Entropy as a Measure of Object-Oriented Design Quality

Alexander Chatzigeorgiou, George Stephanides

Department of Applied Informatics, University of Macedonia,
156 Egnatia Str., 54006 Thessaloniki, Greece
{achat, steph}@uom.gr

Abstract. In this paper, object-oriented designs are approached from an information theoretic point of view and entropy is proposed as a design quality metric. One of the primary aims of object-orientation is the flexibility and the ease in extending a system's functionality, with limited alterations to existing modules. This feature is evaluated defining an appropriate probability space according to the number of unary associations enabling the definition of an entropy metric. The entropy of the next generation of an object-oriented system with enhanced functionality remains close to its previous level in case the added functionality affects a limited number of existing classes; on the other hand, a poorly designed system increases entropy drastically. In this way, not only a given system is evaluated but it is also possible to assess the degradation of a system and its "distance" from the original design.

1 Introduction

The development of large software systems can benefit significantly from the application of appropriate metrics during all life-cycle stages [1]. Following the increased acceptance of object-orientation (OO) in the software industry, a plethora of metrics have been proposed for assessing several aspects concerning the process, the products and the resources associated with the development of OO-systems. The initial and most well-studied suite of OO-metrics has been proposed by Chidamber and Kemerer [2]. Since then, numerous metrics have been developed, reaching a total of 72 OO-metrics according to [3].

The object-oriented paradigm promises to be one of the most flexible frameworks for developing systems by shifting responsibility from functional modules to a more local level, namely to objects. The merits of object-oriented systems concerning ease of reuse, maintenance, extensibility and scalability are well understood and drive the wide acceptance of object-orientation among software developers.

However, object-oriented design is rather a skill than a set of strict guidelines that can be safely applied. Obviously, not all object-oriented designs are of good quality. According to [4] a good object-oriented design has anticipated major *axes of change* and is designed in a way that does not exhibit "odors" of rigidity and fragility. Even in basic textbooks on object-oriented analysis and design, one of the key features of

the object oriented way of thinking is the ability to add new functionality with limited effort, in terms of code update of existing building blocks. Stated differently, if in order to update an object-oriented system, each of its classes has to be revisited by modifying one or more of its methods and attributes, then object-orientation does not perform any better than procedural programming.

However, it is quite difficult to assess whether an OO-system possesses the characteristics that make it flexible and easily extendible. Although there are metrics that evaluate the degree of object-orientation or measure specific characteristics of the design, such as cohesion and coupling, metrics for evaluating the quality of an object-oriented model with respect to its accordance to well defined criteria, would be useful in object-oriented analysis. Only recently, empirical models for assessing high-level design quality attributes of object-oriented systems have been proposed in [5].

In this paper, we propose the use of entropy as defined in Information Theory, to evaluate the initial status of an object-oriented design as well as its status after the addition of new functionality. The difference between the entropy of the two systems provides insight to the quality of the design in terms of how flexible it has been during the enhancement of its functionality. In case enhancements to the functionality of an object-oriented system affect a limited number of existing classes, the proposed entropy metric remains close to the initial value. In case of a poorly designed system, where functionality enhancements enforce modifications to almost all classes, the entropy increases drastically.

Within the context of OO systems, entropy has been first used by [6]. However, in that work, entropy was defined as conceptual inconsistency between hierarchical levels. Entropy from an information theoretic point of view has been proposed in [7] for evaluating the structuredness of software. However, this metric is based on the file structure and its applicability is limited to large procedural software systems.

The rest of the paper is organized as follows: Section 2 introduces the notion of entropy in information theory, while in section 3 the proposed entropy metric for object-oriented systems is presented. Application results to two OO designs are discussed in section 4. Finally, we conclude in section 5.

2 Entropy

A measure of the information (self-information) contained in an outcome is defined as:

$$I(x_i) = -\log_2(p\{x_i\}) \ . \qquad \textbf{(1)}$$

where $p\{x_i\}$ is the probability of x_i. This definition satisfies the intuitive requirement that the information contained in an outcome is proportional to its uncertainty. Moreover, this definition also satisfies the requirement that the total information in independent events should add.

