Object Oriented Design Knowledge: Ontology and Measurement of Impact

Javier Garzás[1] and Mario Piattini[2]

[1] ALTRAN SDB Senior Consultant - Projects Engineering Research Group
C/ Ramírez de Arellano, 15. 28043, Madrid - Spain
jgarzas@altransdb.com
[2] Alarcos Research Group
Escuela Superior de Informática - University of Castilla-La Mancha
Ronda de Calatrava, s/n. 13071, Ciudad Real – Spain
Mario.Piattini@uclm.es

Abstract. It has been a long time since appeared of the Object Oriented (OO) paradigm. From that moment, the designers have accumulated much knowledge in design and construction of OO systems. Patterns are the most refined OO Design Knowledge. However, there are many others kinds of knowledge than not yet classified and formalized. We distinguish and classify the following categories: principles, heuristics, patterns and refactorings. In this paper, we propose an Ontology for Object Oriented Design Knowledge and a measure of impact for patterns, two key elements to create a method based in knowledge.

1 Introduction

By the middle of the 90´s the first catalogue of patterns was published (Gamma *et al.*, 1995). The motivation of the authors of the catalogue and of the community that investigates patterns has been to transfer the Object Oriented Design Knowledge (OODK) accumulated during years of experience. However, more knowledge exists apart from that related to patterns and this other knowledge is frequently "hidden". We denominate, distinguish and classify the following categories in OODK: principles, heuristic, patterns and refactorings (Garzás and Piattini, 2001). But there is much uncertainty with the previous elements, and this have never been studied as a whole, neither them compatibility has been studied nor a method based in this knowledge exists.

For principles the main contributions are Liskov and Zilles (1974), Meyer (1988), Gamma *et al.* (1995) and Martin (1995 & 1996), but the study of design principles is limited, leading to them being used in an isolated way or even being ignored. In Garzás and Piattini (2001) a classification and definition of Object Oriented Design Principles (OODP) is shown in a more extensive way, any examples of principles are the Open-Closed Principle (OCP), Dependency Inversion Principle (DIP), Default Abstraction Principle (DAP), Don't Concrete Superclass Principle (DCSP), etc.

Z. Bellahsène, D. Patel, and C. Rolland (Eds.): OOIS 2002, LNCS 2425, pp. 153-159, 2002.

Nevertheless, the work of discovering, formalizing and classifying principles, an activity that we denominate "Principles Mining", is not still finished and it is one of our current investigation lines. With regard to heuristics the only explicit papers to which we can refer are those of Riel (1996) and Booch (1996). Refactoring techniques are characterized by their immaturity, although it is true to say that this topic is rapidly gaining acceptance, the main works in this area are Kent Beck and Fowler's (2000), Tokuda and Batory (2001) and Opdyke (1992). Patterns are, without doubt, the most refined OODK. The application of patterns in OO began in the late eighty and was consolidated by the work of (Coad, 1992), Gamma *et al.* (1994), Buschmann *et al.* (1996), Fowler (1996) and Rising (1998). However, at the present time, when patterns are used several types of problems can occur (Wendorff (2001) and Schmidt (1995)): Difficult application, Difficult learning, Temptation to recast everything as a pattern, Pattern overload, Ignorance, Deficiencies in catalogues, etc.

In this paper, we will show two necessary work lines to create a method based in OODK: we will analyze in more detail the knowledge elements, we will show an OO Design Knowledge Ontology and we will show a measure of Impact of the patterns use.

2 Object Oriented Design Knowledge Ontology

Before beginning an elaborated study, it is necessary to have a definition for each knowledge element. We consider following definitions for Heuristic and Principle (and we consider anyone of the definite in the existent literature for Patterns and Refactorings, for example, Gamma *et al.* (1995) and Tokuda and Batory, (2001)):

- Heuristic: Pertaining to the use of the general knowledge gained by experience, sometimes expressed as "using a rule-of-thumb". Argument derived from experience.
- Principle: a set of proposals or truths based on experience that form the foundation of OOD and whose purpose is to control this process. A principle is a fundamental rule or law that govern a Design in a given situation. Characteristic than desirably the design must not violate and that provide of quality to the design.

We need some way of characterizing the OODK, and we will create an OODK Ontology for it. An ontology describes domain knowledge in a generic way and provides agreed understanding of a domain. In Gruber (1991)'s words: *ì I use the term ontology to mean a specification of a conceptualization. That is, an ontology is a description of the concepts and relationships that can existî* . There are four types of Ontologies (Jurisica *et al.*, 1999): *Static Ontologies* (encompass static aspects of an application), *Dynamic Ontologies* (dynamic aspects within an application), *Intentional Ontologies* (describe the world of things agents) and *Social Ontologies* (describe social settings in terms of social relationships). According to the previous, our OODK Ontology is in the "static" category and we use the UML for expressing this Ontology. The figure 1 shows this OO Design Ontology. The OODK ontology

showed on figure is semi-formal, since it is not described in a language with formal semantics, theorems and proofs of such properties as soundness and completeness.

