IDEA GROUP PUBLISHING

701 E. Chocolate Avenue, Hershey PA 17033-1117, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com

ITP4150

Toward Entropy Based Metrics For Separation Of Concerns

Jana Dospisil and Nisha Leena Sinha Roy
School of Network Computing, Monash University, Australia {jana.Dospisil, NishaRoy}@infotech.monash.edu.au

ABSTRACT

The claim of improved efficiency and decreased complexity of software of using tools such as Aspect/J or Hyper/J to separate different concerns in OO applications does not seem to have any theoretical underpinning. In this paper, we attempt to review the current research and suggest theoretical framework for complexity ranking in OO applications coded with Hyper/J.

INTRODUCTION

The primary objectives of software engineering discipline are to improve the quality of developed software, and provide tools for reducing the software complexity. These objectives possibly lead to reduced cost of software development, facilitate maintenance and allow evolution and extension of the software.

For some time it has been estimated that over 70% of software development effort is spent in testing and maintenance of software [25]. The reports from large commercial projects, which utilize the object-orientated techniques, indicate that the expected cost savings in maintenance have not been delivered. The increased complexity and size of software projects have led to the development of many different concepts for breaking a system into less complex and manageable modules [8]. The principle of separation of concerns also made its way into OO design.

The source of the problem in software development is that some kinds of behavior or functionality *cross cut* or are *orthogonal to* classes in many object-oriented components and they are not easily modularized to a separate class. Examples of such behavior include the following: synchronization and concurrency, performance optimization, exception handling and event monitoring, coordination and interaction protocols, and object views.

Recent research at Xerox PARC in Aspect Oriented Programming (AOP) [16] and Multidimensional separation project at IBM [19] seek to alleviate this. Language constructs and compilers interleave component and aspect definitions (programs) appropriately to formulate a unified and executable program. The objective of these tools is to reduce the complexity and promote easy maintenance.

Motivation and Background

Hyper/J*:* Ossher and Tarr [19] assert that the separation of concerns and aspect-oriented programming are at the core of software engineering and have sparked the development of many tools and approaches to modularization of software. There is now widespread realization in that a design and implementation that is solely based on class components has its limitations. This is because object-oriented languages, such as C++, Smalltalk and Java, are not capable of expressing certain aspects of applications in a reusable way. Tarr and Osher [23] state that *"Done well, separation of concerns can provide many software engineering benefits, including reduced complexity…"*. To measure the quality of separation either in *N-dimensional* space or even the orthogonal separation seen in *Aspect/J*, the new set of metrics is required.

Aspect/J*:* Aspects tend not to be units of system's functional decomposition, but rather the properties that impact the performance of semantics of components. The examples include synchronization of concurrent objects and memory access patterns. The objective is to alleviate some of the complexity and tangled code that results due to cross cutting/ orthogonal behavior; this behavior is inevitable in any real application. *Aspect Weaverä* provides compositional service based on the concept of *join points* [16, 17].

So far, there have been no metrics or measures to clearly indicate that the complexity of the code has been reduced, and improvements in maintainability have been achieved. This paper deals with theoretical underpinning of proposed metrics to measure the complexity of modules in *Hyper/J*. Unfortunately, due to the limited size of this paper we are not able to demonstrate the methodology in full scale.

The layout of this paper is as follows. Section one provides the overview of current OO metrics including entropy based complexity measures. The second section starts with the overview of Hyper/J concepts followed by the condensed description of the theoretical underpinning of the proposed complexity framework in *Hyper/J*.

Overview of OO Metrics

There is a growing concern that software metrics should have a solid base. Measurement theory can be used to translate mathematical properties of measures to empirical (intuitive) properties. Rigorous approach to OO metrics has been suggested by a number of researches. In literature, more than hundred software measures for OO applications can be found. Since 1995, the trend towards incorporating measurement theory into all software metrics has led to identification of scales for measures thus providing some perspective on dimensions. In [13], the following scales are suggested: ordinal, interval, ratio and absolute.

Axiomatic approach was proposed by Weyuker [24]. This framework is based on a set of nine axioms against which software could be formally evaluated. The description and criticism can be found in [13]. Fenton [10] uses the term software metrics to describe the following artifacts:

- A number which is derived, usually empirically, from a process or code (LOC),
- A scale,
- An attribute which is used to provide specific functionality ("portability").