The *entropy* or *average information content per symbol* for an information source X with alphabet $A = \{a_1, a_2, \ldots, a_n\}$ and probability distribution $P_A = \{ p_1, p_2, \ldots, p_n\}$ is the average self-information associated with each symbol on the output of the source [8]. For the source X as a whole, its entropy is defined as:

$$H(X) = -\sum_{i=1}^{n} p_i \log_2(p_i) \ . \qquad (2)$$

In other words, the more difficult it gets to estimate the outcome at the source output, the larger the entropy becomes. The maximum value attainable by an alphabet's entropy occurs when the symbols are equally likely. In this case, the entropy equals $\log_2 n$. The minimum value (zero) occurs when only one symbol occurs; it has probability one of occurring and the rest have probability zero.

3 Object-Oriented Design Quality

As already mentioned, one of the primary goals of object-orientation is to develop systems that are easily extensible taking advantage of features such as encapsulation, information hiding, inheritance and polymorphism. If an OO design fails to satisfy the requirement for flexibility in upgrading the system's functionality, it certainly lacks one of the most desired features, in a world of constantly changing requirements [4]. Next, we will attempt to define the entropy of an OO system, in order to assess its degradation during functionality upgrades, which in turn depends on the inherent flexibility of the system.

It is assumed that the system is described in a standard class diagram format following UML (Unified Modeling Language [9]) notation for associations between classes. For a randomly selected unary association, we define as $p\{C_i\}$ the probability that the association leads to class C_i. The existence of such an association indicates that class C_i provides services to the rest of the system, since it responds to messages sent to it. Within this context, bi-directional associations are treated as two separate unary associations.

In other words, we consider as *event* the fact that a random association has a direction towards class C_i and consequently, the sample space $S = \{C_1, C_2, \ldots, C_n\}$ consists of all classes in the system. Since, as it will be shown in the next paragraph, probabilities are extracted from the statistics of the system, a randomly selected association with direction towards a class, does not affect the direction of another randomly selected association, and as a result any two events can be considered to be independent. Independence of events is required in order to be able to define the entropy based on the corresponding probability distribution.

The probability $p\{C_i\}$ is extracted from the class diagram as the ratio of unary associations directed towards class C_i over the total number of unary associations in the design. Under this definition, the following two properties are valid:

1. Any probability is a number between 0 and 1: $0 \le p\{C_i\} \le 1$.
2. The sample space, S, of all possible outcomes has a probability of 1: $p\{S\} = 1$.

Now, it is possible to define the entropy of an object-oriented design D with n classes, based on these probabilities:

$$H_D = -\sum_{i=1}^{n} p\{C_i\}\log_2\left[p\{C_i\}\right] \ . \tag{3}$$

This definition of entropy is in agreement with the intuitive feeling that entropy should become larger as the disorder in the system increases. The reason for extracting the probabilities based on the number of unary associations directed *towards the class* and not those leaving the class is best illustrated by an example: Assume that a given system is extended by adding one new class. The unary associations towards the new class, imply changes to the attributes (pointers or references) of the existing classes, and as such are related to the flexibility of the system. On the other hand, if we had selected the unary associations leaving from the new class, all changes would refer to the new class, whose code would be added anyway.

The maximum value for entropy is obtained when all classes "connect" to all classes (including themselves), implying a system with the maximum possible coupling between system components.

For such a system with total disorder, the total number of all possible unary associations (between discrete classes) is given by the 2-permutations of a set with n classes:

$$\#associations = P(n,2) = \frac{n!}{(n-2)!} = n(n-1) \ . \tag{4}$$

Each class can also have a reflexive association, which in the worst case, can also be binary. (Consider for example a *Professor* class, where each professor can advise other professors but can also be advised by other professors).
Consequently, to the number of total associations, we have to add the number of self-associations (counting one for each direction) for all possible classes. Therefore:

$$\#total\_associations = n(n-1) + 2n = n(n+1) \ . \tag{5}$$

As a result, the probabilities for such a fully interconnected class diagram will be equal to:

$$p\{C_i\} = p_{eq} = \frac{n}{n(n+1)} = \frac{1}{n+1} \ . \tag{6}$$

The probabilities in this case are equal, as expected from the definition of entropy in eq. (2).
This uniform distribution will cause entropy to obtain its maximum value:

$$H_D^{\max} = -n \cdot p_{eq} \ \log_2\left(p_{eq}\right) = \frac{\log_2(n+1)}{n+1} \cdot n \ . \tag{7}$$

Consequently, the maximum entropy of a system with n classes tends to infinity as $n \to \infty$.

