and Lyytinen [44] 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 [29, 22]. *Method engineering* is defined here as the engineering discipline to design, construct, and adapt methods, techniques and tools for the development of information systems [5].

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 on a platform that supports the storage and representation of descrip tions made with this method. Research in this area has mainly concentrated on constructing method modelling (or metamodelling) languages [7, 44] or building support environments for them [11, 38, 41]. Earlier attempts to use a common metamodelling language for method comparison have mainly concentrated upon mapping methods onto some "supermethod" [31] or comparing models of methods by identifying their common parts [23]. Instead of these approaches we now try to find quantitative measures of techniques' properties that can be computed without human judgement.

In this paper, we try to establish an approach for method measurement which is systematic, automatic and easy to use. We propose a metric approach and present a suite of metrics for methods. These metrics

measure the complexity of the method or technique. The metrics are proposed on the basis of metamodels. We define a *metamodel* to be a conceptual model of a development method. Metamodels provide a model of the syntactical structure (or representation) of the specification technique.

The relative complexity of methods and techniques based on metamodels is significant because it can be expected to affect the learnability and ease of use of a method. A metamodel has a type-instance relationship with the actual application models, a simple metamodel generally leading to a more complex application model. Or said differently, a metamodel models the expressive power of the specification technique, by representing its vocabulary (i.e. the concepts and properties) and admissible constructs (i.e. the relationships and the roles). There should be a negative correlation between the complexity of the method and the size of the application models, if the method's conceptual complexity does indeed lead to greater expressive power [31]. Consequently, a simple method (metamodel) may be easy to learn as such, but may be more laborous to apply when leading to more complex application models. Furthermore, as the metamodels are used to generate model editors for a CASE tool, the metamodels serve also as an indication of the functional complexity of the generated editor. So, generally speaking, the complexity of a method is related to the learnability and ease of use of the method, even though this relationship may be complex.

Or said differently, the metamodels model the expressive power of the specification technique, by representing its vocabulary (i.e. the concepts and properties) and admissible constructs (i.e. the relationships and the roles). Furthermore, as the metamodels are used to generate model editors for a CASE tool, the metamodels serve also as an indication of the functional complexity of the generated editor. So, generally speaking, the expressive power of a specification technique is related to the learnability of the technique. We therefore claim that there exists an intrisic dependency between the metamodels and the learnability of the specification technique. Whether this dependency can be formulated as "the more complex a metamodel, the harder the method is to learn" is not a statement we will make based on our results. This remains to be investigate in future empirical research.

The metrics can be used for at least 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 [4, 9, 12, 13, 35, 44] or business engineering methods [14]. There is a clear push to develop new methods and consequently the method developers are falling over themselves 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, but rather should be used combined with approaches such as metamodel hierarchies [31] and classification frameworks [24].

To test our claims we apply the 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 language used to describe methods. 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. The appendices contain tables and graphs with the values of the metrics for the methods in Section 4.

2. METHODS AND THE METHOD ENGINEERING ENVIRONMENT

Techniques of interest here consist of traditional graphical formalisms, such as Object Diagrams and Object State Diagrams. These techniques describe the object systems by objects and their relationships. In many 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 of these techniques [5], but we currently limit our investigation to considering methods as simple compositions of the technique

components. An example of a method is the Object Modelling Technique [35] which consists of several techniques such as Class Diagrams, Data Flow Diagrams and Object State Diagrams. See appendix 1 for the list of methods and techniques considered in this investigation.

To be able to compare and analyse techniques, we describe their structure using one common language to define the metamodels of the methods. We use here the OPRR (Object, Property, Relationship, Role) method modelling language, proposed by Welke [45] and enhanced by Smolander [38] to model the techniques and methods. The use of one method modelling language gives us a basis to compare the properties of techniques, and it provides a common background for the formulation of metrics.

It is important to notice, however, that there are a few factors that can bias the results. First, the precision of the method description is dependent on the quality of the technique's description in the textbook. Some authors present detailed formal descriptions of their techniques, and others define their techniques more vaguely. Secondly, the experience and personal preferences of the method modeller affect the model. For example, a given concept might be metamodelled either as a property or as a relationship. In this particular case all of the techniques have been modelled by one experienced method engineer, and we can thus expect that they have been modelled with a consistent style.

In the following sections we describe the CAME environment, the static structure and the concepts of OPRR and develop a model of OMT Class Diagrams [35] using OPRR.

2.1. The CAME Environment

We have used the method engineering environment of the MetaEdit CASE shell, which is based on OPRR [38, 36]. MetaEdit supports the development of method models by allowing their graphical description using OPRR and by translating the method descriptions automatically into diagram editors within the CASE shell. It has been used to develop new methods for MetaEdit itself. We have currently implemented a collection of nearly forty development techniques [34]. All of the metamodels have been developed by one person. The ability to try out modelled methods in the diagram editor in MetaEdit was essential in this research project, as this ensured thet the metamodels used for evaluation of the metrics were complete and correct.

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 using MetaEdit's report definition capabilities. A list of techniques and methods, together with the obtained metrics values, is to be found in Appendices 1 and 3. The metrics computations and graphical outputs were produced using the SPSS for Windows statistical package [42].

2.2. The Definition of OPRR

The acronym OPRR comes from the words Object, Property, Role, and Relationship which are the *meta-types* in OPRR [39]. Welke [45] 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.
- *Properties* are the describing or qualifying characteristics associated with the other meta-types. Typical properties include name, description, definition, 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.

In MetaEdit's CAME environment we have extended OPRR by defining explicit mappings from these concepts onto their representations. We have used the following notation [38]: an object type is represented by a rectangle, a property type by an ellipse, a role type by a circle and a relationship type by a diamond.

The name of each object, property, role or relationship type is written inside its symbol. An example of such a graphical OPRR model is in figure 1.