These descriptions typically lead to a wide spread confusion between models, and their ability to predict desired software characteristics thus their suitability to be used for estimation purposes. One of the criticisms of many proposed metrics is the lack of theoretical basis. Many metrics show dimensional inconsistencies, or their results are derived from correlative or regression analysis [13].

Zuse based his OO software metrics on measurement theory [21] analyzed in [25]. The quantitative criteria for software measures in the area of structured programming are based on the theory of extensive structure and a set of empirical conditions (axioms). The approach of extensive structure can also be applied to cost estimation models, design and maintainability measure. He proved that the dynamic nature of OO software requires the use of quantitative and qualitative probabilities and belief structures (e.g. Dempster-Shafer Function of Belief, the Kolmogoroff axioms, and others).

Chidamber [5] proposed a suite of metrics for OO design which consists of six metrics with foundation in measurement theory: *Weighted Methods per Class* (WMC), *Depth of Inheritance Tree* (DIT), *Number of Children* (NOC), *Coupling Between Object Classes* (CBO), *The Response for a Class* (RFC), and the *Lack of Cohesion Metric* (LCOM). The criticism by Churcher [6] is pointing to the ambiguity of some metrics, particularly WMC. Hitz [15] and Fetchke [9] showed that CBO does not use sound empirical relation system, particularly, that it is not based on the extensive structures. Furthermore, LCOM metric allows representation of equivalent cases differently thus introducing additional error.

Coupling and cohesion measures form the important group of measures in assessment of dynamic aspects of design quality. The example in Hitz [14] clearly distinguishes the difference between static and dynamic class method invocation: *number of methods invoked by a class compared to frequency of method invocation*. Concise survey and discussion of coupling including critique of current metrics can be found in [3] and [27]. The metrics suite capable of capturing dynamic behavior of objects with regard to coupling and complexity has been presented by Yacoub in [27]. A set of scenarios in the implementation depicts dynamic behavior. The *Export* and *Import Object Coupling* metrics are based on percentage of message exchange between class instances (objects) to the total number of messages. The *Scenario Profiles* introduce the estimated probability of the scenario execution. The complexity metrics are aimed predominantly at the assessment of stability of active objects as frequent sources of errors.

Entropy Based Complexity Measures

Entropy based complexity measures rely on theory of information [11, [4]. The approach taken by Davis and LeBlanc [7] who quantify the differences between *anded* and *neted* structures using Shannon and Weaver's concept of information entropy [22]. This measurement is based on chunks of FORTRAN and COBOL code (represented by nodes in the DAG) with the same in-degree and the same out-degree to assess syntactic complexity. In 1976, Belady and Lehman [2] elaborated on the law of increasing entropy: the entropy of a system (level of its unstructuredness) increases with time, unless specific work is executed to maintain or reduce it.

In Harrison [12], software complexity metric is based on empirical program entropy. A special symbol, reserved word or a function call is considered as operator (it is assumed that they have certain natural probability distribution [26]). The probability p_i of *ith* most frequently occurring operator is defined as (Eq 1)

$$p_i = {f_i}/{N_i}$$ Equation 1 Probability of occurrence of i-operator

where f_i is the number of occurrences of the *ith* operator and N_i is total number of nonunique operators in the program. The complexity is given (Eq. 2) as

$$H = -\sum_{i=1}^{Ni} p_i \log_2 p_i$$ Equation 2 Entropy-based complexity measure

The Average Information Content Classification measure (Eq.3):

$$AICC = -\sum_{i=1}^{Ni} \frac{f_i}{N_i} \log_2 \frac{f_i}{N_i}$$ Equation 3 Average Information Content Classification

This metric provides only the ordinal position thus restricting the way of usage. It was tested on C code. It does not indicate the *"distance"* between two programs. The work of Bansiya and Davis [1] introduces similar complexity measure – *Class Definition Entropy* (CDE) replacing the operators of Harrison with name strings used in a class. The assumption that all name strings represent approximately equal information is related to the possible error insertion by misusing the string. The metric has been validated on four large projects in C++ and results have been used to estimate *Class Implementation Time Complexity* measure.

