Object Oriented Design Knowledge

JAVIER GARZÁS
ALTRAN SDB Consultant
Projects Engineering Research Group
ALTRAN SDB
C/ Ramírez de Arellano, 15. 28043,
Madrid - SPAIN
jgarzas@altransdb.com

MARIO PIATTINI
Alarcos Research Group
Escuela Superior de Informática,
University of Castilla-La Mancha
Ronda de Calatrava, s/n. 13071,
Ciudad Real - SPAIN
Mario.Piattini@iuclm.es

Abstract: Nowadays, due to experience acquired during years of investigation and development of Object Oriented systems, numerous techniques and methods that facilitate their design are available to us. In this article we present a compilation of the object oriented design principles, as well as an initial analysis of the design patterns and their relationship with these principles and as this relationship can facilitate a new base for the study, comparison and application of patterns.

Key Words: Objects Oriented Design Patterns, Principles, Heuristics, Refactoring.

1 Introduction

By the middle of the 90's the first catalogue of patterns was published (Gamma *et al.*, 1995). From that moment, 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. Since then, designers have been reading and using patterns and a strong community centered in relation to patterns has consolidated over the years.

The step of classifying and formalizing OODK has led to an enormous advance in the construction of OO Systems, which assures us that the line of investigation being followed in OO Knowledge is both very beneficial and promising (evidence of this are the themes presented for discussion in the last editions of the most important congresses, such as, OOPSLA 2001 Workshop Beyond Design: Patterns (mis) used).

Patterns are, without doubt, the most refined OODK, and proof of this is the existence of numerous publications on the theme. The application of patterns in OO began at late eighties (Beck 1987 or Coad 1992) and was consolidated by the work of Gamma *et al.* (1995), Buschmann *et al.*(1996), Fowler (1996) and Rising (1998). However, at the present time the exclusive use of patterns is not sufficient to guide a design in a formal way, the designer's experience is still necessary. In fact, according to our own observation of developers, Wendorff (2001) and Schmidt (1995), several types of problems occur:

- Difficult Application.
- Difficult Learning.
- Temptation to Recast everything as a pattern
- Pattern overload.

- Ignorance.
- Deficiencies in catalogues: Search and Complex Application, High Dependence of the Programming Language, Comparatives, etc.

Most of the previous problems arise from a lack of classification, cataloguing and formalization of OODK, where, at present, there isn't well-defined and homogeneous catalogue in which their systematic applicability is detailed.

On the other hand, more knowledge exists apart from that related to patterns, although it would be true to say that this other knowledge is frequently "hidden". Other solutions exist for certain problems or design faults that have occurred repeatedly for years and which are not related to patterns. These other solutions are known by the most experienced designers and they are also based on experience, these are object oriented knowledge, too.

For example, let us imaginea class with many clients and a general interface for all these clients. An experienced designer would realize that if a change takes place in one of the methods of the interface all its clients would be affected, and this would force us to make a new compilation even in the case of clients of the interface that never use the method that has changed. The solution to the previous problem is not detailed in a pattern. However, as is the case with the problems that patterns solve, this problem is recurrent and old. Obviously, the solution to this problem is specifically detailed, in what we denominate "OO design principle" (OODP) and more specifically the *Interface Segregation Principle (ISP)*, whose essence is simple: *in the case of a class with several clients, we must create specific interfaces for each client and multiple inheritance from them regarding the class.*

Therefore, as we could see in the previous example, more categories exist within OODK and not only patterns. We denominate, distinguish and classify the following categories in OODK: principles, heuristic, patterns and bad smell – refactorings. Therefore, the application of solutions and the anticipation of problems like the one that was detailed previously is much more expensive and restricted, as there is general confusion about the difference between these concepts and moreover, they have not received the same attention nor do they present a similar degree of maturity.

In fact, with the exception of the contributions of Liskov (1974), Meyer (1988), Gamma *et al.* (1995) and Martin (1995 & 1996), knowledge of design principles is limited, leading to them being used in an isolated way or even being ignored[1]. In Garzás and Piattini (2001) a summary, classification and definition of OODPs is shown in a more extensive way, where they underline, among others, the following principles:

[1] Books such as that of James Martin (not to be confused with Robert C. Martin, previously cited) titled "Principles of Object Oriented Analysis and Design" do not in reality analyse construction principles of OO systems but rather present concepts, techniques and methods for OO analysis and design.