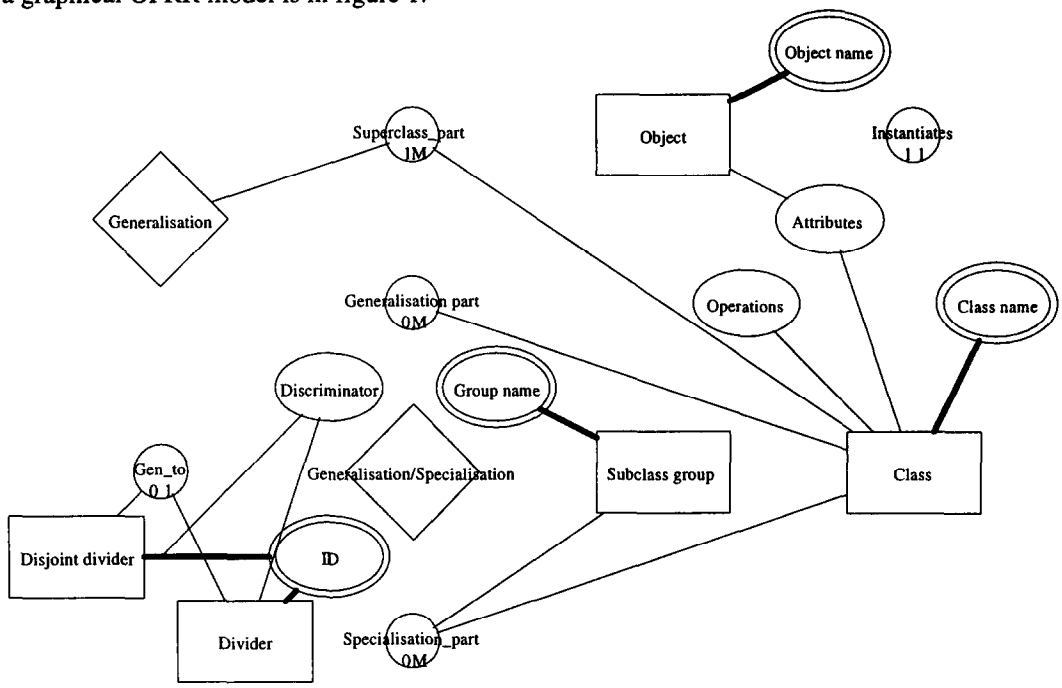

Fig. 1: OPRR Model of the Class Diagram technique of OMT (excerpted, see [34])

Formally, a 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 $r: R \rightarrow \left\{ x \middle| x \in \wp\left(X \times \left(\wp(O) - \{\varnothing\}\right)\right) \wedge n(x) \geq 2 \right\}$, where n(x) is the cardinality of x and $\wp(O)$ is the power set of set O.

In other words, r maps a relationship type to a member of the set of powersets of role types and powersets of objects, i.e. r(e) = {<a_role, {objects}>, ...}, where $e \in R$. This mapping links the role types to the relationship types on the one hand and to the object types on the other hand.

Here we define two functions, that will be used later. Suppose $x \in r(e)$, i.e. x takes the form of <a_role, {objects}>, the function role(x) returns the role included in x and function objects(x) returns the object set of x.

p is a partial mapping $p: NP \rightarrow \wp(P)$, where $NP = \{O \cup R \cup X\}$ is the set of *non-property types*. In other words, p 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 what follows we will use indices, e.g. O_T to indicate the object types, and M_T for the model of a particular technique T. As noted earlier, we consider a method $\mathcal{M}$ to be a set of techniques. The model of

the method is thus $M_{\mathcal{M}} = \bigcup_{T \in \mathcal{M}} M_T$, because we omit here the interconnections between techniques. The concept of model of a method has been added to the original OPRR definition in [38, 39] to allow simple handling of methods which contain sets of techniques.

2.3. OPRR Definition of an Example Technique

In this paper we use the definition of the OMT method's Class Diagram technique [35] as a running example for the discussion of metric values. OMT is an object-oriented method, which extensively uses graphical diagrams to describe information systems. The Class Diagrams are used for analysing and modelling class hierarchies and the associations between classes. A sample Class Diagram is given in Figure 2. Classes are connected to each other by Inheritance, Aggregation or Association relationships. Objects are connected to Classes by Instantiation relationships. Note that our choice of an object-oriented method does not exclude the application of the metrics to conventional methods and techniques.

