On the Notion of Semantic Metric Spaces for Object and Aspect Oriented Software Design

Epaminondas Kapetanios and Sue Black

University of Westminster, London, UK
{e.kapetanios, blacksu}@wmin.ac.uk
http://cisse.wmin.ac.uk

Abstract. Quality assurance via metrics and quality models in conceptual modelling and design for software and data or knowledge bases has always been of a major concern for software, systems and database developers. Given the inherent difficulty of suggesting as objective as possible design metrics, this paper discusses a theoretical framework for software design metrics, which conceives parameters as dimensions of a metric space. To this extent, it provides a bridge between techniques for similarity measurement in the field of information retrieval and software design. The introduced metric space is conceived as a vector space, which enables comparisons among proposed software development alternatives. This is illustrated in terms of metric parameters for aspect-oriented software system design and its impact on the object-oriented counterpart. However, the theoretical framework and discussion could also be considered as a design quality metrics framework for alternative conceptualizations as applied to object-oriented software design and its persistent storage counterpart via object-oriented databases.

1 Introduction

Software measurement has been around now for some forty years [Zus98]. It has been used to gauge software and software engineering quality in terms of the inherent products, processes and resources. Because we do not always know what it is that causes a project or its software to be of poor quality it is essential that we record and examine trends and characteristics via measurement. Early measurement of software focused almost entirely on source code with the simplest measure being lines of code (LOC). In 1983, Basili and Hutchens [BH83] suggested that LOC be used as a baseline or benchmark to which all other measures be compared i.e. an effective metric should perform better than LOC so LOC should be used as a "null hypothesis" for empirical evaluation.

Much empirical work has shown it to correlate with other metrics [She93] most notably with McCabe's Cyclomatic Complexity. The earliest code metric based on a coherent model of software complexity was Halstead's software science []. Early empirical evaluations produced high correlations between predicted and actual results but later work showed a lower correlation. Bowen [Bow78] found only modest correlation, with software science being outperformed by LOC.

According to Shepperd [She93] the most important legacy of software science is that it attempts to provide a coherent and explicit model of program complexity as a framework within which to measure and make interpretations regarding the quality of software artefacts.

With an increasing complexity in information and software systems design as well as the emergence of new software design and development paradigms, the focus of software measurement widened to include measurement during the earlier stages of the software development lifecycle, not only at code level. Design level metrics can in theory be obtained much earlier in the development of a project thus providing information which can be used for many purposes including more informed resource management.

Providing a methodological framework, however, for quality assurance of non-functional requirements such as design quality and reflecting user concerns within software artefacts requires a different approach to measurement and metrics. *Semantic metrics* based on design and structural properties as well as contents are being currently discussed and introduced, which help in measuring cohesion and coherence of software artefacts. For instance, *ontology metrics*, i.e., metrics calculated on the ontologies that form part of an ontology-based software system, are also being discussed. Currently ontology metrics are calculated using techniques similar to semantic metrics, but other semantically based expansions, including some similar to conceptual metrics (Latent Semantic Indexing based), are possible.

With the advent of the aspect-oriented software development (AOSD) and programming (AOP) as a new paradigm in software engineering, things become more challenging. Despite the fact that AOP is meant to enable developers to capture *crosscutting* concerns such as debugging or logging functionality, which cannot be captured cleanly by Object-Oriented (OO) software, the key question is how we can quantify that applying AOSD/AOP is beneficial and indeed more effective than the OO counterpart. In other words, we need to quantify to which extent design compromises lead to tangled and scattered code within a software engineering structure, since it is often difficult to arrive at a design that modularises all the required system features when designing and building systems.

Despite the fact that several studies have shown that AOSD/AOP is effective at improving the modularity of an existing object-oriented (OO) software system [LL00,CK03], there is a lack of a robust theoretical framework for AOSD/AOP design metrics. In our approach, we conceive AO as part of the overall system design process and not merely as an extension of OO system development. To this extent, the introduced abstraction of *Concerns* could be seen as part of the system conceptualisation meeting the different aspects of a software system design. Moreover, we aspire at providing a framework for the evaluation of the impact of aspect-oriented system development on the design objectives of the existing system or the one currently being designed such that certain kinds of concerns or non-functional requirements are also taken into consideration. The proposed framework will also enable us to provide answers as to which extent

AOSD/AOP is of good quality and beneficial in comparison with the existing object-oriented software counterpart.