For simplicity, we assume first that the functionality of the design is enhanced by the addition of a new class. A "good" OO-design implies that the addition of the new class will influence as few of the existing classes as possible. Otherwise, if the initial part of the design is to fully "embrace" the new class by adding associations between almost all of the existing classes and the new one, then obviously there is a significant

degradation of the quality. This quality degradation is captured by the notion of entropy, which in this case, will obtain a significantly larger value for the upgraded design.

The difference between the entropy value of the initial and the new design not only indicates the deterioration of the design (for a well-designed system this difference should be small), but can also serve as a measure of the number of generations of the design. Assuming a low entropy value for the initial design (corresponding to the first generation) and having an estimate (from past data) for the entropy increase per generation, it is possible to calculate the number of generations elapsed during the development of the system. Moreover, if the entropy approaches its maximum possible value, this fact indicates whether the design has reached a saturation level, beyond which it is relatively difficult to improve the quality of the design, and the system has to be re-designed from scratch.

The proposed entropy metric does not consider so far one of the most important features of object-orientation, namely inheritance. The presence of a generalization relationship between two classes, is a form of a static association. The introduction of a unary association directed from class *A* to class *B*, implies a modification to the attributes and methods of class *A*. However, the presence of a generalization relationship, where class *A* is the parent class and *B* the descendant (the arrow in a class diagram pointing to A), implies a modification only to the added class *B* and not to the existing class *A*. Therefore, generalizations are not counted during the extraction of the entropy metric when determining the corresponding probabilities.

4 Results

We believe that software metrics that assess any aspect of quality in an object-oriented software system, should be evaluated against appropriate Design Patterns [10]. Since there is a general agreement on the fact that Design Patterns improve the quality of an object-oriented design, such a qualitative improvement should be measurable by the applied metrics. Any of the metrics from the plethora of suggested ones that does not validate the usefulness of well-established Design Patterns should possibly be discarded.

To this end, we apply the following methodology for evaluating the proposed entropy metric against one Behavioral and one Structural Design Pattern, namely the *Strategy* and *Composite* [10]. Testing against these two patterns will also be valuable in evaluating whether the proposed entropy metric can be of help in enforcing the *Open-Closed* and the *Liskov Substitution Principle* [4], which both patterns help to apply. We observe the value of entropy for successive generations of an object-oriented system, both for a "naïve" design as well as for a more sophisticated design employing design patterns. The initial design is for both products the same. Each new set of requirements calls for a new generation of the design; However, during the upgrade, the "good" design is redesigned employing appropriate design patterns.

Our primitive application is an adaptation of the infamous "Shape" Application used in many OOD books. Initial and further requirements are listed below:

- initial requirement: *application must be able to draw circles*
- second requirement: *application must be able to draw squares as well*
- third requirement: *application must be able to draw complex shapes consisting of several squares*
- fourth requirement: *application must be able to draw complex shapes consisting of both squares and circles*

We first follow the evolution of the "naïve" design. The class diagram of the first design includes, except for the *Client* class, a *Circle* class, which the Client *uses*. In respond to the second requirement, the designers choose to add a second class *Square*, which the Client also uses. Unfortunately, the new class also imposes changes to the code of the Client class. The third requirement is handled by adding a *Complex* class which a) uses the Square class and b) is used by the Client. Since the Client must be able to draw complex shapes as well as primitive squares, the initial association between *Client* and *Square* should remain. Finally, the fourth requirement enforces the programmers to change the code of the Complex class to insert an association to the Circle class as well. The UML class diagram for each generation of the software design is shown in Figure 1.