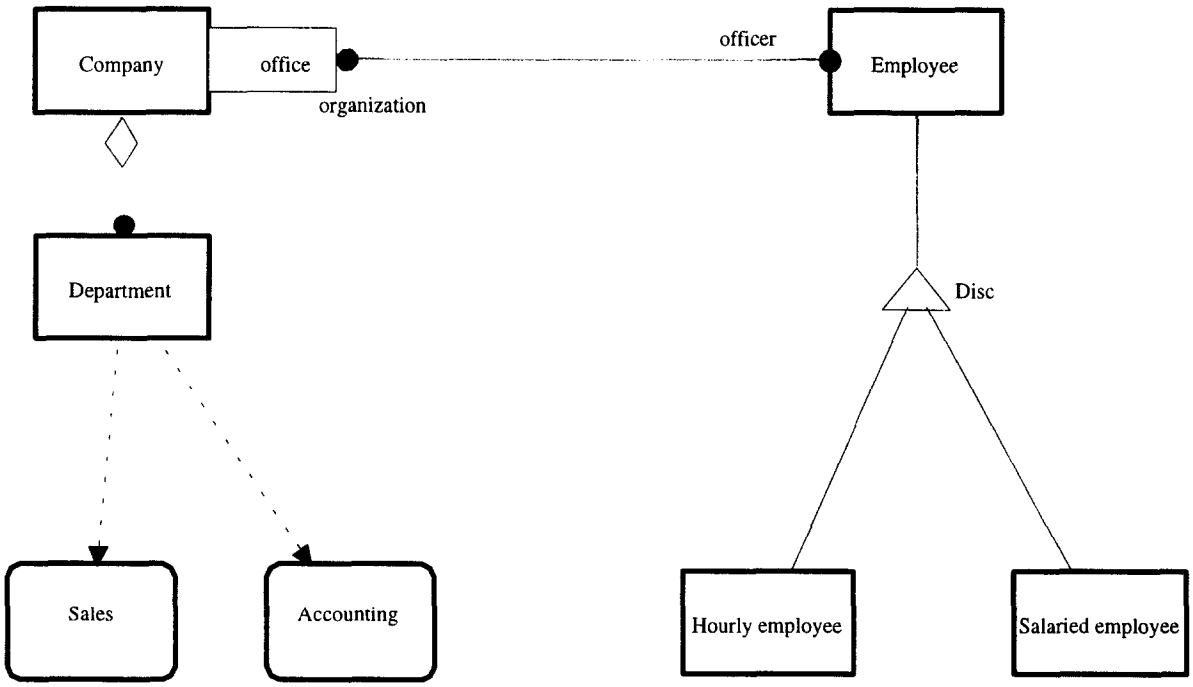

Fig. 2: A sample Class Diagram

The Class Diagram technique has been formally specified using OPRR. The result is expressed in the sixtuple $M_{CD} = \langle O, P, R, X, r, p \rangle$ as shown in Table 1. The equivalent graphical OPRR model is, for the sake of brevity, given only partially in figure 1. This model shows among other things the naming of Objects and Classes, and the Generalisation/Specialisation hierarchy.

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

3. METRICS FOR TECHNIQUES AND METHODS

This Section outlines a number of metrics and their purpose. The metrics are derived and enhanced from metrics proposed in earlier literature for the complexity of specification techniques [43]. We describe the metrics on two levels: the technique level, which describes the characteristics of one technique, and the method level, which describes the complexity of a set of techniques.

We restrict the complexity of techniques and methods to two aspects. On the one hand we try to measure the complexity of learning and understanding the technique, which is related to the number of different concepts and constructs (say, object types and relationship types) used in the technique. On the other hand it is desirable to get insight in the complexity of the internal structure of the models resulting from applying the technique. This complexity is dependent on the number of describing properties of the technique's 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 methods [16]. This would require extensive empirical evidence from systems development practice.

3.1. Basic Preliminaries

Design and specification metrics, found in, for example, Albrecht's and Henry and Kafura's work [1, 21], have several reported problems, such as poor theoretical foundations, being hard to analyse, and being

flawed derivatives of code measures [26, 30]. In order to avoid this, we present a formal mathematical basis of the metrics and guidelines for the interpretation of the obtained values. Furthermore, the metrics are defined so that they are directly computable from the properties of the models of the methods [16].

For each metric the following is described: the Formula for computing the metric, a brief explanation of the metric, the range in which the values obtained from data in Appendix 1 are located, and some comments on their interpretation. The range is given as a box-plot, which is a five number summary of smallest observed value that isn't outlier, lower quartile, median, higher quartile and largest observed value that isn't outlier [42, 45]. Notice that extreme values (values, which are more than 1.5 times the difference between the lower quartile and upper quartile outside of the quartiles) are plotted separately.

The box-plot gives an interval where the values have been observed. If a more precise positioning of a technique was needed, one could use for example box-plots or medians and variances from a particular category of techniques. For example, in the case of Class Diagrams we could take the category of class description techniques contained in object-oriented methods [23]. The box-plots for techniques and methods can be found in Appendices 2 and 4 respectively.

The quartiles and median give the range of observed values for a given metric: most of the techniques 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 it.

The obtained metric values have the usual properties of software metric data, i.e. the distributions are discrete, heavily skewed and there are a lot of outliers [28], which make the usual statistical techniques unsuitable. Thus we apply data analysis and outlier analysis as ways of presenting the data. As Kitchenham [28] points out, 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.

3.2. Technique Level Metrics

Let the model of a technique T be given as $M_T = \langle O_T, P_T, R_T, X_T, r_T, p_T \rangle$. 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.

3.2.1. 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. These measures were already suggested by Teichroew et al. [43]. We assume that a technique with many concepts is more complex to learn, than one with fewer concepts. On the other hand, a 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 [31].

Definition 1 $n(O_T)$ is 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 OMT Class Diagrams, we find out that $n(O_{\text{Class Diagram}}) = 5$. This can be compared to the range of values derived from the full set of 36 techniques shown in the box-plot in Appendix 2, where we see, that the value is on the maximum line. This means that the OMT Class Diagram technique possesses a relatively large number of object types.

Definition 2 $n(R_T)$ is the *count of relationship types per technique.*

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

Definition 3 $n(P_T)$ is the *number of property types per technique.*

The value, $n(P_{\text{Class Diagram}}) = 14$, is an outlier, which shows that the Class Diagrams use the largest number of properties from the techniques observed. Note that most CASE tools allow the specification of various properties per object or relationship type (such as definition, user comments, etc.), so $n(P_T)$ can be rather high in comparison to $n(O_T)$ and $n(R_T)$. However, in the particular case of OMT Class Diagrams, there are more relationship types than property types. The reader should notice that this metric measures the total number of property types in the whole technique, whereas the following metrics count properties of individual object types or relationship types.

The following three metrics (Formulae 5, 7 and 9) suggest metrics that aim at describing the complexity of the description of the object or relationship types.

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

Definition 5 $\overline{P}_O(M_T) = \frac{1}{n(O_T)} \sum_{o \in O_T} P_O(M_T, o)$

The fourth Formula is the *number of properties for a given object type.* It is defined separately in order to define the aggregate metrics for the technique in Section 3.2.2. The fifth Formula is the *average number of properties per object type.* This metric shows the average complexity of the descriptions of the object types in a technique. The value $\overline{P}_O(M_{\text{Class Diagram}}) = 2$ is quite typical, and it seems that most of the techniques fall into the range of one to three properties per object type.

Definition 6 $P_R(M_T, e) = n(p_T(e)) + \sum_{x \in r_T(e)} n(p(\text{role}(x)))$, where $e \in R_T$.

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

The sixth Formula is the *number of properties of a relationship type and its accompanying role types.* Inside the summation of Formula 6 the number of properties for all the role types associated with the current relationship type is counted. Formula seven counts the *average number of properties per relationship type.* This metric shows the complexity of the interface between object types. The value $\overline{P}_R(M_{\text{ClassDiagram}}) = 4.4$ is below the maximum line. This indicates, together with the high number of relationship types in Formula 2, that the OMT Class Diagrams use a large number of simple relationships and thus put more emphasis on the types of the relationships than on their content.

