An Analytical Evaluation of Static Coupling Measures for Domain Object Classes

Geert Poels
Research Assistant of the Fund for Scientific Research – Flanders
Department of Applied Economic Sciences
Katholieke Universiteit Leuven
Naamsestraat 69 B-3000 Leuven, Belgium
geert.poels@econ.kuleuven.ac.be

ABSTRACT

We present a set of static coupling measures for domain object classes. Although these measures do not directly measure the external quality of an OO software system, they might be useful as early indicators of high-risk components of the system. In this paper we focus on the analytical evaluation of the measures using a well-known set of generic coupling properties.

1. INTRODUCTION

Whereas software users are primarily interested in quality attributes such as for instance defined in the ISO 9126 standard (i.e., functionality, reliability, efficiency, usability, maintainability, portability), these are external software product attributes that can only be evaluated with respect to how the software product relates to its environment. During software development software products are mostly isolated from the user environment. Therefore, most software quality models are based on the 'axiom' of software engineering [10, p. 151]:

$$\text{internal structure} \Rightarrow \text{external quality}$$

Internal structure attributes characterise software products used/produced in the early stages of software development. Moreover, these attributes can be measured directly. It is therefore common practice to use internal structure measures as early indicators for software quality.

One of the internal structure attributes related to software quality is coupling. This concept has been defined in the context of structured design methods as the degree of interdependence between modules [27, 28]. In the context of object-oriented software systems it refers to the degree of interdependence between object classes. According to Chidamber and Kemerer coupling between object classes must be controlled [9, p. 486]:

1. Excessive coupling between object classes is detrimental to modular design and prevents reuse. The more independent a class is, the easier it is to reuse it in another application.
2. In order to improve modularity and encapsulation, inter-object class couples should be kept to a minimum. The larger the number of couples, the higher the sensitivity to changes in other parts of the design, and therefore maintenance is more difficult.
3. A measure of coupling is useful to determine how complex the testing of various parts of a design are likely to be. The higher the inter-object class coupling, the more rigorous the testing needs to be.

Some of these arguments have been confirmed by empirical research. For instance, in [4] coupling between object classes, such as measured by Chidamber and Kemerer's CBO measure, was significantly related to the fault-proneness of the object classes. In [2] it was shown that Abreu's coupling factor (COF) showed a high positive correlation with defect density and normalised rework measures.

In an object-oriented context the number of variants of interdependency rises considerably in comparison to conventional systems developed with the structured paradigm [12]. Generally, coupling between object classes is due to the representation of associations between the classes (i.e., static coupling) and the message passing between instances of the classes (i.e., dynamic coupling) [1]. Dynamic coupling is the type of coupling that is more or less similar to the coupling between modules in a structured design. It can for instance be modelled using a call-graph model of a software system, where the calls represent messages. To model associations between object classes, most object-oriented analysis methods use some type of static model based on the concepts of the (extended) entity-relationship diagramming technique [8]. For instance, figure 1 shows generalisation/specialisation and aggregation relationships between object classes. Inside the object class specifications, these static coupling relationships are realised through respectively the inheritance (figure 2) and abstraction (figure 3) mechanisms [15].

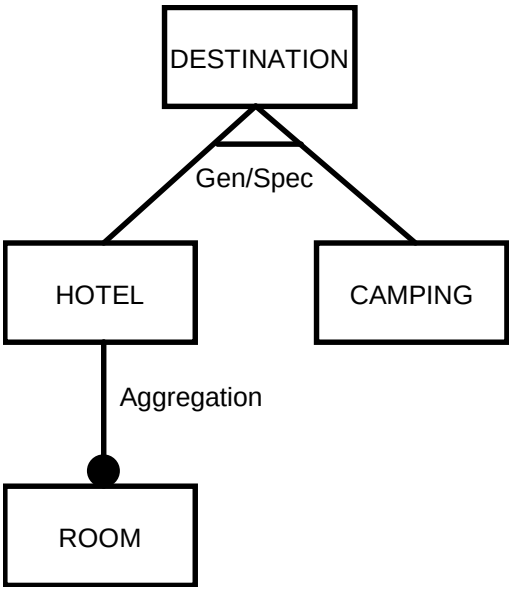

Figure 1: generalisation/specialisation and aggregation relationships between object classes