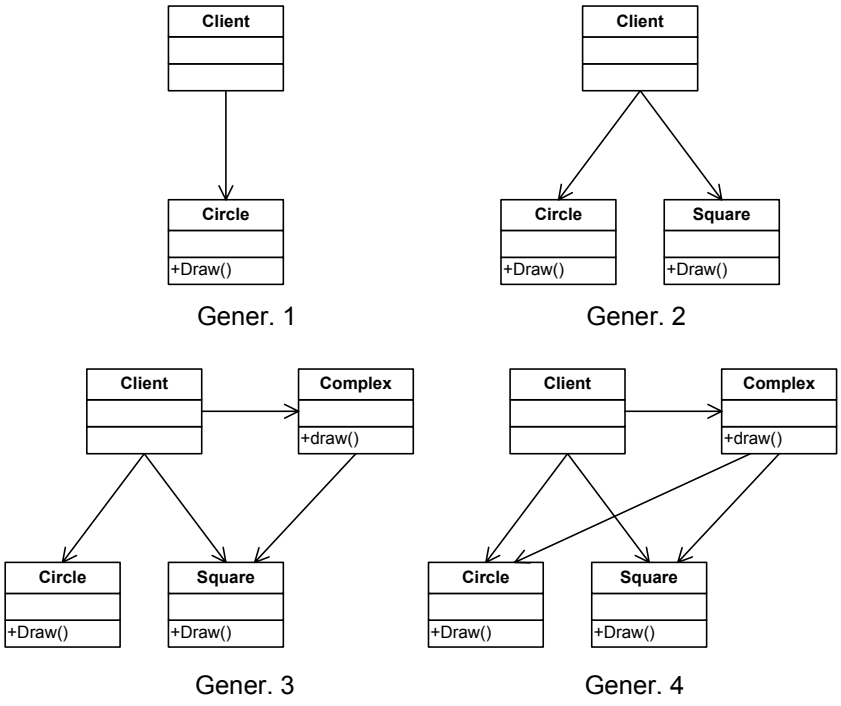

Fig. 1. Successive generations of "naïve" design

On the other hand, the improved design employs design patterns to avoid changing the code, each time a new requirement is implemented (Open-Closed Principle [4]). To accommodate the initial requirement the development team will obviously come up with the same solution as in the previous case and consequently the class diagram for the first generation is the same. However, when the second requirement arises, the designers identify an *axis of change* acknowledging that several shapes can be requested. Therefore, to close function *Draw* against different shape types, a kind of "shape abstraction" is required. This calls for the *Strategy Pattern,* which according to the GoF [10] "lets an algorithm vary independently from clients that use it". The successive generations for the improved design are illustrated in Figure 2.