Similar to the well established design and code metrics introduced by Chidamber and Kemerer in [CK94], our proposed methodology deals with both design and code metrics, however, tailored for the nature of AO/OO software rather than solely OO software. Another distinguishing issue is the consideration of the metrics as dimensions of a metric space. A proposed AOSD is conceived as being a cloud of points standing for *aspects* located within the metric space according to the various metrics at syntactic and semantic level. Some of the metric parameters are firstly introduced with this paper in order to express more adequate AOSD-to-OOSD metrics. To this extent, it is possible to compare proposed aspect-oriented software designs by calculating their position within the metric space and the distance from each other as well as for the OO counterpart. The closer the AOSD lies to the centre of the multi-dimensional metric space and away from its boundary, the higher the necessity and the more sense it makes to apply the proposed AOSD. The suggested multi-dimensional metric space has its counterpart in mathematics in a Euclidean n-sphere.

2 Related Work and Overview

Most of the evaluation techniques for aspect-oriented programs and designs are based on a number of studies such as [LL00,CK03] showing that AO is significantly more effective than OO for system development. The formats of these studies involve an initial development using OO and a subsequent development of the same system in AO. To this extent, they do not address evaluation of system development where AO and OO are both integral parts of the system development process.

This is due to the fact that evaluations are based on code-driven metrics. For instance, suggestions for aspect-oriented software metrics such as [TCB04] rely on their counterparts from well established object-oriented software metrics such as the C&M metrics [CK94] suite as the best validated set of measures [HCN98]. *Weighted Methods per Class*, *Depth of Inheritance Tree*, *Number of Children*, etc., are being adapted for aspects to be considered as classes. Estimations of the impact on system properties such as *undestandability*, *maintainability*, *reusability*, *testability* are made on the basis of increase or reductions of the metrics, which are key metrics to particular system properties, prior and after the use of aspects.

Following this path, it is hard to evaluate the impact of aspects oriented system development on the design objectives of the existing system or the one currently being designed such that certain kinds of concerns or non-functional requirements are also taken into consideration. A distinction between design and code metrics has already been addressed by Chidamber and Kemerer in [CK94] because of the abstractions introduced in OO. Abstractions such as *concerns* introduced in AO, however, are not taken into consideration by the suggested metrics suite for AO system development.

To this extent, it has also been stated that building on general software metrics besides defining completely new metrics is a key issue. Given that AOSD introduces new abstractions and new composition techniques, most existing metrics cannot be applied straight forwardly to aspect-oriented software. For instance, new kinds of *coupling* and *cohesion* are needed. In the literature, one can find design metrics for AO such as *separation of concerns metrics* [Lop97] and *coupling* or *cohesion metrics* either entirely tailored to AO [Zha02,ZX04] or extending existing metrics such as *lack-of-cohesion* [SGC$^+$03]. All attempted proposals for metrics, however, are bound with validation scenarios as provided by validation frameworks such as [KPF95].

Given that we are particularly interested in augmenting the metric space with design metrics for AO system development, related work from other research areas such as ontology or ontology-based metrics need to be drawn into consideration. As pointed out in the introduction, we conceive AO as part of the overall system design process and not merely an extension of OO system development. To this extent, the introduced abstractions of *Concerns* could be seen as part of the system conceptualisation. Suites of metrics to assess the quality of ontologies as specifications of conceptualisations have already been suggested in the literature.