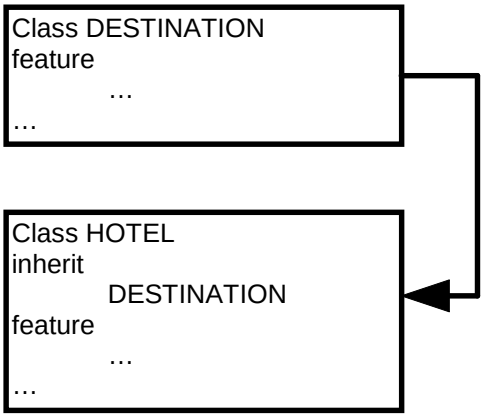

Figure 2: static coupling between object classes through the inheritance mechanism

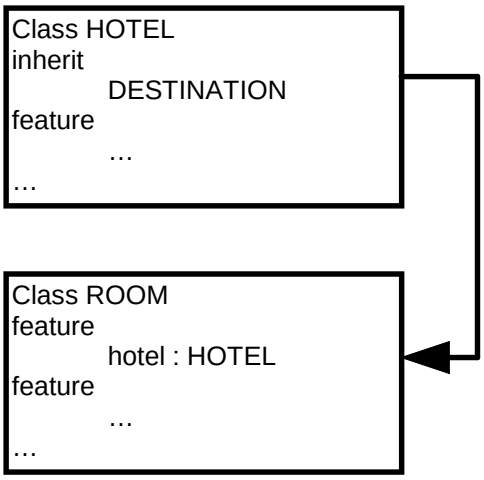

Figure 3: static coupling between object classes through the abstraction mechanism

According to Schach [23] static coupling, and especially inheritance coupling, is different from dynamic coupling, which he calls classical coupling. On the one hand, inheritance allows the internal reuse of object class features, which may lead to increased development productivity and hence lower development costs. On the other hand, inheritance coupling may result in change propagation effects that may have a negative effect on maintainability. It also has an effect on reusability as it is hard to reuse an object class without reusing its ancestor object classes. More important still, the inheritance mechanism implements generalisation/specialisation relationships between object classes, which are constrained by the domain being modelled. Static coupling is therefore a relevant structure attribute of OO analysis software products, implying that it can be measured very early in the development life cycle.

In this paper static coupling measures are proposed for OO analysis documents. More specifically we focus on the static coupling between domain object classes. The method we use to specify an OO domain model is MERODE [26]. This method combines formal specification techniques with the OO paradigm in order to specify and analyse the functional and technical requirements for an information system. A characteristic of MERODE is that it is model-driven. A clear distinction is made between functional requirements related to the application domain (i.e., business functionality), functional requirements related to user-requested information system functionality, and technical requirements. These three types of requirements are respectively specified in the business (or domain) model, the function model and the implementation model. The method is model-driven in the sense that starting from the domain model the other types of models are generated by incrementally specifying the other types of requirements[1]. We noticed that MERODE domain object classes are themselves not dynamically coupled, although they might be dynamically coupled to the function object classes of the function model. However, as MERODE supports the modelling of generalisation/ specialisation and existence dependency[2] relationships between domain object classes, there exist two types of static coupling between domain object classes. The first type is inheritance

[1] Note that this methodology is fundamentally different from the OOSE method of Jacobson [14] that is use-case driven (i.e., starts with modelling the information system functionality such as seen by the user).