Definition 8 $R_O(M_T, o) = n\left(\left\{e \in R_T \middle| o \in \bigcup_{x \in r_T(e)} \text{objects}(x)\right\}\right)$, where $o \in O_T$.

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

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 which is interested only in the existence of relationships between object types, not in their content, 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 normal application of a technique 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 $\overline{R}_O(M_{\text{Class Diagrams}}) = 4.0$, which is on the upper quartile line and shows that the technique 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 as showing that the complexity of using the relationships in this technique is in the selection of the correct relationship type and not in the description of the relationship.

3.2.2. 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.

Definition 10 $C(M_T, o) = \frac{P_O(M_T, o)}{\sum_{e \in A} P_R(M_T, e)}$, where $A = \left\{x \in R_T \middle| o \in \bigcup_{y \in r_T(x)} (\text{objects}(y))\right\}$

Definition 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 by external connections. The quotient will get higher values if there are many properties and a few relationship types with a few properties. Formula 11 gives the average for the whole technique. The value for $\overline{C}(M_{\text{Class Diagram}}) = 0{,}91$ is quite close to the upper quartile line, and it shows that the technique gives considerable importance to the properties of objects.

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

This Formula, the total conceptual complexity of a technique is not a straightforward measure, but we use the modulus vector of the individual complexity factors of Formulae 1, 2 and 3. We propose to use it as the complexity vector in a three-dimensional coordinate system. The vector can be compared with those for other techniques. The idea of using the complexity vector is that one can see the complexity of the technique by looking at how long the vector is and at the same time one can see the "style" of the technique by looking at in which direction the vector goes. For example the Class Diagram technique is the most complex by this measure as it uses properties and relationships extensively, but it contains an average number of objects. In Figure 3 we show an xyz-plot of the modulus vectors. In fact, as the box-plots in

Figure 5 show, Class Diagrams are marked as extreme values in both the number of relationships and properties.

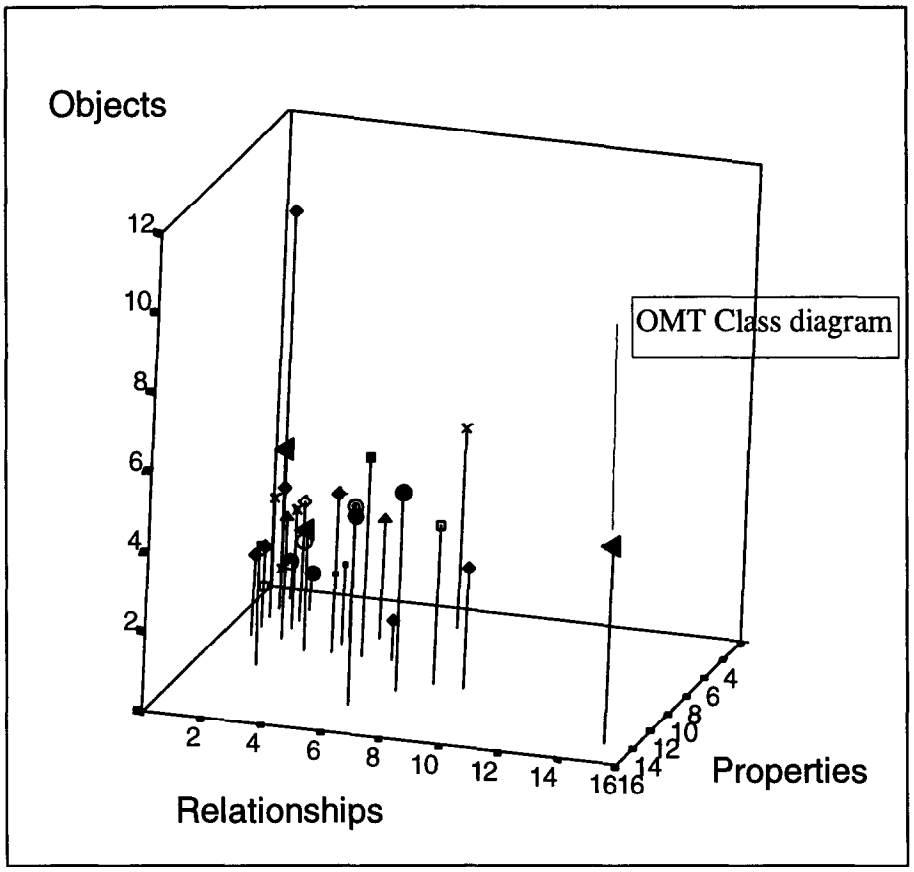

Fig. 3: Object-Relationship-Property cube for Techniques

3.3. Method Level Metrics

Methods are treated here as collections of individual techniques, and thus we are omitting the problems related to the complexity of interconnected methods, due to the inability of OPRR to deal with the connections of multiple techniques. This area clearly needs to be addressed in the future, but currently there is a lack of formal models of technique interconnections as well as a clear and unambigious description of these interconnections in the method descriptions in textbooks [27]. Thus method level complexities are simply summaries of individual technique complexities.

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

Definition 13 $n(O_{\mathcal{M}}) = \sum_{T \in \mathcal{M}} n(O_T)$

Definition 14 $n(R_{\mathcal{M}}) = \sum_{T \in \mathcal{M}} n(R_T)$

Definition 15 $n(P_{\mathcal{M}}) = \sum_{T \in \mathcal{M}} n(P_T)$

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

Definition 16 $\overline{C}(\mathcal{M}) = \frac{1}{n(O_{\mathcal{M}})} \sum_{T \in \mathcal{M}} \sum_{o \in O_T} C(M_T, o)$

Formula 16 gives the division of work between the objects and relationships in the whole method. It is summed for individual objects and their properties and relationships in each of the techniques methods.

Definition 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 mainly by properties, relationships, objects, and whether these are used in a coherent and consistent style. In Figure 4 we show these complexity vectors in an xyz-plot.

The values for OMT are the following: $n(O_{OMT}) = 12$, $n(R_{OMT}) = 19$, $n(P_{OMT}) = 26$ and $\overline{C}(OMT) = 0{,}59$. The total complexity value is: $C'(\text{OMT}) = 34.73$. In Appendix 3 the values of these metrics are given for 11 methods. The reader should notice that we have not 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. We therefore present the methods in appendix 3 in order of the number of techniques and to the value of $C'(M)$. This enables us to place a method among methods of comparable complexity.