Open-Closed Principle (OCP): A module should be open for its extension and closed for its modification
Dependency Inversion Principle (DIP): Depend upon abstractions. Do not depend upon specifications.
Default Abstraction Principle (DAP): Put an abstract class between an interface and their implementations
Black Box Principle (BBP): Favour the object composition over class inheritance.

Substitution Principle (SP): The subclasses must be substitutable by their base classes
Interface Segregation Principle (ISP): Many client specific interfaces are better than one general purpose interface.
Interface Design Principle (IDP): "Program" an interface, not an implementation.

Don't Concrete Superclass Principle (DCSP): Avoid maintaining concrete super classes.

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.

Regarding OO design heuristics the only papers to which we can refer are those of Riel (1996) and Booch (1996). For example, we can mention the following ones (Riel, 1996):

All data should be hidden within its class.

Minimize the number of messages in the protocol of a class.

Eliminate classes that are outside the system.

Do not change the state of an object without going through its public interface.

Eliminate irrelevant classes from your design.

A class must know what it contains, but it should never know who contains it.

Lastly, *bad – smells* and refactoring techniques are characterized by their immaturity, although it is true to say that this topic is rapidly gaining acceptance, largely thanks to Kent Beck and Fowler's (2000) work.

In essence, the problem confronting the designer is how to articulate all this explicit knowledge and to apply it in an orderly and efficient fashion in the OOD. In relation to this, Ralph Johnson remarks about patterns: *"For one thing, the large number of patterns that have been discovered so far need to be organized. Many of them are competitors; we need to experiment and find which are best to use.[...] Analyzing existing patterns, or making tools that use patterns, or determining the effectiveness of patterns, could all be good topics"* (Johnson, 2001).

In this paper we will analyze in more detail the main relationships among the different categories of OODK, providing a first approach to their applicability. In the following section we lay down the main bases for formalizing the application, cataloguing and connection of OODK. Lastly the basic outlines of a method will be sketched for the application of OODK. In section 3 we present our conclusions and future projects.

2 Foundations of an Object Oriented Design Method Based on Knowledge

2.1 Main Characteristics of the Different OODKs

As a first approach to the existing relationship among the previous types of knowledge, we can synthesize the previous ideas in the following table.

	Heuristic	Principle	Pattern	Refactoring
Abstraction	+++	+++	++	+
Maturity	++	++	+++	+
Systematization	+	+	++	++
Formalization	+	+	+++	++
Automation	+	+	++	+++

Table 1. Main Characteristics of the Different OODKs

Heuristics and Principles are the OODKs of most abstraction, while the element of least Abstraction is refactoring, because this requires a high level of detail and its application is very specific. Without a doubt the most mature element is the pattern, whilst Refactoring is the least mature element. Systematization is maximum in Patterns and Refactorings, and in fact catalogues exist for both, an aspect that also converts them into the most popular knowledge. Lastly, as far as Automation is concerned or the degree to which these concepts are integrated in CASE tools, the Refactorings are the most automated elements, because their nature means that they can be more easily automated and in fact the number of these tools is growing remarkably (see, for example, www.refactoring.com).

2.2 The Applicability of the different divisions of Knowledge

Once we have distinguished the different OODK categories we can begin to advance more and study their applicability. We should distinguish several intervention levels when actions can be carried out in an OOD and relate each category of knowledge to its corresponding level. It is important to clarify the division of intervention levels in an OOD because in the future this will be fundamental for defining a design method based on knowledge and this division will form the group of phases of the method, with its input - output.

When we study a design we can observe: The relationships among entities (this is denominated Inter-modulate Level) or The entities in a specific and isolated way without keeping in mind the interconnections or relationships between them (Modular level). On the other hand, each one of the previous levels can be observed from different abstraction or detail levels. In Fowler (1997) and Cook & Daniels (1994) three perspectives are detailed to observe a class diagram: Conceptual or Essential, Specification and Implementation.