[2] The existence dependency relation is a valuable alternative to the aggregation relation as its semantics is very precise and its use clear cut, in contrast with the concept of aggregation [26].

coupling. As a rule, a child object class inherits the features of its parent object class [25]. A change of the features of the parent object class is propagated into the child object class as the new defined features of the child may reference the features inherited from the parent. The second type of coupling is abstraction coupling. The relationship between an object class X and an object class Y, where X is existent dependent of Y, is established by declaring in X an instance variable v of type Y (see figure 3). The instance variable v may be referenced by the methods of X.

In section 2 a number of static coupling measures are defined for MERODE domain object classes. Next, in section 3 these measures are analysed using the well-known set of coupling properties of Briand, Morasca and Basili [6]. The main goal of this analytical evaluation is to check whether the measures correspond to the concept of coupling such as understood by these experienced software engineering researchers associated (or previously associated) with the Software Engineering Laboratory (SEL). If enough evidence of the concept validity of the proposed static coupling measures can be gathered, then a ready-to-use set of measures is at our disposal to conduct empirical research trying to establish relationships between static coupling in a domain model and software quality attributes.

2. STATIC COUPLING MEASURES

Published object class coupling measures such as CBO (i.e., the number of other classes to which a class is coupled) have been criticised for being insensitive to the strength of the coupling relationship between classes [3, 13]. The strength of coupling depends on two factors:

- the type of coupling relationship between the object classes;
- the number of coupling relationships (of the same or different type) between the object classes.

For the coupling between modules in a structured design context Fenton and Melton have proposed a measure that takes account of both factors [11, p. 184].

$$M(x, y) = i + n / (n + 1),$$

where i is a numerical ordinal ranking corresponding to the worst coupling type[3] between modules x and y, and n is the number of couples of type i between x and y. The function M can be used as an ordinal measurement scale for the strength of coupling between two modules. A proposal for an ordinal measurement scale of OO class coupling can be found in [12].

We believe that the effect of the type of coupling on the strength of coupling must be evaluated with respect to the impact that changes (or defects) in one class have on the other (coupled) class. Schach and Binkley [24] use the same criterion in their construction of the coupling dependency measure CDM when evaluating different types of coupling. In case of static coupling between MERODE domain object classes, we are inclined to believe that inheritance coupling leads to more change propagation effects than abstraction coupling. However, this result is probably method-specific and cannot be generalised. For instance, the ordinal measurement scale for class coupling of Hitz and Montazeri [12] does not differentiate between a class Y being a parent class of a class X (i.e., inheritance) and Y being the type of an instance variable of class X (i.e., abstraction). We therefore choose to measure both types of static coupling using separate measures.

The static coupling between an object class and its environment (i.e., the other object classes in the model) may be evaluated from an inbound/importing or an outbound/exporting perspective [6, 16]. Whereas the former refers to the incoming static coupling relationships, the latter type of static coupling refers to the outgoing relationships. To avoid the 'double counting' of coupling relationships (from a system point of view), a measure of class coupling must be defined from a specific perspective. We therefore define four measures of static class coupling:

- inbound inheritance coupling (IIC)
- outbound inheritance coupling (OIC)
- inbound abstraction coupling (IAC)
- outbound abstraction coupling (OAC)

With respect to the second factor of coupling strength (i.e., the number of couples) we must note that MERODE defines the generalisation/ specialisation relation as a binary relation on the set of domain object classes. Hence, between two object classes there is at most one inheritance coupling relationship. Moreover, the generalisation/specialisation relation is a hierarchy. Therefore, only single inheritance is allowed. On the other hand, the existence dependency relation is defined as a bag on the set of object classes. A class X may therefore be more than once existent dependent of a class Y, and the (potential) number of abstraction couplings between X and Y is in principle unlimited. The measures IAC and OAC must take account of this property of the existence dependency relation. As it is allowed by MERODE that domain object classes are not coupled at all

[3] Coupling types are ranked on a 6 point ordinal scale from loosely coupled (i = 1) to tightly coupled (i = 5). If there is no coupling between x and y then i = 0.