At the method level we can observe that the OMT has the largest number of relationships and the second largest number of properties and objects. As the most complex method by these measures is OODA [4], we claim that the new object-oriented methods may have a tendency to be more complex than traditional 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 has many more concepts than others, or the parts of the method are very different in style, this should be made explicit. In the case of OMT, the Class Diagrams use 14 of the total of 19 relationship types and the other measures also have their highest values for Class Diagrams. This means that the Class Diagrams may be harder to learn and apply in practice and they are probably quite important for the method. The checking of the balance can be done by counting the method's internal variances for 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 describe something we denote as "the complexity" of systems development techniques and methods. By doing so we wish to guide and instruct the method developers to understand and analyse the methods they suggest. Our goal is to establish one set of instrumental tests, that can be used easily and in a cost effective manner as an aid in evaluating methods. To obtain a thorough understanding, one needs to use these metrics together with other comparison aids such as Iivari's classification hierarchies [24] and tabular feature comparisons [10, 23]. The proposed metrics are relatively simple to understand and easily implemented in a tool. Furthermore, there is little point in developing more complex metrics before we know more about the nature and measuring of methods.

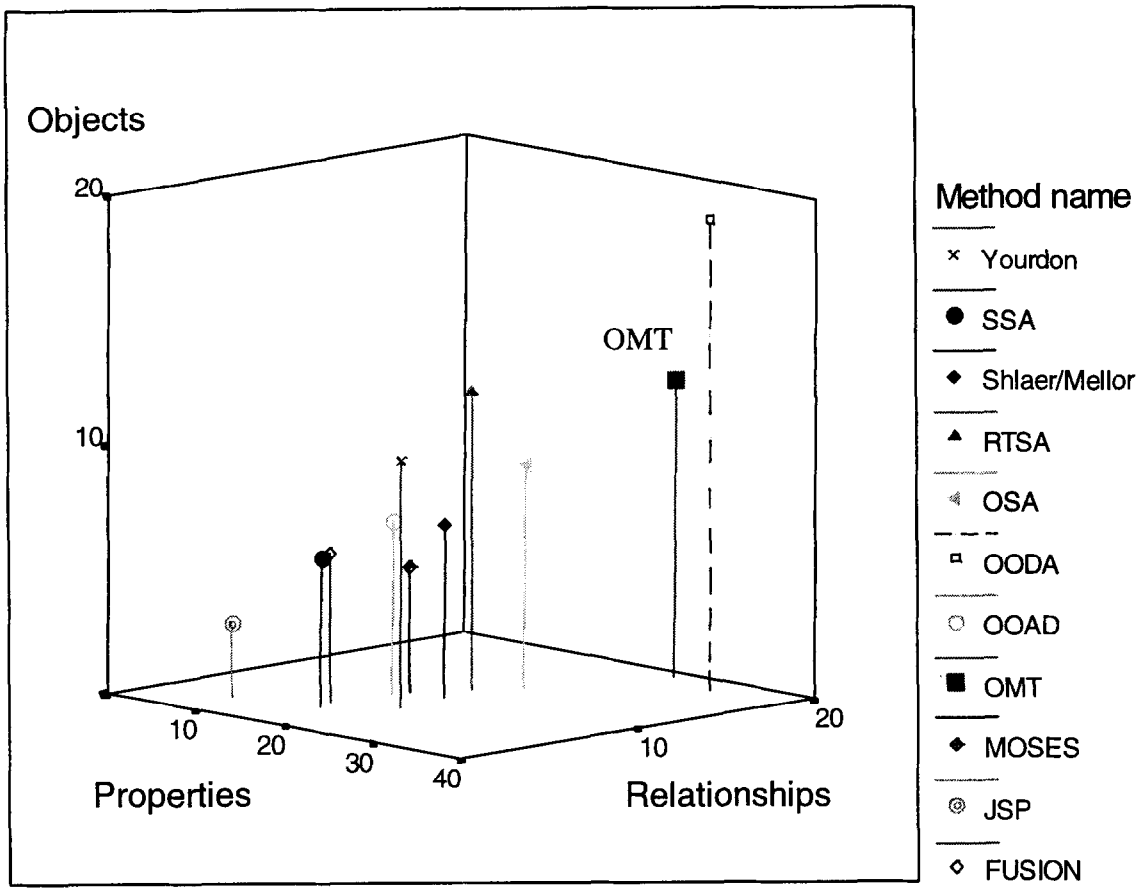

Fig. 4: Object-Relationship-Property cube for Methods

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. The metrics have been applied here only for OPRR-based models of methods, but their adaptation for ER-based models of methods in other CASE tools should be straightforward.

The metrics proposed here analyse only the conceptual part of the technique definitions based on the method models, and they should be accompanied by a set of metrics for the complexity of the models produced by applying the techniques. The analysis of application 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 application models, if the method's conceptual complexity does indeed lead to greater expressive power [31]. We believe that there is a balance between learnability and expressive power of a method, and that organizations selecting methods should be aware of the fact that more powerful methods may be harder to learn, whilst being more effective for experienced users. This balance could be empirically investigated by tests, such as the one in Batra et. al, where users with different experience in method use apply the method into an ISD task [3].

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

In the future we will have to consider integrated methods and derive metrics for them. In that work we will need a better understanding of the integration of techniques and how that complicates, or simplifies, the methods. We must also gather empirical data about the learnability of different techniques and their implementations, and about the use of different constructs in different techniques. This kind of research should be accompanied with studies about the possiblities to avoid constructs that are error prone and hard to apply as in [2].

The metrical comparison of conventional systems development methods against object-oriented methods [17] would be an intriguing further opportunity for additional research.

Acknowledgements — The authors would wish to thank Steven Kelly, Hui Liu and the other members of MetaPHOR team of the University of Jyväskylä, the Design Methodology Research Group of the University of Twente and the anonymous reviewers for their constructive remarks that led to improvement of the presentation of the metrics research results in the paper.