For instance, [BJSSA05] proposed a suite of metrics comprising ten metrics as derived from a theory of semiotics [SLHA00]. These metrics address syntactic, semantic, pragmatic and social quality of an ontology. Attributes such as interpretability, consistency, clarity, are adhered to semantic quality, whereas comprehensiveness, accuracy, relevance, are adhered to the pragmatic level. Conceptual modelling quality has also been addressed by [CK94,WW95]. In knowledge engineering, formal ontologies have been used to evaluate the quality of *is-a* relationships [GW02]. Furthermore, ontology cohesion and coupling as they might affect a software system that employed those ontologies has been the intention of the work in [OYE06,YOE05].

Though these suites of metrics lack a validation framework as normally found in software engineering, their major drawback is another one. They can hardly apply to AO system design due to the intertwined character of AO software with OO software systems. Any proposed suite of design metrics for AO software needs to be tailored to the nature of AO software design and programming. This is the main reason we embarked on a suite of AO metrics and a corresponding metric space, which embraces both AO design and code metrics. The metric space will enable a more comprehensive estimation of the quality of an AO/OO system design as well as a comparison framework for proposed system designs.

3 The Theoretical Framework

3.1 Mathematical Background

The question of how to measure similarity and distance as well as how to represent the idea of relevance among words and concepts has led to the definition of space models and the structure of such spaces, together with inherent metrics and logic. Answers to this question have long been sought in order to provide appropriate solutions to problems in Information Retrieval (IR), where set–theoretical approaches fell short to cope with "relatedness" and "aboutness" [vR04], as well as to problems in Computational Linguistics and Knowledge Discovery and Text Mining, where word meaning disambiguation and clustering pose a challenge.

High dimensional mathematical spaces such as graphs and hierarchies have always been introduced in order to measure distances and similarities. The most standard distance measures in mathematics are called metrics, which must satisfy certain conditions or axioms such as being symmetric and transitive. For example, if it holds that A is the same as B and B is the same as C, then it follows that A is the same as C, according to the *transitivity* axiom. Similarly, if it holds that A is close to B, then, according to the *symmetry* axiom, B is close to A too.

A very important similarity measure for points in the plane and in higher dimensions is the *cosine measure.* The cosine measure assigns a high similarity to points that are in the same direction from the origin, zero similarity to points that are perpendicular to one another, and negative similarity for those that are pointing in opposing directions to one another.

It has been a matter of concern, however, that these mathematical techniques are actually inappropriate when we are dealing with natural language and measure of distance and similarity among words or concepts. Some of the properties of metrics are not always ideal for describing distances between words and concepts. For example, the axioms of symmetry and transitivity do not hold. It is more obvious to say that *my car is close to Regent's park* instead of also saying that *Regent's park is close to my car*, if the axiom of symmetry would have been valid. Moreover, saying that *the handle is part of the door and the door part of the apartment* does not necessarily follow that *the handle is part of the apartment.*

Even when we put words and concepts on graphs and taxonomies, we should be able to measure similarity or distance by finding the lengths of paths among concepts or words. However, there are at least three problems with this. First of all, finding shortest paths is often computationally expensive and may take a long time. Secondly, we may not have a reliable taxonomy. The fact that there is a short path between two words in a graph does not necessarily mean that they are very similar, because the links in this short path may have arisen from very different contexts. Thirdly, the meanings of words we encounter in documents and corpora may be very different from those given by a general taxonomy such as WordNet. For example, WordNet 2.0 only gives the fruit and tree meanings for the word apple, which is a decided contrast with the top 10 pages currently returned by the Google search engine when doing an Internet search with the query apple, which are all about *Apple* Computers.

Going beyond individual words or concepts, another limitation of current methods and metrics so far is that they have focused purely on individual concepts, mainly single words. Ideally, we should be able to find the similarity between two *collections of words*. Therefore, we need some process for semantic composition, i.e., working out how to represent the meanings of a sentence or document based on the meaning of the words it contains. It might also happen that words belong to more than one collection, since they are ambiguous. These words can sometimes behave like semantic worm holes, accidentally transporting us from one area of meaning to another. But we can also use this effect positively, because finding those words which have this strange wormhole effect when measuring distances helps us to recognise which words are ambiguous in the first place.