(although very unlikely), we use a hypothetical domain object class that is not involved in any generalisation/specialisation or existence dependency relationship as our reference point for the coupling measurement scales.

Tentative definitions for the four static coupling measures are:
For a domain object class X in a domain model DM

- IIC(X) = 1 if X inherits from another object class in DM
  = 0 otherwise
- OIC(X) = the count of other object classes in DM that inherit from X
- IAC(X) = the count of instance variables of X that have another class in DM as type[4]
- OAC(X) = the count of instance variables of type X that have been declared in other classes of DM

The measure definitions are tentative in the sense that only direct coupling relationships are taken into account. Also, the measures are defined at the object class level of granularity. For instance, IIC is unrelated to the number of features that is inherited from the parent object class. Such a measure might be acceptable as a measure of the extent of internal reuse (through inheritance).

Static coupling measures are also defined at the system level of granularity. The measures IC and AC are used respectively to measure the extent of inheritance coupling and the extent of abstraction coupling in a domain model:
For a domain model DM

- IC(DM) = the count of inheritance relationships in DM
- AC(DM) = the count of abstraction relationships in DM

It holds that

$$IC(DM) = \sum_{X \in DM} IIC(X) = \sum_{X \in DM} OIC(X)$$

and

$$AC(M) = \sum_{X \in DM} IAC(X) = \sum_{X \in DM} OAC(X)$$

3. ANALYTICAL EVALUATION

We check whether our static coupling measures satisfy the coupling properties published by Briand, Morasca and Basili [6]. Measure properties must be regarded as desirable properties for software measures. They formalise the concept being measured such as intuitively understood by the developers of the property set. However, most sets of measure properties consist of necessary, but not sufficient properties [17, 22]. Briand et al. repeat several times that their measure properties are not sufficient, for instance “by no means should these properties be regarded as the unique set of properties that can be possibly defined for a given concept” [6, p. 69] and “we consider these properties necessary but not sufficient because they do not guarantee that the measures for which they hold are useful or even make sense” [6, p. 71]. As a consequence, measure properties are useful to invalidate proposed software measures, but they cannot be used to formally validate them. In other words we wish to check whether the static coupling measures do not contradict a minimal set of subjective, but experienced, viewpoints on the concept of coupling. However, showing the invariance of the coupling properties for our measures is not the same as formally proving that they are valid.

The coupling properties of Briand et al. were proposed in [6] and further refined in [7] and [19]. These coupling and other concept (i.e., size, length, complexity, cohesion) properties have been used to analytically assess high-level design measures [5], measures for Petri Net-based specifications of concurrent software [18], and measures for object-relational databases [20].

The coupling properties are defined for a generic representation of a modular software system.

- A **system** S is a tuple <E, R>, where E is a set of elements and $R \subseteq E \times E$ is a set of relationships between the elements of E.
- A **modular system** MS is a triple <E, R, M>, where <E, R> is a system and M is a collection of modules that partitions E.
- A system $m = <E_m, R_m>$ is a **module** of MS if and only if $E_m \subseteq E$ and $R_m \subseteq R$.
- The modules of M are **disjoint**, i.e., for modules $m_1 = <E_{m1}, R_{m1}>$ and $m_2 = <E_{m2}, R_{m2}>$ of MS it must hold that $E_{m1} \cap E_{m2} = \emptyset$ (and hence $R_{m1} \cap R_{m2} = \emptyset$).
- The **inter-modular relationships** that a module $m = <E_m, R_m>$ of MS is involved in are captured in the sets InputR(m) and OutputR(m):
  $InputR(m) = \{<e_1,e_2> \in R \mid e_2 \in E_m \text{ and } e_1 \in E - E_m\}$;
  $OutputR(m) = \{<e_1,e_2> \in R \mid e_1 \in E_m \text{ and } e_2 \in E - E_m\}$.
  OuterR(m) denotes either InputR(m) or OutputR(m) and is used to reflect the perspective (inbound/outbound) of the coupling measure.
- The set of all **intra-modular relationships** in MS is $IR = \bigcup_{\text{m is a module of MS}} R_m$.