REFERENCES

[1] A.J. Albrecht, J.E. Gaffney. Software function, source lines of code, and development prediction: a software science validation. *IEEE Transactions on Software Engineering*, **9**(6): 639-647 (1983).

[2] D. Batra. A framework for studying human error behavior in conceptual database modeling. *Information & Management*, Vol. 25, pp.121-131 (1993).

[3] D. Batra, J. Hoffer and P. Bostrom. Comparing representations with relational and EER models. *CACM* **33**(2): 126-139 (1990).

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

[5] S. Brinkkemper. *Method Engineering: Engineering of Information Systems Development Methods and Tools*. Information and Software Technology, Forthcoming (1996).

[6] S. Brinkkemper, M. de Lange, R. Looman and F. H. G. C. van der Steen. On the derivation of method companionship by metamodelling. *ACM SIGSOFT Software Engineering Notes*, **15**(1): 49-58 (1990).

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

[8] John Cameron. *JSP & JSD: The Jackson Approach to Software Development*," IEEE Computer Society Press, Washington (1989).

[9] D. de Champeaux, D.D. Lea and P. Faure. *Object-Oriented Systems Development*. Addison-Wesley (1993).

[10] D. de Champeaux, P. Faure. A comparative study of object oriented analysis methods. *Journal of Object-Oriented Programming*, **5**(1): 21-33 (1992).

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

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

[13] D. Coleman, P. Arnold, S. Bodoff, C. Dollin, H. Gilchrist, F. Hayes and P. Jeremaes, *Object-Oriented Development The Fusion Method*. Prentice Hall, Englewood Cliffs (1994).

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

[15] D. Embley, D. Kurtz and S. Woodfield. *Object Oriented Systems Analysis, A Model-Driven Approach*. Yourdon Press, Prentice-Hall, Englewood Cliffs, NJ, USA (1992).

[16] N. Fenton. Software Measurement: A necessary scientific basis. *IEEE Transactions on Software Engineering* **20**(3):199-206 (1994).

[17] R. Fichman, C. Kemerer. Object-oriented and conventional analysis and design methods - comparison and critique. *IEEE Computer*, **25**(10): 22-39 (1992).

[18] C. Gane, T. Sarson. *Structured Systems Analysis: Tools and Techniques*. Prentice Hall, Englewood Cliffs, NJ (1979).

[19] G. Goldkuhl. Contextual activity modeling of information systems. In *Proceeding of the Third International Working Conference on Dynamic Modelling of*, pp. 125-145, Delft Technical university, Noordwijkerhout (1992).

[20] B. Henderson-Sellers, J. Edwards. *BookTwo of Object-Oriented Knowledge The Working Object*, Prentice Hall, Sydney (1994).

[21] S. Henry, D. Kafura. Software structure metrics based on information flow. *IEEE Transactions on Software Engineering* **7**(5): 510-518 (1981).

[22] M. Heym, H. Österle. Computer-aided methodology engineering. *Information and Software Technology* **35**(6/7): 345-354 (1993).

[23] S. Hong, 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*, pp. 689-698, IEEE Computer Science Press (1993).

[24] J. Iivari, P. Kerola. A sociocybernetic framework for the feature analysis of information systems development methodologies. In *Information Systems Methodologies: A Feature Analysis*, pp. 87-139, North-Holland, Amsterdam (1983).

[25] Iivari, Juhani. Object-oriented information systems analysis: A comparison of six object-oriented analysis methods. pp. 85--110 in *Methods and Associated Tools for the Information Systems Life Cycle (A-55)*, A. A. Verrijn-Stuart and T. W. Olle (Ed.), Elsevier Science B.V., North-Holland (1994).

[26] ISO. *EXPRESS Language Reference Manual.* Technical report ISO TC184/SC4/WG5, Document N14 (1991).

[27] S. Kelly, K. Smolander. Evolution and issues in metaCASE. *Information and Software Technology* (1995).

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

[29] 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).

[30] Karl J. Lieberherr, Ignacio Silva-Lepe and Cun Xiao. Adaptive object-oriented programming using graph-based customization. *CACM*, No.4: 94-101 (1994).

[31] J.L.H. Oei, 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).

[32] T.W. Olle, H.G. Sol and A.A. Verrijn-Stuart. *Proceedings of the IFIP WG 8.1 Working Conference on Comparative Review of Information Systems Design Methodologies.* North-Holland, Amsterdam (1982).

[33] T.W. Olle, H.G. Sol and A.A. Verrijn-Stuart. *Proceedings of the IFIP WG 8.1 Working Conference on Comparative Review of Information Systems Design Methodologies: Improving the Practice.* North-Holland, Amsterdam (1986).

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

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

[36] S. Shlaer, S.J. Mellor. *Object Lifecycles: Modelling the World in States.* Prentice Hall, Yourdon Press Computing Series, Englewood Cliffs, NJ, USA (1992).

[37] 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*, pp. 167-186, North-Holland, Amsterdam (1993).

[38] Kari Smolander, 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*, pp. 168-193, Trondheim, Norway, May, Springer-Verlag, Berlin (1991).

[39] Kari Smolander. OPRR: a model for modelling systems development methods. In *Next Generation CASE Tools*, pp. 224-239, IOS Press, Amsterdam, the Netherlands (1991).

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

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

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

[43] Daniel Teichroew, 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*, pp. 15-38, orth-Holland (1980).

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

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

[46] P. Ward, S. Mellor. *Structured Analysis for Real-Time Systems.* Prentice-Hall, New Jersey (1985).

[47] E. Yourdon. *Modern Structured Analysis.* Prentice-Hall, Englewood Cliffs, NJ, USA (1989).

APPENDIX 1. VALUES OBTAINED FROM 36 TECHNIQUES

Table 2 lists the values of 36 techniques modelled with 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 to by their functions as introduced in the main text. Thus the third column gives the number of object types, the fourth gives the relationship types and the fifth gives the number of properties of the technique. The sixth column gives the average number of properties per object type, the seventh the average number of properties per relationship type and the eighth column gives the average number of relationship types that can be connected to a given object type. The ninth column gives the quotient of the sums of the object's properties and object's relationships and their properties, and the tenth column gives the length of the "complexity vector" of the technique.