The technique of using vectors to represent points in space is a rather old one. Those interested in the mathematical development of vectors could start with the pioneering work of Descartes (1637). This addresses the idea of describing points by measuring their coordinates on two chosen lines. A good historical account can be found in [BM91]. Hermann Grassmann (1809–1877), however, with his work on *extension theory* (Ausdehnungslehre) [Gra62] lays the foundation of the modern notion of vectors in *any* number of dimensions.

It is not surprisingly that vector spaces have been introduced as a space model for *words*. A contemporary typical representative of such a vector space is the *Infomap* vector space model [Wid04], or WORDSPACE, as pioneered by Heinrich Schütze [Sch98]. The model works by mapping words to points in a high dimensional space, by recording the frequency of co-occurrence between words in the text, for example, the number of times two words appear in the same document. Each document might be conceived as a dimension.

The distribution of co-occurrences between a word and some set of content bearing terms then serves as a profile of the word's usage and can associate words with similar meanings. A word can thus be described by a spectrum of related words. In the context of search engines, comparing the query words' profiles to profiles generated for each document, articles which are supposed to be conceptually related to the query words, are returned, even if the words themselves do not appear in the text.

To this extent, a *Logic with Vectors* as well as *Vector Negation* has also been introduced in WORDSPACE as an operator in order to cope with word disambiguation by manipulating vectors in WORDSPACE so that they are perpendicular or orthogonal to one another. Moving the vectors so that they are at right angles to one another effectively makes the meanings of two word vectors irrelevant to one another. AND, OR, and NOT operators can all be defined and built in a vector space such as WORDSPACE, and these are the natural logical operations in this space, just as the Boolean logic gates are the natural logical operations for set theory.

Classical set theory is binary and discrete, whereas vector spaces are smooth and continuous. You find this difference between continuous and discrete interpretations of logic in very simple sentences. If somebody says "you should take bus 60 or bus 70", and you later discover that they meant you to take bus 67, you would think you had been very badly advised. If, however, you were told that the journey would take "60 or 70 minutes", and it took 67 minutes, you would agree that you had been advised perfectly wisely.

Therefore, it has been attempted that these properties are reflected in the logics that arise in these spaces. As an example, the *Vector Negation* in WORDSPACE is inspired by Quantum Logic, which was introduced by Garrett Birkhoff [Bir67], the founder of modern lattice theory, and John von Neumann [BvN36], the founder of modern computer architecture. It is acknowledged by this logic that a highly ambiguous word within a word space resembles the status of a particle in Physics, where a particle could be in more than one state simultaneously according to Quantum Mechanics.

A continuous interpretation of logic is also subject to two other models, the probabilistic and the fuzzy set models, which pose an alternative to vector space models. These three together constitute the principal mathematical models used for information retrieval. The fuzzy set model is an adaptation of the traditional Boolean model to cope with continuous values. A traditional thesaural relationship in information retrieval is that of a "Broader Term". Fuzzy sets for information retrieval are discussed at length by [Miy90]. A probabilistic model, on the other side, is based upon networks using probability theory to find, for instance, important paths in directed graphs and the path, which leads between a query and relevant documents. One of the most interesting probabilistic models is the *inference network* used in the INQUERY system [CCH92,TC89].

Despite the practical differences among the models, there is a unifying theme behind normalised vectors, fuzzy sets and probability distributions: they all replace binary values (0 and 1), which signify 'not belonging' and 'belonging', with continuous values, which signify 'partially belonging' or 'probably belonging'. To this extent, they all seem to be different implementations of one underlying space model.

At attempt has been undertaken in 20th century's mathematics to weave the threads of hierarchies, vector spaces and Boolean logic as separate mathematical systems together. The are conceived as different variants of one underlying structure called *Lattice* [Bir67]. The distinctive property of a lattice is that two elements can be disjoined (allowed to spread out and diversify) and conjoined (woven together).