[4] If there are n existence dependency relationships between X and Y, then X will have n instance variables of type Y.

- The set of all **inter-modular relationships** in MS is R – IR.

The coupling properties are presented below. Note that, such as proposed by us in [21], the naming of property Coupling.V has been changed from Disjoint Module Additivity to Non-connected Module Additivity as the premise of the property is only satisfied for **non-connected** modules (i.e., having no relationships between them).

Property Coupling.I: **Nonnegativity**. Given a module $m = <E_m, R_m>$ of a modular system $MS = <E, R, M>$

$$[Coupling(m) \geq 0 \mid Coupling(MS) \geq 0]$$

Property Coupling.II: **Null Value**. Given a module $m = <E_m, R_m>$ of a modular system $MS = <E, R, M>$. When OuterR(m) is empty, the coupling of m is null. When R – IR is empty (i.e., no inter-modular relationships), the coupling of MS is null.

$$[OuterR(m) = \varnothing \Rightarrow Coupling(m) = 0 \mid R - IR = \varnothing \Rightarrow Coupling(MS) = 0]$$

Property Coupling.III: **Monotonicity**. Adding inter-modular relationships cannot decrease coupling.

Let $MS' = <E, R', M'>$ and $MS'' = <E, R'', M''>$ be two modular systems (with the same set of elements E) such that there exist two modules $m' \in M'$ and $m'' \in M''$ such that $R' - OuterR(m') = R'' - OuterR(m'')$, and $OuterR(m') \subseteq OuterR(m'')$. Then

$$[Coupling(m') \leq Coupling(m'') \mid Coupling(MS') \leq Coupling(MS'')]$$

Property Coupling.IV: **Merging of Modules**. When merging two modules m'_1 and m'_2 into a new module m", all the inter-modular relationships between m'_1 and m'_2 become intra-modular relationships of m". Therefore, the coupling of m" cannot be larger than the sum of the couplings of m'_1 and m'_2.

Let $MS' = <E, R, M'>$ and $MS'' = <E, R, M''>$ be two modular systems (with the same underlying system $<E, R>$) such that $M'' = M' - \{m'_1, m'_2\} \cup \{m''\}$, with $m'_1 \in M'$, $m'_2 \in M'$, $m'' \notin M'$ and $E_{m''} = E_{m'1} \cup E_{m'2}$ and $R_{m''} = R_{m'1} \cup R_{m'2} \cup \{<e_1,e_2> \in R \mid (e_1 \in E_{m'1}$ and $e_2 \in E_{m'2})$ or $(e_1 \in E_{m'2}$ and $e_2 \in E_{m'1})\}$. Then,

$$[Coupling(m'_1) + Coupling(m'_2) \geq Coupling(m'') \mid Coupling(MS') \geq Coupling(MS'')]$$

Property Coupling.V: **Non-connected Module Additivity**. The coupling of non-connected modules is additive since there are no inter-modular relationships that become intra-modular relationships.

Let $MS' = <E, R, M'>$ and $MS'' = <E, R, M''>$ be two modular systems (with the same underlying system $<E, R>$) such that $M'' = M' - \{m'_1, m'_2\} \cup \{m''\}$, with $m'_1 \in M'$, $m'_2 \in M'$, $m'' \notin M'$ and $m'' = m'_1 \cup m'_2$. Then,

$$[Coupling(m'_1) + Coupling(m'_2) = Coupling(m'') \mid Coupling(MS') = Coupling(MS'')]$$

In order to analytically evaluate the static coupling measures the MERODE domain model representation must be mapped into the generic system representation of [6]. The domain object classes and the domain model correspond to respectively the system elements and the modular software system. When evaluating the inheritance coupling measures the inheritance relationships map into the relationships between the system elements. When evaluating the abstraction coupling measures the abstraction relationships map into the system relationships. As we are interested in the coupling of object classes to the rest of the system, object classes also correspond to the modules of the system. Hence, each module contains a single element of the system and all relationships are inter-modular relationships (i.e., IR = ∅). In case of abstraction relationships the set R is defined such that it is a bag on E. We assume that this modification does not change the meaning of the measure properties.