Method	**Technique**	$n(O_T)$	$n(R_T)$	$n(P_T)$	$\bar{P}_O(M_T)$	$\bar{P}_R(M_T)$	$\bar{R}_O(M_T)$	$\bar{C}(M_T)$	$C'(M_T$
Demeter [27]	Demeter	3	2	5	4	2	,50	1,33	6,16
Express [26]	EXPRESS-G	5	6	13	2,20	4	2,67	,14	15,17
FUSION [13]	Object Interaction Graph	2	1	7	3	1	3	,75	7,35
FUSION	Object Model	4	4	8	2	2,25	,75	,54	9,80
Goldkuhl [19]	Activity model	5	5	8	1,60	2,80	,20	,55	10,68
JSP [8]	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
MOSES [20]	Event Model	1	1	6	3	1	4	,60	6,16
MOSES	O/C Model	4	8	10	4,50	6	1,88	,67	13,42
OMT [35]	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
OOAD [12]	Object Oriented Analysis and Design	3	5	6	2,33	4	1	,37	8,37
OOAD	Object State Diagram	1	1	3	2	1	1	1	3,32
OOAD	Service Chart	3	2	7	2,67	2	,50	,89	7,87
OODA [4]	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
OSA [15]	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
RTSA [46]	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	State Transition Diagram	3	1	5	2	1	4	,40	5,92
RTSA	Structure Chart	1	1	4	2	1	2	,67	4,24
Shlaer/ Mellor [36]	Action Data Flow Diagram	2	4	7	2,50	3	1,75	,30	8,31
Shlaer/ Mellor	Information Structure Diagram	2	4	8	2,50	2	5	1,06	9,17
Shlaer/ Mellor	State Transition Diagram	3	1	5	2	1	4	,40	5,92
SSA [18]	Entity Relationship Attribute Diagram	3	2	6	2,33	1,67	,50	1,44	7
SSA	Structured Systems Analysis	3	2	10	3	2	2	,50	10,63
Yourdon [47]	Data Flow Diagram	3	2	6	1,67	2	1	,42	7
Yourdon	Entity Relationship Attribute Diagram	3	2	6	2,33	1,67	,50	1,44	7
Yourdon	State Transition Diagram	3	1	5	2	1	4	,40	5,92
Yourdon	Structure Chart	1	1	4	2	1	2	,67	4,24

Table 2: Values obtained from 36 techniques

APPENDIX 2. BOX-PLOTS FOR TECHNIQUES

In Figures 5 and 6 are the box-plots for techniques. The 5-point box-plots can be read as (from left to right): a bar representing minimum, a box starting from lower quartile, in the box there is the median bar and at the end of the box is the upper quartile: the fifth bar is the maximum. The outliers are indicated by a marker with the name of the technique (for example, * indicating Booch module diagrams). Notice that the scales of the various Figures containing box-plots differ.