Consequently, *Concept Lattices* [GW99] are used to describe a way words are represented, e.g., different kinds of *Horse*. *Ordered Sets* form the most basic building block of a lattice, where a an ordering relationship, e.g., $\leq$ for natural numbers, is assumed. There are also two key operators, however, necessary to turn an ordered set into a lattice: the *join* and *meet* operators as a complementary pair of operators, which correspond to disjunction and conjunction, within a lattice. In other words, the paths upwards or downwards from any two nodes always meet together.

It turned out, however, that concept lattices are quite complicated as a data structure in terms of the number of concepts generated. In order to simplify a generated lattice, association rules are typically used for clustering concepts on the lattice. To this extent, concept hierarchies and conceptual scaling have been introduced. Moreover, in order to cope with uncertainty information for conceptual clustering and concept hierarchy generation, fuzzy logic has been recently incorporated into *Formal Concept Analysis* (FCA) [THFC06], since traditional FCA-based conceptual clustering approaches are hardly able to represent vague information.

The use of coordinates and dimensions has also become a cornerstone in representing mental processes in Cognitive Science. In particular, the formation of *Conceptual Spaces* in order to address different stimuli such as taste, colour, etc., as points in a conceptual space has been subject to [Gär00].

3.2 The Suite of Metrics

Metrics are an important technique in quantifying software and software development characteristics. Theoretical and empirical validation of metrics and their relation to software attributes is a cumbersome and long process [FP97]. It has been stated that it is not sufficient to prove their definitions correct but also their usefulness to describe the software characteristics needs to be validated. In most cases, this can only be achieved through controlled experiments or through analysing large amounts of data.

However, in the remainder of this paper, we introduce a methodology, which deals with metrics as dimensions of a metric space. A proposed AOSD is conceived as being a cloud of points standing for *aspects* located within the metric space according to the various metrics at syntactic and semantic level. Some of the metric parameters are firstly introduced with this paper in order to express more adequate AOSD-to-OOSD metrics. To this extent, it is possible to compare by calculating the distance of proposed AOSD's from each other as well as for the OO counterpart. The closer the AOSD lies to the centre of the multi-dimensional metric space and away from its boundary, the higher the quality and the more sense it makes to apply the proposed AOSD. The suggested multi-dimensional metric space has its counterpart in mathematics in a Euclidean n-sphere.

We distinguish between *syntactic metrics* and *semantic* ones. Syntactic metrics are based solely on counting items in the source code, whereas semantic metrics are meant to measure some aspects of aspect oriented software design. Therefore, we address both code and design evaluation.

Syntactic Metrics Measuring building blocks and constructs of AOP.

Number of Aspects Counting the total number of aspects implemented by the software developer.

Number of Pointcuts per Aspect (NPA) Counting the number of *pointcuts* implemented within an aspect.

Number of Advices per Aspect (NAA) Counting the number of *advices* implemented within an aspect.

Number of Adviced Pointcuts per Aspect (NAPA) Counting the number of *locations* at the core code, where the described code needs to be inserted (joinpoints). In other words, this is a measure of how many described *pointcuts* are being used by *advice* constructs.

Response for Advice (RAD) Counting the number of methods as assigned to a specific advice. This amounts to the number of methods to be inserted and executed at a particular location in the core code.

Degree of Crosscutting per Pointcut (DCP) Counting the number of classes, which are being crosscut by a pointcut within an aspect.

Crosscutting for Aspect (DCA) Counting the number of classes, which are being crosscut by an aspect. Assumption: Same classes are counted as separate ones.

Impact of Aspect on Design (IAD) This metric is meant to count the number of classes within the OO software design, which are affected by an aspect...This metric will be based on CCP, NPA.

Impact of Aspect on Core Code (IACC) This metric is meant to count the amount of methods and variable assignment to be inserted all over the OO based core code. The metric will be based on RAD, NPA, CCP.

Semantic Metrics Here we need not only numbers but also references to knowledge such as names of classes and structures of the OO software..Therefore, we are assuming the existence of a knowledge repository based on a sort of an Ontology for the OO design or implementation.