Having defined the intervention or study levels of an OOD we can delimit where it is applied or where each one of the categories of knowledge works. The following table is obtained:

what knowledge is applicable to what study level		Heuristics	Principles	Patterns	Refactoring
Conceptual	Modular	X			
	InterModular	X	X	X	
Specification	Modular	X			
	InterModular		X	X	
Implementation	Modular				X
	InterModular				X

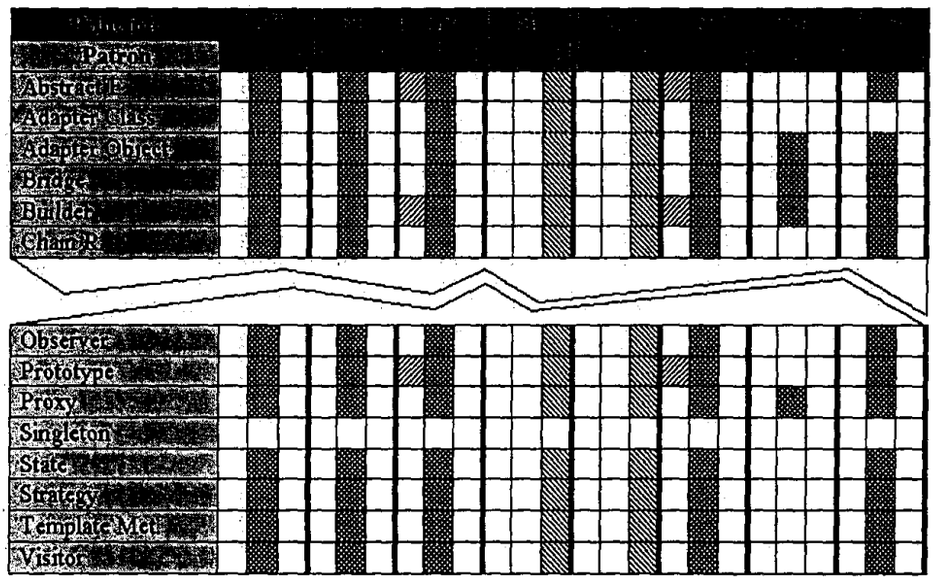
Table 2. Applicability of the different divisions of Knowledge

2.3 Implicability between Knowledges

If we observe carefully the car we can observe certain implicability among themselves. For example, in order to conform to the DIP, one of the strategies could be to use the abstract Factory pattern. The purpose of other patterns such as Prototype, Factory method, etc. is more to perform the Abstract Factory than to directly conform to a principle. Therefore, we can conclude that there are patterns that directly allow a principle to be complied with, whilst other patterns are more related to patterns than to principles. Consequently, patterns could be classified according to the principles they follow. The principles would even enable us to create a different catalogue of patterns to that currently existing (in most cases they are simply presented in alphabetical order). Checklists of principles could also be drawn up which assess the design and offer us solutions patterns that ensure that they are complied with. We can specify more and considering their relationship with the patterns, the principles can contemplate one or several of the following types:

- *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").

The next table shows an analysis of the principles mentioned in previous epigraphs and their relationship with each pattern of the detailed by Gamma in function of the previous types. We can observe as the relationship of patterns has been ordered alphabetically, we can obtain this way an objective order, later on, and based on the principles, we will be been able to obtain analogies.



Several considerations, uses and investigation lines can be extracted, some examples are the following ones:

- It allows to break down in forces of smaller grain each one of the patterns, facilitating the study of elements common to all the patterns of oneself character: "patterns in the patterns" or "meta-patterns".

- It allows to guide the use of patterns, since it is easier to know how to apply in a correct way a principle that a pattern and once applied the principle is easy to arrive to the pattern. This facilitates us the pattern's good use, in their fair measure, without abuse. For example, the use of NSCP implies us to use creational patterns and this assures us that our system is written in function of interfaces and not in function of implementations.

- It allows a formal study of micro architectures.