Single valued measure of complexity is appealing to managers as the simple indicator of development complexity. However, as discussed in Fenton's book [10], single value cannot be used for assessment of quality of the entire product. The measures bound to a single product attribute (e.g. *Comprehensibility or reliability* etc) cannot be used as prediction models or as guidance for improving the quality of the product.

PROPOSED ENTROPY BASED METRIC FRAMEWORK FOR SEPARATION OF CONCERNS

Overview of Hyper/J concepts

The term *multi-dimensional separation of concerns* denotes the separation of multiple, arbitrary kinds (dimensions) of concerns simultaneously. A clean separation of concerns allows isolation and encapsulation of all concerns, which promotes traceability and reduces complexity. *Concerns* are defined as primary entities for decomposing software into manageable and comprehensible modules [20], such as classes, features, aspects and roles. The prevalent kind of concern is a class (data type). A *hyperspace* describes the following properties of concerns: identification, encapsulation, and mutual relationships. The concern matrix organises units according to dimensions and concerns. The encapsulation of concerns is accomplished by introducing mechanism of *hyperslices*. A *hypermodule* comprises a set of *hyperslices* being integrated and a set of relationships, which determine mutual dependency between *hyperslices*. The level of mutual dependency is an important parameter.

Formally, *hyperspace* is a tuple *{U,M,H}* where *U* is a set of units (methods are primitive units, classes are modules), M is a concern matrix and *H* is a set of *hypermodules*. *Hypermodule* is a tuple (HS, CR) where HS is a set of *hyperslices* and CR is a set of composition relationships. A *hyperslice* is a *declaratively* complete concern ($hs \in C$). A *concern* is modeled as a predicate, *c*, over units *U*. The unit set is then defined as (Eq. 4)

$$U(c) = \{u \in U \mid c(u)\}$$ Equation 4 Units set with a concern

Concerns are said to overlap if their unit sets are not disjoint. A *dimension of concern* is a set of concerns whose unit sets partition *U*. It implies that *the concerns within a dimension cannot overlap*, and must cover all the units. This leads to the declarative completeness constraint. Declarative completeness serves as a mechanism to reduce high coupling in interrelated units (methods). In order to make *hyperslice* declaratively complete, we have to at least declare units from other *hyperslices* (thus allowing later binding). For example, the unit $u_1 \in hs$ calls unit $u_2 \in U, then \rightarrow u_2 \in hs$ and it must be implemented in some other *hyperslice*. We denote these units u_{2decl} to satisfy the completeness constraint. *Hyper/J* also defines the *Implementation set* (Eq. 5)

$$I(hs) = \{u \in hs \mid \neg decl(u)\}$$ Equation 5 Implementation set

A *Composition Relationship* is a tuple *(I,r,f,o)* where *I* is a tuple of input units, *r* is a correspondence relationship characterizing the relationship of units in *I*, *o* is an output unit produced using *f* which is the composition function (Eq. 6)

$$f : (I \times r) \mapsto U$$ Equation 6 Composition function

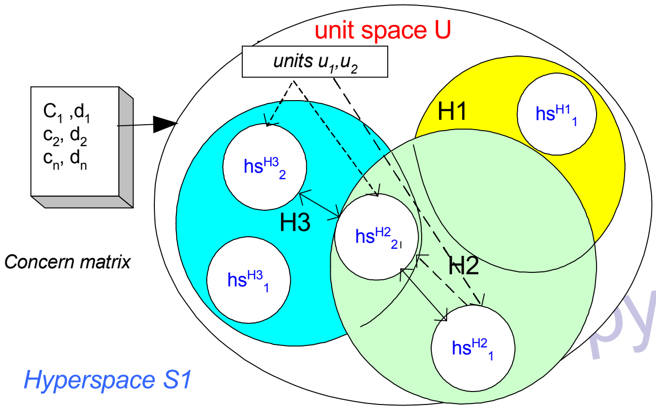
This property means that the *hyperslice* is self-contained, providing we define the association called *correspondence* and supply corresponding units u_{decl}.

Proposed Entropy based ordering framework

Figure 1 shows graphically a conceptual representation of overlapping *hypermodules*, *concerns matrix* and self- contained *hyperslices*. The *hypermodules* H_2 and H_3 overlap through *hyperslices* $hs^{H3}{}_2$ and $hs^{H2}{}_2$. The units u_1 and u_2 are present in both *hypermodules*.