Cohesion of Aspect (CoA) This metric is meant to calculate the degree to which the methods in *advice* constructs are related with each other. Recursively, the degree to which advices are related to each other and to pointcuts, respectively.

Crosscutting Cohesion of Aspect (CCA) This metric is meant to calculate the degree to which a crosscutting concern is implemented by an aspect. It relies on (a) number of affected classes as well as (b) their distance on the OO software design structure. For instance, the farther affected classes are on the OO taxonomy, the lower the degree of crosscutting cohesion.

Semantic Distance of Aspect to Classes (SDAC) This metric is meant to calculate the degree to which an aspect is similar to the subset of classes, which are being crosscut. It might provide the basis for calculating improvement of the core code and, eventually, modular improvement.

Semantic Distance of Aspects (SDAA) This metric is meant to calculate the degree to which categories of implemented aspects are close to each other in terms of (overlapping?) crosscut classes and posed advices.

Semantic Distance of AO to OOSD (SDAO) This metric is meant to calculate how far the AOSD is from the OOSD counterpart. The farther the better.

3.3 The Metric Space

Another distinguishing issue is the consideration of the metrics as dimensions of a metric space. A proposed AOSD is conceived as being a cloud of points standing for *aspects* located within the metric space according to the various metrics at syntactic and semantic level. Some of the metric parameters are firstly introduced with this paper in order to express more adequate AOSD-to-OOSD metrics. To this extent, it is possible to compare proposed aspects-oriented software designs by calculating their position within the metric space and the distance from each other as well as for the OO counterpart. The closer the AOSD lies to the centre of the multi-dimensional metric space and away from its boundary, the higher the necessity and the more sense it makes to apply the proposed AOSD. The suggested multi-dimensional metric space has its counterpart in mathematics in a Euclidean n-sphere.

References

[BH83] V.R. Basili and D. H. Hutchens. An empirical study of a syntactic complexity family. *IEEE Transactions on Software Engineering*, 9(6):664–672, 1983.

[Bir67] G. Birkhoff. *Lattice Theory.* American Mathematical Society, 3rd edition, 1967.

[BJSSA05] A. Burton-Jones, V.C. Storey, V. Sugumaran, and P. Ahluwalia. A Semiotic Metrics Suite for Assessing the Quality of Ontologies. *Data and Knowledge Engineering*, 55(1):84–102, October 2005.

[BM91] C. B. Boyer and U. C. Merzbach. *A History of Mathematics.* Wiley, 1991.

[Bow78] J. B. Bowen. Are current approaches sufficient for measuring software quality? In *Proceedings Software Quality Assurance Workshop*, pages 148–155, 1978.

[BvN36] G. Birkhoff and J. von Neumann. The Logic of Quantum Mechanics. *Annals of Mathematics*, 37:823–843, 1936.

[CCH92] J. P. Callan, W. B. Croft, and S. M. Harding. The INQUERY retrieval system. In *Proceedings of DEXA-92, 3rd International Conference on Database and Expert Systems Applications*, pages 78–83, 1992.

[CK94] S. R. Chidamber and C. F. Kemerer. A Metrics Suite for Object-Oriented Design. *IEEE Transactions on Software Engineering*, 20(6):476–493, 1994.

[CK03] Y. Coady and G. Kiczales. Back to the Future: A Retroactive Study of Aspect Evolution in Operating System Code. In *Proceedings International Conference on Aspect Oriented Software Development (AOSD'03)*, pages 50–59, Boston, Massachusetts, USA, 2003.

[FP97] Norman E. Fenton and Shari L. Pfleeger. *Software Metrics: A Rigorous and Practical Approach.* PWS Publisher, 2nd edition, 1997.

[Gär00] Peter Gärdenfors. *Conceptual Spaces: The Geometry of Thought.* The MIT Press, Cambridge, Massachusetts, 2000.

[Gra62] Hermann Grassmann. Extension Theory. In *History of Mathematical Sources*. American and London Mathematical Society, 1862.

[GW99] B. Ganter and R. Wille. *Formal Concept Analysis: Mathematical Foundations*. Springer, 1999.

