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.

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 researchers. Literature, more than hundred software metrics 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 that provide 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 a regression or correlation 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 nanded 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 $i$th most frequently occurring operator is defined as (Eq 1)

$$p_i = \frac{N_i}{N}$$

where $N$ is the number of occurrences of the $i$th operator and $N_i$ is total number of nonunique operators in the program. The complexity is given (Eq. 2) as

$$H = -\sum_{i=1}^{N} p_i \log_2 p_i$$

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

$$AICC = -\frac{1}{|C|} \sum_{c \in C} f_c \log_2 \frac{f_c}{N}$$

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 missing the string. The metric has been validated on four large projects in C++ and results have been used to estimate Classification Time Complexity measure.

Single valued measure of complexity is appealing to managers as the simple indicator of development complexity. However, as discussed in Fenich’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 clear 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 $(H,S)$ where $H$ is a set of hyperslices and $S$ 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 $u_i \in hs$ calls unit $u_j \in U$, then $u_i \in hs$ and it must be implemented in some other hyperslice. We denote these units $u_{i,hs}$ 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 $(I,x,r,o)$ 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)

$$f : (I \times r) \rightarrow U$$

Equation 6 Composition function

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

**Proposed Entropy based ordering framework**

Figure 1 shows graphically a conceptual representation of overlapping hypermodules, concerns matrix and self-contained hyperslices. The hypermodules $H_i$ and $H_o$ overlap through hyperslices $hs^u_{i,o}$, and $hs^{uo}$. The units $u_i$ and $u_o$ 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 $u_i$
Issues and Trends of IT Management in Contemporary Organizations

Figure 1: Hyperspace and overlapping hypermodules

Concern matrix ordering per hypermodule: the complexity of the concern matrix defined over the units space $U$ is the entropy (Eq. 8)

$$H(C) = -\sum_{i=1}^{M} p(c) \log p(c)$$

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)

$$D(p\|q) = \sum_{u_i \in U} p(u_i) \log \frac{p(u_i)}{q(u_i)}$$

Equation 10 Relative entropy for units and concerns

Let’s consider an execution scenario in which a set of units $U$ implements a given concern $C$. 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 $U$ and $C$ is a measure of dependence between two variables $U$ and $C$ and the product distribution $p(u)p(c)$

$$H(U;C) = H(U) - H(U | C) = \sum_u \sum_c p(u,c) \log \frac{p(u,c)}{p(u)p(c)}$$

Equation 2

Figure 2: Relative entropy representation
tive 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 \( u_1, u_2, u_3, ..., u_n \) 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.

\[
H(w(u); p(u)) = -\sum_{i=1}^{n} w(u)_i p(u)_i \log_p(u)_i
\]

Equation 11 Weighted entropy

Where \( p(u)_i = p(u)_i = \frac{f_{u_i}}{N_{all}} \) 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.

\[
w(u)_i = -\frac{p(u)_i}{\log_p(u)_i}
\]

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 HyperJ. 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 HyperJ.
- Validation study on a larger scale must be conducted on at least two comparative applications:
  - Case 1: the application is designed and coded without separation of concerns concept in mind
  - Case 2: the application is designed and coded specifically for HyperJ

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**