- It allows to obtain the forces (principles) that conform the pattern and how depending in its way of incidence in the pattern (type 1, 2 or 3) this can be of different character, examples:

  - We can observe as Abstract Factory, Builder, Factory Method and Prototype maintain an almost identical kernel of principles while Singleton doesn't complete any principle. Singleton is not a micro architecture (it only describes a class), Singleton treats the creation of objects but he doesn't make it with the same character and the same abstraction that the other four creational patterns, an Idiom considers it. With regard to the four remaining creation patterns, we observe that they complete the same principles except Builder, since this has the same character that the previous ones but by means of a composition strategy. As we see the study

of the principles that intervene in a pattern allows us, among other many things, a finer and based classification.

- We observe as any micro architecture with some hierarchy that we want to consider design pattern should complete (type 2), at least, the following principles: OCP, SP, DIP, IDP and DCSP.

- We observe that in patterns structurally identical as State and Strategy the same principles are completed and with the same character.

- All pattern that completes OCP, SP, DIP, IDP and DCSP in type 2, ISP and DAP in type 3 and BBP don't contemplate it is classified (according to Gamma's book) as of behavior.

▪ We will be able to look for and/or to validate new design patterns observing if they complete certain of meta-patterns.

Well, we can generalize the application of the previous types to cover all the categories of OODK. Table 3 defines the existing relationships among the categories of OODK. Reading the table from rows to columns, "Type 1" means that having once applied the corresponding OODK in the rows the solution obtained almost necessarily or obligatorily implies the corresponding OODK in the column. On the other hand, "Type 2" means that the essence of the OODK in the rows contains or completes the corresponding OODK in the columns.

Implication of use		Heuristics	Principles	Patterns	Refactorings
Heuristics	Type 1				
	Type 2				
Principles	Type 1				
	Type 2				
Patterns	Type 1				
	Type 2				
Refactorings	Type 1				
	Type 2				

Table 3. Implicability between Knowledges

We can see that the table is not symmetrical... a refactoring would imply the use of a pattern but a pattern would not imply the use of refactoring.

2.4 Applying OODK

We can fuse the previous conclusions, synthesized in the tables 2 and 3, in the figure 1. Without entering in a level of very low detail, the general idea of a method based on knowledge is sketchy. The arrows in the figure represent implicability

relationships (Type 1). This way, we can observe that if we begin the application of OODK from the first Design stages (Conceptual and Specification) the categories of applicable OODK are, although not necessarily in this order: Heuristic, Principles and Patterns. The application of Principles can, in many cases, imply the use of other Principles or of other Patterns, something that also happens with Patterns. On the other hand, when, either for the evolution of the Design or for a re-engineering process, we are confronting a Design at implementation level, refactorings are the applicable OODK. These refactorings could lead us to improve or to re-design aspects at an abstraction level,or they could take us to the Conceptual - Specification levels. Heuristics are completed or contained (Type 2) in the others OODK.

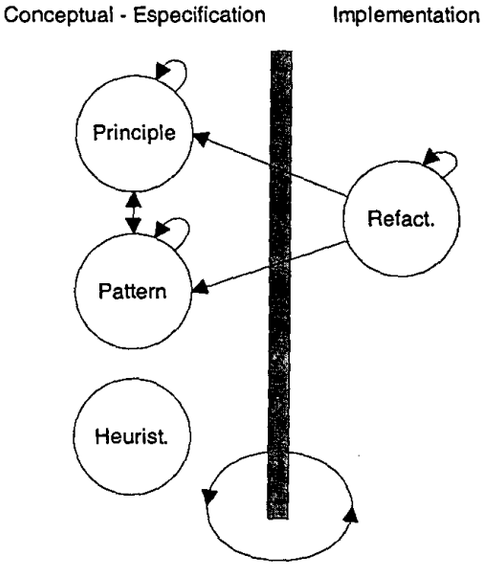


Figure 1. Appliying OODK

The idea is to achieve a method able to guide and classify the application of OODK and once it has been detailed in more depth, (objective of our current line of investigation), facilitate its automation (CASE). The study of the relationships and classes of OODK have as their goal the building of an Objects Oriented Design Knowledge Base (OODKB) that supports the CASE, being able to infer or advise developments in OOD

3 Conclusions and Future Projects

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, we believe that there is a lot of work still to be done in order to systematize and offer this knowledge to OO designers in such a way that it can be easily used in practical cases. In fact, up until now the different studies that have been published present these elements in a disconnected way, which at times makes

their application more difficult. This very problem occurs when choosing a pattern and incorporating it into an existing model.

