Chapter XIV

Improving OO Design Process Using Rules, Patterns and Refactoring

Javier Garzás
mCentric, University Rey Juan Carlos, Spain

Mario Piattini
University of Castilla-La Mancha, Spain

Abstract

In recent years different areas of knowledge related to the construction of object-oriented (OO) designs 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 knowledge to OO designers in such a way that it can be easily used in practical cases. In order to clarify this, we have defined an ontology of OO Micro Architectural Design Knowledge and the foundations of an OO design method based in the knowledge.
Many authors (Shaw, 1990; McConnell, 2003) have commented on the need for defined chunks of knowledge in the software engineering field. In this regard, the software engineering community has advanced greatly in recent years, and we currently have much accumulated knowledge: standards, methodologies, methods, metrics, techniques, languages, patterns, processes, concepts, and so forth. Nevertheless, the field of software engineering is still beset by a lack of structured and classified chunks of knowledge, and not all knowledge is transmitted, accessible, or studied in the same way.

One example of this lack of structured and classified knowledge is the Object-Oriented (OO) Micro Architectural Design. Object-oriented knowledge is popularized in different forms—principles, heuristics, patterns, refactoring, lessons learned, defects, best practices, and so forth—but the difference between these concepts is generally unclear, and moreover, not all of them have received the same amount of attention or have reached the same degree of maturity.

In this sense, we find the OO design principles; in this field, there are several important contributions, such as Meyer (1988), Helm, Johnson, and Vlissides (1995), or Martin (1996) (Table 2 shows examples). Regarding OO design heuristics, the main works to which we can refer are Riel (1996) and Booch (1996) (Table 3 shows examples). On the other hand, bad smells and refactorings are rapidly gaining acceptance, thanks to Fowler (2000) and Beck and Fowler’s (2000) work. Finally, patterns are the elements that have undergone the greatest evolution; proof of this is the numerous publications on

Table 1. Examples of OO principles

| Dependency Inversion Principle (DIP): Depend upon abstractions. Do not depend upon specifications. | Interface Segregation Principle (ISP): Many client-specific interfaces are better than one general-purpose interface. |
| Do not Concrete Super class Principle (DCSP): Avoid maintaining concrete super classes. | Interface Design Principle (IDP): “Program” an interface, not an implementation. |

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.
Related Content

User Requirements Validation and Architecture Discovery through Use Case Invariants and Model Animation
www.igi-global.com/chapter/user-requirements-validation-architecture-discovery/25635?camid=4v1a

Using Linguistic Patterns to Model Interactions
www.igi-global.com/chapter/using-linguistic-patterns-model-interactions/27358?camid=4v1a

A Framework Based on Design Patterns: Implementing UML Association, Aggregation and Composition Relationships in the Context of Model-Driven Code Generation
www.igi-global.com/chapter/framework-based-design-patterns/27359?camid=4v1a

Software Reuse and Object Technology
Jane Fedorowicz and Denis Lee (2000). Object Oriented Technologies: Opportunities and Challenges (pp. 77-88).
www.igi-global.com/chapter/software-reuse-object-technology/27348?camid=4v1a