Hypermodule complexity ordering: a *hypermodule* is a message in which the special symbol carrying the information is a *unit* u_i (method or a shared variable). In a complex system, the invocation of a unit u_i

Figure 1: Hyperspace and overlapping hypermodules



is not deterministic; it depends on the occurrence of outside events – execution scenarios. Let the unit space U be a discrete random variable which occurs with the probability mass function *p(u)*. The measure of uncertainty of U (entropy) is defined as:

$$H(U) = -\sum_{i=1}^{M} p(u_i)\log p(u_i)$$ Equation 7 Entropy of units

The entropy depends only on the probabilities. The probability $p(u_i)$ of the most occurring unit is the percentage

$$p(u_i) = \frac{f_{ui}}{N_{all}}$$ where f_{ui} is the number of occurrences of the unit u_i

and N_{all} is total number of nonunique units in the *hypermodule*.

Concerns matrix ordering per hypermodule: the complexity of the *concern matrix* defined over the units soace U is the entropy (Eq. 8)

$$H(C) = -\sum_{i=1}^{M} p(c_i)\log p(c_i)$$ Equation 8 Entropy of concerns

$$p(c_i) = \frac{f_{ci}}{N_{Call}}$$ where f_{ci} is the frequency of occurrence the units

using the concern c_i and N_{Call} is the number of all non unique units using all concerns encapsulated by the *hypermodule*.

These two ordinal measures provide only ordering of *hypermodules* according to their entropic complexities of units and concerns. They only indicate design differences among *hypermodules*.

Relative entropy as uncertainty reduction measure: we assume that relative entropy calculated for units and concerns provides ranking of dependencies among units. *Definition*: *Mutual information* is a measure of the amount of information that one random variable contains about another random variable. *Relative entropy* is a measure of distance between two probability mass functions p and q representing different distributions of *units* u_i. It is also said to be a measure of inefficiency of assuming that the distribution is q when the true distribution is p. The relative entropy is (Eq. 9)

$$D(p\|q) = \sum_{u_i \subset U} p(u_i)\log_2\left(\frac{p(u_i)}{q(u_i)}\right)$$ Equation 9 Relative entropy of units

Mutual information indicates discrepancy between real-time outcomes and states (with $pmf = p$) and anticipated design intentions ($pmf = q$).

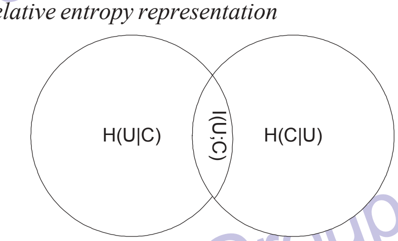
Declarative completeness in Hyper/J: software artifacts are subject to a completeness constraint in which each declaration unit in a system must correspond to compatible definition or implementation in some *hyperslice* [20]. Let's consider an execution scenario in which a set of units U implements a set of concerns C: $U(c) = \{u \in U \mid c(u)\}$. Furthermore, assume joint probability mass function *p(u,c)* indicating the dependency of units and concerns, marginal probability mass functions for units and concerns respectively *p(u)* and *p(c)* written about the edges of the joint probability table. The relative entropy between the joint distribution and product distributions of participating sets *I(U;C)* is a measure of dependence between two variables U and C and the product distribution *p(u)p(c)*

$$H(U;C) = H(U) - H(U \mid C) = \sum_{u}\sum_{c} p(u,c)\log\frac{p(u,c)}{p(u)p(c)}$$

Equation 10 Relative entropy for units and concerns

The *mutual information* is the reduction in the uncertainty in unit space of U due to knowledge of concern space C (see Figure 2.).