In this article we have presented a compilation of the OO principles that we consider to be most important, as well as an initial analysis of the OO design patterns proposed by Gamma *et al.* (1995) and their relationship with these principles. However, we still have a considerable amount of work to do both in the compilation of more principles and in the study of other patterns.

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.

4 Acknowledgements

This research leaves of the DOLMEN project supported by CICYT (TIC 2000-1673-C06-06).

5 References

Beck K. Cunningham W. *Using Pattern Languages for Object-Oriented Programs.* OOPSLA 1987, workshop on the Specification and Design for Object-Oriented Programming.

Booch G., *Object Solutions. Managing the Object-Oriented project,* Addison-Wesley,1996

Buschmann F., Meunier R., Rohnert H., Sommerlad P. and Stal M., *A System of Patterns: Pattern-Oriented Software Architecture*, Addison-Wesley, 1996.

Coad P., "Object-Oriented Patterns", *Comm.* ACM, Vol. 35, No 9, Sep. 1992, pp. 152-159.

Cook S., Daniels J., *Designing Object Systems: Object-Oriented Modeling with Syntropy,* Prentice Hall, 1994.

Fowler M., UML *Distilled*, Addison-Wesley, 1997.

Fowler M., *Analysis Patterns: Reusable Object Models*, Addison-Wesley, 1996.

Fowler M., *Refactoring improving the design of existing code,* Addison Wesley, 2000.

Gamma E., Helm R., Johnson R. and Vlissides J., *Design patterns: Elements of Reusable Object Oriented Software*, Addison-Wesley, 1995.

Garzas J., Piattini M. *Principles and Patterns in the Object Oriented Design,* 7th International Conference on Object-Oriented Information Systems (OOIS '01). August 27-29, 2001, University of Calgary, Calgary, Canada.

Henderson – Seller B., Eduards, J. M. "The Object-Oriented System Life Cycle", *Comm.* ACM, Vol. 33, No 9, Sep. 1990, pp. 142-159.

Johnson, R. (2001). Personal Communication.

Liskov B. H., Zilles S. N., *Programming with Abstract Data Types,* Computation Structures Group, Memo N° 99, MIT, Project MAC, Cambridge Mass, 1974.

Martin R. C., *Engineering Notebook,* C++ Report, Aug-Dec, 1996. (Publicated in 4 parts).
Martin R. C., *Object Oriented Design Quality Metrics: An analysis of dependencies,* ROAD, Vol. 2, N° 3, Sep – Oct, 1995.
Meyer B., *Object Oriented Software Construction,* Prentice Hall, 1988.
Noda N., Kishi T., *On Aspect-Oriented Design: An Approach to Designing Quality Attributes*, Sixth Asia Pacific Software Engineering Conference 7 - 10 December, 1999 Takamatsu, Japan.
Priestley M., *Practical Object Oriented Design with UML,* McGrawHill, 2000.
Prieto, Diaz, "Status Report: Software Reusability", *IEEE Software,* Mayo 61-66, 1993.
Riel, A. J. *Object-Oriented Design Heuristics*, Addison-Wesley, 1996.
Rising L., *The Patterns Handbook: Techniques, Strategies, and Applications*, Cambridge University Press, 1998.
Schmidt D. C., *Experience Using Design Patterns to Develop Reusable Object-Oriented Communication Software*, Communications of the ACM 38,10, October 1995, pp 65-74.
Shull F., Melo W. L., and Basili V. R., *An Inductive Method for Discovering Design Patterns from Object-Oriented Software Systems*, College Park, MD: University of Maryland, 1996.
Wendorff P., "Assessment of Design Patterns during Software Reengineering: Lessons Learned from a Large Commercial Project", *Procedings of the Fifth European Conference on Software Maintenance and Reengineering* , CSMR 2001, IEEE Computer Society.