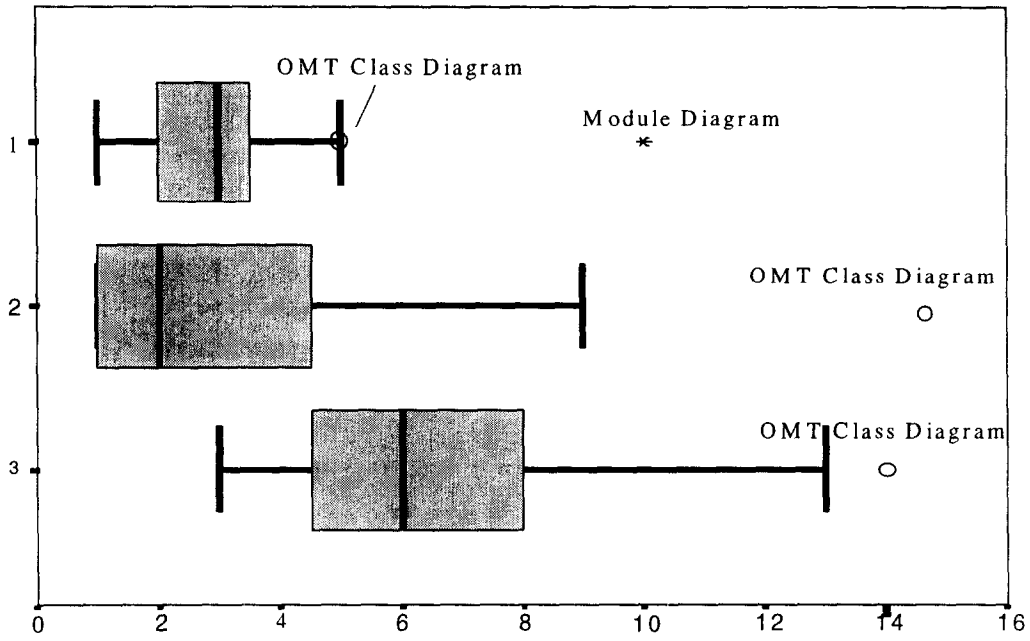

Fig. 5: Box-plots for formulae 1, 2 and 3 for techniques

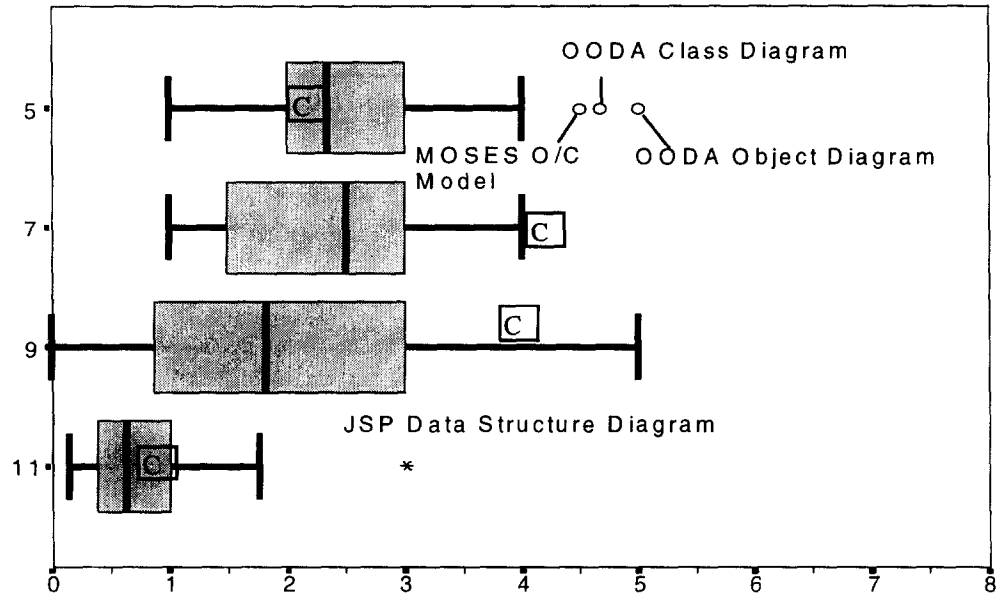

Fig. 6: Box-plots for formulae 5, 9, 7, 11 for techniques

APPENDIX 3. VALUES OBTAINED FROM 11 METHODS

Table 3 lists the values for 11 methods, as the aggregates of the values of the techniques listed in Table 2. The table shows the name of the method and the values obtained for the Formulae, which are referred to their functions as introduced in the main text. The second column gives the number of techniques of the method, the third column gives the total number of object types of the method, the fourth gives the number

of relationship types, and the fifth gives the number of the property types. The sixth column gives the summed complexity of the individual techniques of the methods and the last column gives the length of the "complexity vector" of the method.

Method	$n(\mathcal{M})$	$n(O_{\mathcal{M}})$	$n(R_{\mathcal{M}})$	$n(P_{\mathcal{M}})$	$\bar{C}(\mathcal{M})$	$C(\mathcal{M})$
JSP	2	3	3	8	2,17	9,06
FUSION	2	6	5	15	0,61	16,91
SSA	2	6	4	16	0,97	17,55
MOSES	2	5	9	16	0,66	19,03
OOAD	3	7	8	16	0,68	19,21
Shlaer/Mellor	3	7	9	20	0,56	23,02
OSA	3	9	13	21	0,56	26,29
OMT	3	12	19	26	0,59	34,37
Yourdon	4	10	6	21	0,75	24,02
RTSA	4	12	11	19	0,64	25,02
OODA	5	19	18	32	1,05	41,34

Table 3: Values obtained from 11 methods

APPENDIX 4. BOX-PLOTS FOR METHODS

This appendix contains the box-plots for methods in Figures 7 and 8. See Appendix 2 for explanations.

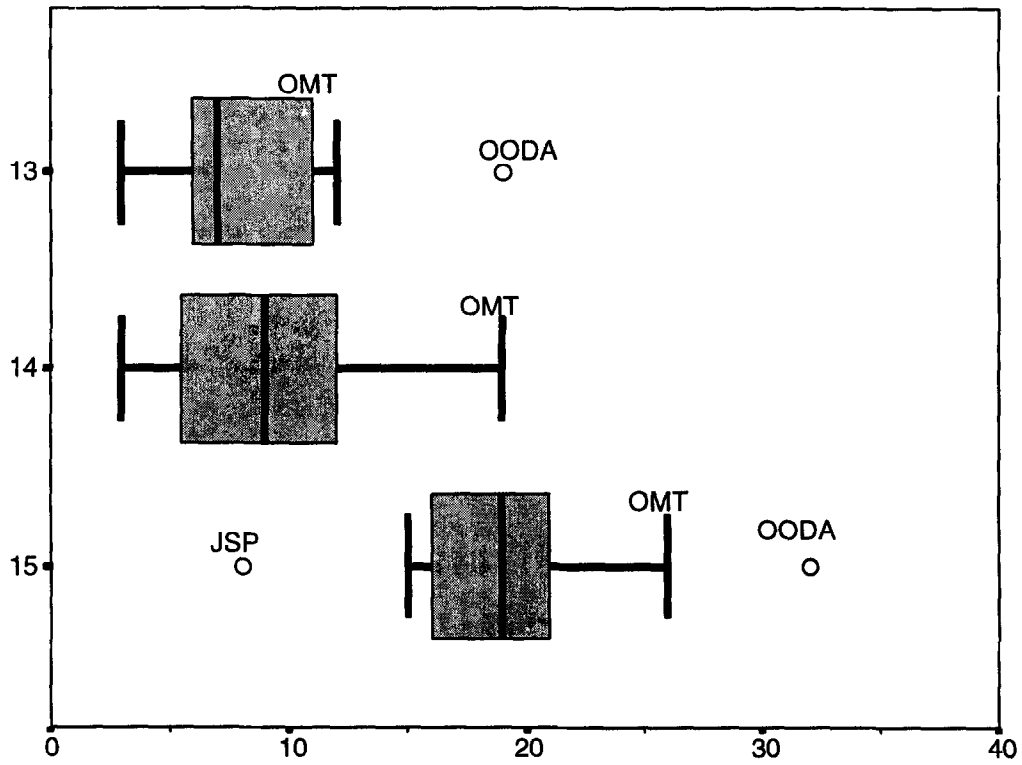

Fig. 7: Box-plots for formulae 13, 14 and 15 for methods

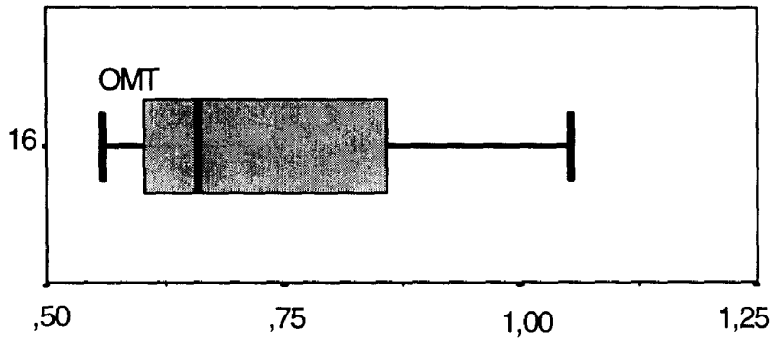

Fig. 8: Box-plot for formula 16 for methods