Garzás and Piattini (2001) explicitly detail the possible relationships between principles and patterns. They define three types of relationships:

- *Type 1*, the pattern contributes a good solution to the resulting model of the application of the principle ("from the principle towards the pattern").
- *Type 2*, the pattern completes or contains the principle.
- *Type 3*, the principle can improve a solution to which a pattern has been previously applied ("from the pattern towards the principle").

Previous relations and types are detailed in the figure, in the relation between Principle and Patter entities. We may observe as the entities Principle and Pattern can be introduced in the design for Refactorings (here Refactoring Patterns would be reflected). The entities Principle and Pattern are made out of Heuristics (this are "micro good practices").

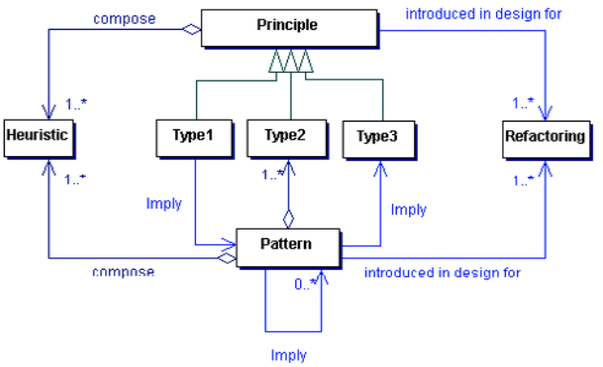

Fig.1. Ontology of Object Oriented Design Knowledge

3 Measure of Impact of the Patterns Use

According to ISO 9126 standard (ISO, 1999), maintainability is subdivided into Analyzability, Changeability, Stability, Testability and Maintainability Compliance. If we obtain a correct OO design, we will obtain a best maintenance, more cheaply and operating. Considering 9126 standard, exist two important parameters for OO Design:

- *Changeability* allows that a design may change easily, important requirement at the time of extend functionality into existing code.
- *Analyzability* allow us understand the design. This is an essential requisite to be able to modify the design in a realist period of time.

When patterns are applied to a software design two opposites forces appear which are related directly with maintainability: we have changeable solutions, but we have the inconvenience that the solution once was obtained can be very complex, and this does

that the design be less analyzable. Thus appears a curious relation between Changeability and Analyzability: If we increase the design's Changeability then we will decrease the design's Analyzability, and vice versa.

The figure 2 shows graphically the relationship between Changeability and Analyzability when patterns are applied. The Breaking Point determine the "Optimal Patterns Number", where is the best maintenance in relation to Patterns.

Obviously, we have a problem: how much Changeability can we apply in order not to lose the design's Analyzability? Obtaining a metrics to answer previous question would be a great contribution.

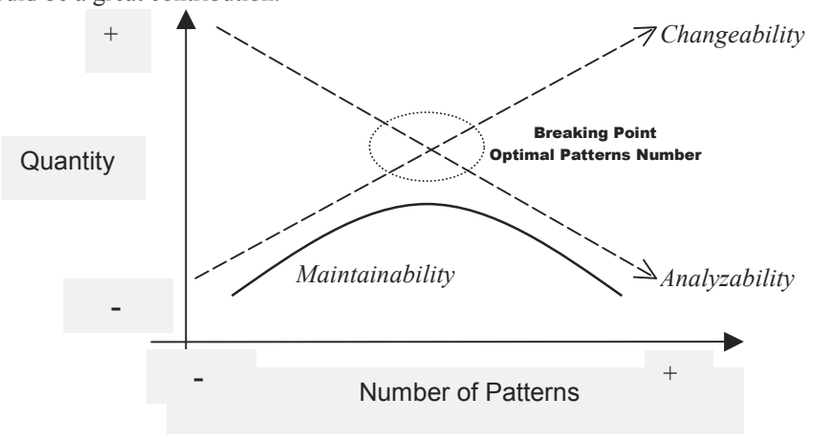

Fig.2. Relationship between Changeability and Analyzability

3.1 The Indirection: The Heart of Design Patterns

Nordberg (2001) comment as "at the heart of many design patterns is an indirection between service provider and service consumer. With objects the indirection is generally via an abstract interface". Unfortunately each level of indirection moves the software farther from the real world or analysis level view of the problem and deeper into relatively artificial mechanism classes that add overhead to both design comprehension and implementation debugging. With respect to the previous, we have observed the following:

- Every time that a pattern gets introduced, at least, an indirection into the design appear and these elements are not of the domain or business logic, such as notifies, observer classes, updates methods, etc.
- Every time that we add an indirection the software moves around further off of the analysis. To the adding indirections or design classes the design becomes less semantic, less comprehensible and less analyzable.
- And every time that an indirection is added this one increases the design changeability or flexibility.

3.2 A Metric for Optimal Patterns Number

With all the previous, we can define a parameter than quantify how of flexible or changeable a design is in relation to indirections:

$$\text{Changeability Number (CN)} = \text{Indirection Classes Number (ICN)} \quad (1)$$

A value that measure design's analyzability must consider the number of design classes introduced. These are of two groups:

- Simplify classes, reusing, such as the subject class into observer pattern.
- Indirection classes, such as the observer class into observer pattern.