Figure 2: Relative entropy representation



By using pairs u_i and declaration of u_{idecl} as the mutual information the *hyperslices* keep about each other, we can derive the estimate of efficiency with regard to the declarative overlap in *H(U;C)*. We assume that $f_{ci} = f_{ui} - f_{uidecl}$ where f_{uidecl} is the frequency of units u_{decl} relevant to correspondence association. Furthermore, joint probability mass function of units and concerns $p(u_i, c_i)$ represents the percentage of units encapsulating a given concern to all non-unique units in a *hyperslice*. Tabular representation of joint probability of concerns/units and marginal probabilities for each component in a *hypermodule* provides the following information:

- Estimated complexity of each *hyperslice* with regard to encapsulated concerns and relative entropies of *hypermodule*;
- Ordering of *hyperslices* in a *hypermodule* according to their mutual dependency;

Weighted entropy for ordering units and concerns according to their contribution to utility: Sometimes we deal with units where it is necessary to take into account their importance and some qualitative characteristics. We need to associate elementary unit with both the probability with which it occurs and its qualitative weight. A criterion for a qualitative differentiation of the units of a given scenario represented by the relevance, the significance, or the utility of the information they carry with respect to an outcome, with respect to a qualitative characteristic. The occurrence of a unit removes a double uncertainty: the qualitative one, related to the probability with which it occurs, and qualitative one, related to a given qualitative characteristic. For instance, a unit of a small probability can have a great utility with respect to concurrency aspects; likewise, a unit of great probability can have small impact on maintainability (we shall relate this observation to utility). The weights may have either objective or subjective character. Thus, the weight of one unit may express some qualitative objective characteristics, but also it may express the subjective utility of the respective unit with respect to the software complexity. The weight ascribed to an elementary unit may also be related to the subjective probability with which respective units are used, and it does not always coincide with the objective probability.

In order to distinguish the units $u_1, u_2, u_3, \ldots u_n$ in the unit *space U* according to their importance with respect to a given qualitative characteristics of implemented or referred to concern, we assign to each unit a non-negative weight proportional to its importance and significance.

$$H\left(w(u)_i ; p(u)_i\right) = -\sum_{i=1}^{n} w(u)_i \, p(u)_i \log_e p(u)_i$$

Equation 11 Weighted entropy

Where $p(u)_i = p(u_i) = \frac{f_{ui}}{N_{all}}$ Equation 12 Probability of occurrence for a unit u_i

The weights are constructed as ratio of the objective probability of the occurrence of this unit to the amount of information it holds.

$$w(u)_i = -\frac{p(u_i)}{\log_e p(u_i)}$$

Equation 13 Objective probability weights assignment

In this case we obtain the following expression for weighted entropy.

$$H(u_i) = \sum_{i=1}^{n} p(u_i)^2$$

Equation 14 Weighted entropy for a *hypermodule*

The tabular representation of entropy with objective weights for each *hypermodule* enables ordering of *hypermodules* with regard to different aspects of concern (e.g.concurrency implementation, memory utilization and others).

CONCLUDING REMARKS

This paper provides an overview and theoretical underpinning of entropy based complexity metrics for ordering *hypermodules* and *hyperslices* according to their complexity in *Hyper/J*. We are proposing the following complexity measures:

- *Hypermodule* complexity ranking,
- *Concerns matrix* ranking per *hypermodule,*
- Relative entropy as uncertainty reduction measure,
- Weighted entropy for ranking units and concerns according to their contribution to utility.

We acknowledge that the following aspects still have to addressed:

- Parser is being constructed to allow data collection in Java classes and *Hyper/J,*
- Validation study on a larger scale must be conducted on at least two comparative applications:
  - o Case 1: the application is designed and coded without separation of concerns concept in mind
  - o Case 2: the application is designed and coded specifically for Hyper/J

With regard to the limited scope of this paper the methodology addressing the practical use of entropy metrics and precise interpretations of metrics could not be covered.

REFERENCES

[1] Bansiya, J., Davis, C., Etzkorn, L., An Entropy-Based Complexity Measure for Object-Oriented Designs, *Theory and Practice of Object Systems*, Vol. 5(2), pp.11-118, 1999

[2] Belady, L.A. and Lehman, M.M. A Model of a large program development. *IBM Systems Journal*, Vol 15(3), pp.225-252, 1976

[3] Briand, L. Daly, J., and Wurst. A Unified Framework for Coupling Measurement in Object Oriented Systems. *IEEE Transactions on Software Engineering*, Vol. 25, No. 1 Jan/Feb 1999, pp. 99-121

[4] Cover, T.M. and Thomas, J.A., "*Elements of Information Theory*," Willey Series in Telecommunications, John Wiley & Sons, new York, 1991

[5] Chidamber, S. and Kemerer, C. A Metric Suite for Object Oriented Design. *IEEE Transactionas on Softwrae Engineering*, Vol.20,No.6, June 1994, pp.476-49