It is easy to verify that all our static coupling measures satisfy the above coupling properties. The 'prototype' coupling measure underlying the coupling properties is the count of relationships in OuterR(m) for the module m and the count of relationships in R – IR for the system. Our static coupling measures are in this regard 'prototype' coupling measures.

4. CONCLUSIONS

In this paper a set of coarse-grained static coupling measures was proposed for domain object classes. The measures are constructed such that they measure the strength of the couplings between object classes, and not merely the number of classes that are coupled to the class of interest (such as done by CBO). The measures are analytically evaluated using a well-known set of coupling properties. The conformance to these measure properties does not formally validate our proposed measures, but shows that they do not contradict popular and substantiated beliefs regarding the concept of coupling.

In future research we plan to investigate the meaningfulness of measuring indirect coupling relationships. We also would like to define more fine-grained measures by focusing on the features of object classes that are affected by change or defect propagation due to static coupling. Further, we wish to extend the focus of our research to the function model, introducing additional (dynamic) types of coupling. We also plan to do a more thorough analytical evaluation of the coupling measures based on our distance-based approach to software measurement [22]. Eventually, we hope to have gathered convincing evidence on the concept validity of our coupling measures in order to start with the really interesting part of our research, i.e., empirical research on the relationship between the characteristics of software specifications and the characteristics of software implementations and processes. More specifically we wish to investigate whether static coupling measurements of a domain model are useful as early indicators of high-risk system components.

ACKNOWLEDGMENTS

This work was supported by a doctoral research grant from the Fund for Scientific Research – Flanders.

REFERENCES

[1] F.B. Abreu, R. Esteves and M. Goulao, 'The Design of Eiffel Programs: Quantitative Evaluation Using the MOOD Metrics', *Proceedings of TOOLS'96*, Santa Barbara, Ca., USA, July 1996.

[2] F.B. Abreu and W. Melo, 'Evaluating the Impact of Object-Oriented Design on Software Quality', *Proceedings of the IEEE 3rd International Software Metrics Symposium (METRICS'96)*, Berlin, Germany, March 1996.

[3] L. Badri, M. Badri and S. Ferdenache, 'Towards Quality Control Metrics for Object-Oriented Systems Analysis', *Proceedings TOOLS EUROPE '95*, 1995, pp. 193-206.

[4] V.R. Basili, L.C. Briand and W.L. Melo, 'A Validation of Object-Oriented Design Metrics as Quality Indicators', *IEEE Transactions on Software Engineering*, Vol. 22, No. 10, October 1996, pp. 751-761.

[5] L.C. Briand, S. Morasca and V.R. Basili, 'Defining and Validating High-Level Design Metrics', *Technical Report CS-TR-3301, Version 2*, University of Maryland, USA, 1994.

[6] L.C. Briand, S. Morasca and V.R. Basili, 'Property-Based Software Engineering Measurement', *IEEE Transactions on Software Engineering*, Vol. 22, No. 1, January 1996, pp. 68-86.

[7] L.C. Briand, S. Morasca and V.R. Basili, 'Response to: Comments on "Property-Based Software Engineering Measurement": Refining the Additivity Properties', *IEEE Transactions on Software Engineering*, Vol. 23, No. 3, March 1997, pp. 196-197.

[8] P.P-S. Chen, 'The Entity Relationship Model - Towards a Unified View of Data', *ACM Transactions on Database Systems*, Vol. 1, No. 1, March 1976, pp. 9-36.

[9] S.R. Chidamber and C.F. Kemerer, 'A Metrics Suite for Object Oriented Design', *IEEE Transactions on Software Engineering*, Vol. 20, No. 6, June 1994, pp. 476-493.