Thus:

$$\text{Analyzability Number (AN)} = \text{Domain Classes Number (DCN)} - \text{Indirection Classes Number (ICN)} - \text{Simplify Classes Number (SCN)} \quad (2)$$

We may observe in the last formula, as when we have an analysis diagram its semantics is in the topmost point. When we introduced in the designing phase artifacts on the analysis diagram the model's semantics decrease. We also may observe, as certain patterns will have a larger impact in the semantics than other ones, depending on the classes that the patterns introduce. Now, we may calculate the Optimal Patterns Number (OPN) as follow:

$$\text{Changeability Number (CN)} = \text{Analyzability Number (AN)} \quad (3)$$

$$\text{Indirection Classes Number (ICN)} = [\text{Domain Classes Number (DCN)} - \text{Simplify Classes Number (SCN)}] / 2 \quad (4)$$

Considering in the previous formula that the DCN parameter is a fixed value in design phase. The rest of parameters depend of the kind of pattern or design artifact introduced.

Acknowledges

This research leaves of the DOLMEN project supported by CICYT (TIC 2000-1673-C06-06). We want to thank to ALTRAN SDB the support given to this research.

4 Conclusion and Future Projects

The experts always have used proven ideas. It is in the last years when these ideas, materialized in the pattern concept have reached their biggest popularity and diffusion. Although over recent years different areas of knowledge related to the construction of OO system such as principles, heuristics, patterns and refactoring techniques have been consolidated, but there is a lot of work still to be done in order to systematize and offer this OODK to designers in such a way that it can be easily used in practical cases. However, we still have a considerable amount of work to do.

Our final aim is to offer a detailed systematization of principles, heuristics, patterns and refactoring techniques (together with their respective interrelationships), which will facilitate their application for the designer. Empirical validation of the proposed metrics is also required.

References

1. Booch G. (1996). Managing the Object-Oriented project. Addison-Wesley
2. Buschmann F., Meunier R., Rohnert H., Sommerlad P. and Stal M. (1996). A System of Patterns: Pattern-Oriented Software Architecture, Addison-Wesley.
3. Coad P. (1992, Septiembre). Object-Oriented Patterns. *Comunications ACM, v*ol. 35, n 9, pp. 152-159.
4. Fowler M. (1996). Analysis Patterns: Reusable Object Models. Addison-Wesley
5. Fowler M. (2000). Refactoring improving the design of existing code. Addison Wesley
6. Gamma E., Helm R., Johnson R. and Vlissides J. (1995). *Design patterns: Elements of Reusable Object Oriented Software.* Addison-Wesley.
7. Garzás J. and Piattini M. (2001, Agosto). Principles and Patterns in the Object Oriented Design. En Wang Y., Patel S. and Johnston R.H. (Eds.), *OOIS 2001, 7th International Conference on Object-Oriented Information Systems* (pp. 15-24). University of Calgary, Calgary, Canada: Springer.
8. Gruber, T. (1991). The Role of a Common Ontology in Achieving Sharable, Reusable Knowledge Bases. *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning*. Cambridge.
9. ISO (1999). ISO 9126 – Software Product Quality.
10. Jurisica I., Mylopoulos J. and Yu E. (1999, Octubre). Using Ontologies for Knowledge Management: An Information Systems Perspective Knowledge: Creation, Organization and Use. *Proceedings of the 62nd Annual Meeting of the American Society for Information Science (ASISi99).* Washington, D.C. pp. 482-296.
11. Liskov B. H. and Zilles S. N. (1974). Programming with Abstract Data Types*., Computation Structures Group*, n 99, MIT, Project MAC, Cambridge Mass.
12. Martin R. C. Engineering Notebook. C++ Report 1996; Aug-Dec (published in four parts)
13. Martin R. C. Object Oriented Design Quality Metrics: An analysis of dependencies. ROAD 1995; Vol. 2, Nº 3
14. Meyer B. (1988). Object Oriented Software Construction. Prentice Hall.
15. Nordberg M. E. (2001, October). Aspect-Oriented Indirection – Beyond OO Design Patterns. *OOPSLA 2001, Workshop Beyond Design: Patterns (mis)used*. Bahía Tampa, Florida, EEUU.
16. Opdyke W. (1992). Refactoring OO frameworks. PhD Thesis, Department of Computer Science. University of Illinois.
17. Riel A. J. (1996). *Object-Oriented Design Heuristics*. Addison-Wesley.
18. Rising L. (1998). The Patterns Handbook: Techniques, Strategies, and Applications, Cambridge University Press.

19. Schmidt D. C. (1995, Octubre). Experience Using Design Patterns to Develop Reusable Object-Oriented Communication Software. *Communications of the ACM,* 38,10, pp 65-74.
20. Tokuda L. and Batory D. (2001). Evolving Object-Oriented Designs with Refactorings. *Kluwer Academic Publishers - Automated Software Engineering*, vol8, Nº1, pp 89-120
21. Wendorff P. (2001). Assessment of Design Patterns during Software Reengineering: Lessons Learned from a Large Commercial Project., *CSMR 2001 - European Conference On Software Maintenance And Reengineering,* pp 77-84.