[6] Churcher, N. and Shepperd, M.. A Metric Suite for Object Oriented Design. *IEEE Transactionas on Softwrae Engineering*, Vol.20, No.6, June 1994, pp.476-49

[7] Davis, J.S., and LeBlanc, R.J. A Study of the Applicability of Complexity Measures. *IEEE Transactions on Software Engineering*, Vol 14(9), pp.1366-1371, 1988

[8] Dijkstra, E.W. *A Discipline of Programing*. Prentice Hall, 1976

[9] Fetchke, T. Software Metriken bei der Objectorientierten Programmierung. *Diploma Thesis*, GMD, St. Augustin, 1995

[10] Fenton, N. and Pfleeger, S. L. *Software Metrics: A Rigorous & Practical Approach*. International Thomson Computer Press, 1997

[11] Gray, R. M., "*Entropy and Information Theory"*, Springer-Verlag, New York, 1990

[12] Harrison, W. An Entropy-Based Measure of Software Complexity. *IEEE Transactions of Software Engineering,* Vol. 18, No. 11, Nov. 1992, pp. 1025-1029

[13] Henderson-Sellers, B. "*Object-Oriented Metrics measures of Complexity*", Prentice Hall PTR, 1996

[14] Hitz, M., and Montazeri, B. Measuring Product Attributes of Object-Oriented Systems. In *Proc. 5th European Software Engineering Conference (ESEC'95*), Barcelona, Spain, 1995, pp. 124-136

[15] Hitz, M., and Montazeri, B. Chidamber & Kemerer's metric Suite: A Measurement Theory Perspective. *IEEE Transactions on Software Engineering,* Vol. 22. No. 4, April 1996, pp.270-276

[16] Kiczales, G., J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J. - M. Loingtier, and J. Irwin, "Aspect Oriented Programming," Xerox Corporation, 1997. *http://www.parc.xerox.com/spl/projects/aop/*

[17] Kiczales, G., C. Lopes, "Tutorial: Aspect-Oriented Programming w/AspectJ™, Xerox PARC, *www.parc.xerox.com/aop*.

[18] Kiczales, G., J. M. Ashley, L. Rodriguez, A. Vahndat, and D. G. Bobrow, "Metaobject Protocols: Why we want them and what else they can do," in *Object Oriented Programming: The CLOS Perspective*, A. Paepcke, Ed., MIT Press, 1993, pp. 101- 11

[19] Ossher, H., Tarr, P., Multi-Dimensional Separation of Concerns and The Hyperspace Approach, *Technical Report, IBM T.J. Watson Research Center*, New York, 1998

[20] Ossher, H., Tarr, P., Multi-Dimensional Separation of Concerns in Hyperspace: *Position Paper, IBM T.J. Watson Research Center, New York, 1998*

[21] Roberts, Fred, S. "*Measurement Theory with Applications to Decision Making, Utility, and Social Sciences"*. Encyclopedia of Mathematics and its Applications, Addison Wesley Publishing Company, 1979

[22] Shannon, C. E., "*A Mathematical Theory of Communication"*, University of Illinois Press, Illinois, 1949

[23] Tarr, P. Ossher, H. Harrison, W, and Sutton, Jr. N degrees of Separation: Multi-Dimensional Separation of Concerns. *Proc. of the 21st International Conference on Software Engineering*, 1999

[24] Weyuker, E. J., *Evaluating Software Complexity Measures*, *IEEE Transactions on Software Engineering*, Volume: 14 No. 9, September 1988, pp.1357 - 1365

[25] Zuse, H. *Software Complexity Metrics/Analysis*. Marciniak, J. (Ed): Encyclopedia of Software Engineering, Vol. I, John Willey& Sons, Inc. 1994, pp. 131-166.

[26] Zweben, S. and Hasltead, M. The Frequency Distribution of Operators in PL/I Programs. *IEEE Transactions of Software Engineering*, Vol.5. pp. 91-95, Mar. 1979.

[27] Yacoub Sherif M., Ammar, Hany,H., and Tom Robinson. Dynamic metrics for Object Oriented Designs. *Proc. Of 6th International Symposium on Software Metrics (METRICS'99)*, Boca Raton, Nov. 4-6, 1999, pp. 50-61

Related Content