[10] N.E. Fenton, *Software Metrics: A Rigorous Approach*, Chapman & Hall, 1991.

[11] N. Fenton and A. Melton, 'Deriving Structurally Based Software Measures', *Journal of Systems and Software*, Vol. 12, 1990, pp. 177-187.

[12] M. Hitz and B. Montazeri, 'Measuring Coupling in Object-Oriented Systems', *Object Currents*, April 1996.

[13] M. Hitz and B. Montazeri, 'Chidamber and Kemerer's Metrics Suite: A Measurement Theory Perspective', *IEEE Transactions on Software Engineering*, Vol. 22, No. 4, April 1996, pp. 267-271.

[14] I. Jacobson et al., *Object-Oriented Software Engineering, A Use Case Driven Approach*, Addison-Wesley, 1992.

[15] W. Li and S. Henry, 'Object-Oriented Metrics that Predict Maintainability', *Journal of Systems and Software*, Vol. 23, 1993, pp. 111-122.

[16] H. Lounis, H.A. Sahraoui and W.L. Melo, 'Defining, Measuring and Using Coupling metrics in OO Environment', position paper presented at *OOPSLA'97 workshop on Object-Oriented Product Metrics*, Atlanta, USA, October 1997.

[17] B.A. Kitchenham and J.G. Stell, 'The danger of using axioms in software metrics', *IEE Proceedings on Software Engineering*, Vol. 144, No. 5-6, October-December 1997, pp. 279-285.

[18] S. Morasca, 'Defining Measures for Petri Net-based Specifications of Concurrent Software', *Proceedings of the Annual Oregon Workshop on Software Metrics (AOWSM'97)*, Coeur d'Alene, Id., USA, May 1997.

[19] S. Morasca and L.C. Briand, 'Towards a Theoretical Framework for Measuring Software Attributes', *Proceedings of the IEEE 4th International Software Metrics Symposium (METRICS'97)*, Albuquerque, NM, USA, November 1997.

[20] M. Piattini, C. Calero, M. Polo and F. Ruiz, 'Maintainability in Object-Relational Databases', *Proceedings of the 1st European Software Measurement Conference (FESMA'98)*, Antwerp, Belgium, May 1998, pp. 223-230.

[21] G. Poels and G. Dedene, 'Comments on "Property-Based Software Engineering

Measurement": Refining the Additivity Properties', *IEEE Transactions on Software Engineering*, Vol. 23, No. 3, March 1997, pp. 190-195.
[22] G. Poels and G. Dedene, 'Distance-Based Software Measurement: Necessary and Sufficient Properties for Software Measures', paper submitted to *IEEE 5th International Software Metrics Symposium (METRICS'98)*.
[23] S.R. Schach, 'The cohesion and coupling of objects', *Journal of Object-Oriented Programming*, Vol. 8, No. 8, January 1996, pp. 48-50.
[24] S.R. Schach and A.B. Binkley, 'CDM: A Successful
Object-Oriented Coupling Metric', position paper presented at *OOPSLA'97 workshop on Object-Oriented Product Metrics*, Atlanta, USA, October 1997.
[25] M. Snoeck and G. Dedene, 'Generalisation/ Specialisation and Role in Object Oriented Conceptual Modeling', *Data and Knowledge Engineering*, Vol. 19, No. 2, June 1996, pp. 171-195.
[26] M. Snoeck and G. Dedene, 'Existence Dependency: The key to semantic integrity between structural and behavioural aspects of object types', *IEEE Transactions on Software Engineering*, Vol. 24, No. 4, April 1998.
[27] W.P. Stevens, G.J. Myers and L.L. Constantine, 'Structured Design', *IBM Systems Journal*, Vol. 13, No. 2, 1974, pp. 115-139.
[28] E. Yourdon and L.L. Constantine, *Structured Design*, Prentice-Hall, 1979.