[GW02] N. Guarino and C. Welty. Evaluating ontological decisions with Ontoclean. *Communications of the ACM*, 45(2):61–65, 2002.

[HCN98] R. Harrison, S. J. Counsell, and R. V. Nithi. An Evaluation of the MOOD Set of Object-Oriented Software Metrics. *IEEE Transactions on Software Engineering*, 24(6):491–496, 1998.

[KPF95] B. Kitchenham, S. L. Pfleeger, and N. E. Fenton. Towards a Framework for Software Measurement Validation. *IEEE Transactions on Software Engineering*, 21(12):929–943, 1995.

[LL00] M. Lippert and C. Lopes. A Study on Exception Detection and Handling Using Aspect-Oriented Programming. In *Proceedings International Conference on Software Engineering (ICSE'00)*, pages 418–427, Limerick, Ireland, 2000.

[Lop97] Christa V. Lopes. *D: A Language Framework for Distributed Programming.* Phd thesis, College of Computer Science, Northeastern University, November 1997.

[Miy90] S. Miyamoto. *Fuzzy Sets in Information Retrieval and Cluster Analysis.* Kluwer, 1990.

[OYE06] A.M. Orme, H. Yao, and L. Etzkorn. Coupling Metrics for Ontology-based Systems. *IEEE Software*, 23(2):102–108, March/April 2006.

[Sch98] Heinrich Schütze. Automatic word sense discrimination. *Computational Linguistics*, 24(1):97–124, 1998.

[SGC$^+$03] C. Sant'Anna, A. Garcia, C. Chavez, C. Lucena, and A. von Staa. On the Reuse and Maintenance of Aspect-Oriented Software: An Assessment Framework. In *XVII Brazilian Symposium on Software Engineering*, 2003.

[She93] M. Shepperd. *Software Engineering Metrics*, volume Volume I: Measures and Validations. McGraw Hill International, London, UK, 1993.

[SLHA00] R. Stamper, K. Liu, M Hafkamp, and Y. Ades. Understanding the role of signs and norms in organisations - A semiotic approach to information systems design. *Behaviour and Information Technology*, 19(1):15–27, 2000.

[TC89] H. Turtle and W. B. Croft. Inference networks for document retrieval. In *Proceedings of the 13th Annual Conference on Research and Development in Information Retrieval (SIGIR-89)*, pages 1–24, 1989.

[TCB04] S. L. Tsang, S. Clarke, and E. Baniassad. An Evaluation of Aspect-Oriented Programming for Java-based Real-Time Systems Development. In *7th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*. IEEE Computer Society, 2004.

[THFC06] Quan Thanh Tho, Siu Cheung Hui, A.C.M. Fong, and Tru Hoang Cao. Automatic Fuzzy Ontology Generation for Semantic Web. *IEEE Transactions on Knowledge and Data Engineering*, 18(6):842–856, 2006.

[vR04] Keith van Rijsbergen. *The Geometry of Information Retrieval.* Cambridge University Press, Cambridge, United Kingdom, 2004.

[Wid04] Dominic Widdows. *Geometry and Meaning.* CSLI publications, Stanford, California, 2004.

[WW95] Y. Wand and R. Weber. On the Deep Structure of Information Systems. *Journal of Information Systems*, pages 203–223, 1995.

[YOE05] H. Yao, A.M. Orme, and L. Etzkorn. Cohesion Metrics for Ontology Design and Application. *Journal of Computer Science*, 1(1):107–113, 2005.

[Zha02] J. Zhao. Towards a metrics suite for aspect-oriented software. Technical Report SE-136-25, Information Processing Society in Japan, 2002.

[Zus98] H. Zuse. *A Framework of Software Measurement*. Walter de Gruyter, Berln-NY, 1998.

[ZX04] J. Zhao and B. Xu. Measuring Aspect Cohesion. In *Proceedings International Conference on Fundamental Approaches to Software Engineering*, volume 2984, pages 54–68. Springer Verlag, 2004.