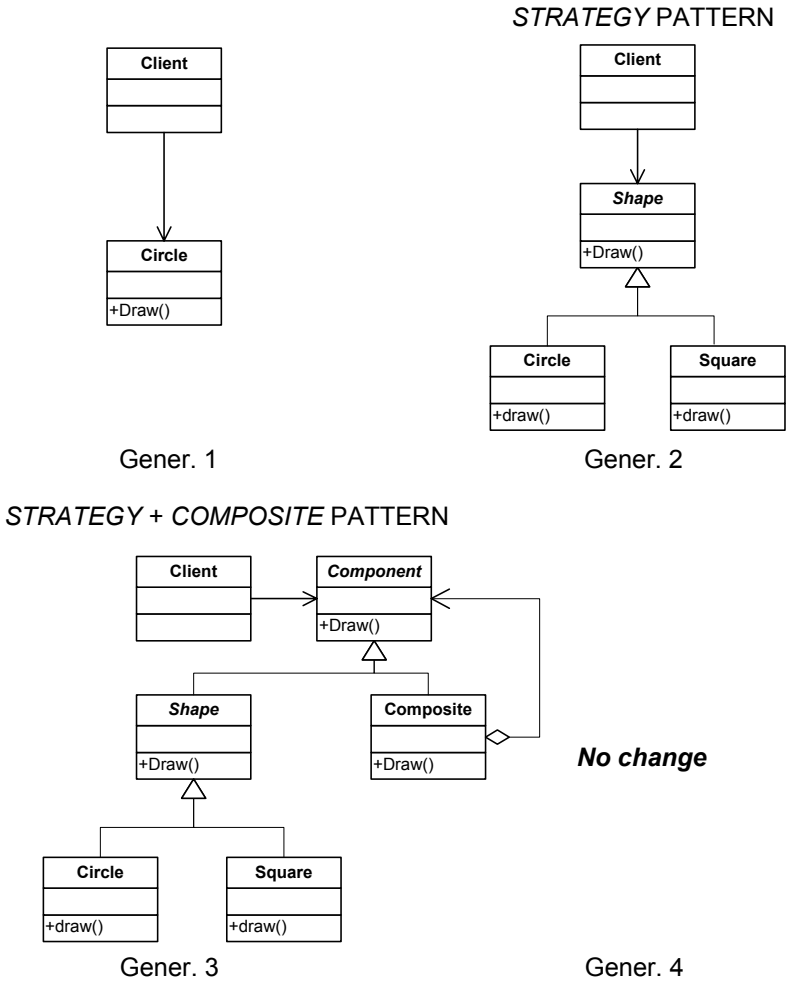

Fig. 2. Successive generations of improved design

The third requirement is not handled by simply adding a Complex class, since this solution would not follow the Open-Closed principle against different shapes contained in this class. Moreover, in order to allow drawing both primitive as well as composite objects, the design would require additional associations. To improve the design, the development team opts for the *Composite Pattern*, which lets clients treat individual objects and compositions of objects uniformly, eliminating unnecessary associations. The solution is shown in the third generation in Figure 2. Once the Strategy and Composite pattern are applied, there is no need to change to design in order to accommodate the fourth requirement: It can be directly implemented based on the class diagram of the third generation, without any modification of existing classes!

The application of the entropy metric to each generation of both designs reveals the following results:

"Naïve" Design:
Generation 1: $H_D = 0$
Generation 2: $H_D = 1$
Generation 3: $H_D = 1.5$
Generation 4: $H_D = 1.521928$
The final entropy value is obtained from the following probabilities:
$p(Client) = 0$, $p(Complex) = 1/5$, $p(Circle) = 2/5$, $p(Square) = 2/5$,

Improved Design:
Generation 1: $H_D = 0$
Generation 2: $H_D = 0$
Generation 3: $H_D = 0$
Generation 4: $H_D = 0$

As it can be observed, the entropy values are in agreement with the fact that an object-oriented design employing Design Patterns is better structured. In the "naïve" design the entropy value increases from each generation to the next and since the addition of each new class "spreads out" the probabilities even more, the entropy value will continue to increase.

On the other hand, in the improved design, not only the entropy value has a lower final value and does not change between successive generations, but the addition of any new shapes to the final design will not change further the entropy value, indicating the stability and flexibility of the system.

5 Conclusions

The merits of object-oriented systems concerning ease of reuse, maintenance, extensibility, scalability are well understood and drive the wide acceptance of object-orientation among software developers. However, it is generally difficult to differentiate "good" from "bad" designs, since object-orientation is rather a skill than a set of guidelines. In this paper, an information theoretic entropy metric has been proposed

for evaluating the flexibility of an OO design and the ease in extending a system's functionality.

The metric is based on the assumption that the entropy of the next generation of an object-oriented system with enhanced functionality remains close to its previous level in case the added functionality affects only a limited number of existing classes. Results for successive generations of two designs, where one of them is better structured employing design patterns, validates the proposed metric against two popular design patterns.

References

1. Basili, V. R., Briand, L. C., Melo, W. L.: A Validation of Object-Oriented Design Metrics as Quality Indicators. IEEE Trans. on Software Engineering, Vol. 22, (1996) 751-761
2. Chidamber S. R., Kemerer, C. F.: A metrics suite for object oriented design. IEEE Trans. on Software Engineering, Vol. 20, (1994) 476-493
3. Dumke, R., Foltin E.: Metrics-based Evaluation of Object-Oriented Software Development Methods, Preprint Nr. 10, Fakultät für Informatik, Universität Magdeburg (1997)
4. Martin, R. C.: Agile Software Development: Principles, Patterns and Practices, Prentice Hall, Upper Saddle River, NJ (2003)
5. Bansiya, J., Davis, C. G.: A Hierarchical Model for Object-Oriented Design Quality Assessment. IEEE Transactions on Software Engineering, Vol. 28, (2002) 4-17
6. Dvorak, J.: Conceptual Entropy and Its Effect on Class Hierarchies. IEEE Computer, Vol. 27, (1994) 59-63
7. Snider, G.: Measuring the Entropy of Large Software Systems. HP Technical Report, HPL-2001-221, Sept. 2001 www.hpl.com/techreports/2001/HPL-2001-221.pdf
8. Papoulis, A.: Probability, Random Variables, and Stochastic Processes, 2nd ed. McGraw-Hill, New York (1984)
9. Rumbaugh, J., Jacobson, I., Booch, G.: The Unified Modeling Language Reference Manual, Addison-Wesley, Reading, MA (1999)
10. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, Boston, MA (1995)