Design Patterns – education and classification challenge

Mario Konecki, Tihomir Orehovački, Dragutin Kermek
Faculty of Organization and Informatics
University of Zagreb
Pavlinska 2, 42000 Varaždin, Croatia
{mario.konecki, tihomir.orehovacki, dragutin.kermek}@foi.hr

***Abstract**. Design patterns have been recognized as a way to simplify and unify applications development process. There are two problems among others that can be identified when talking about design patterns.*

First there is a problem in education of novices about patterns and training them to fully understand how to find the right pattern for some problem. Second problem is classification and location of patterns that would make it easier to find the right pattern for the right problem and right technology.

There are a lot of patterns created almost every day and there are a lot of books that mention some classifications and a number of patterns but there is still a lack of some more practical classification that would enable one to find the right pattern easily.

In this paper we give a brief overview of and discussion about problems associated with education of novices. We give a brief view at patterns and the most common classification of patterns and we comment some commonly used technologies in combination with design patterns. We also discuss two mentioned problems along with possible ideas for their resolution.

Keywords. design patterns, education, novices, classification

1 Introduction

Business world today has become very demanding regarding the huge number of applications that are used as a support for various business processes. There is a need to increase the speed and the degree of automatization of applications development process. One of possible approaches to this problem is usage of design patterns.

Design patterns have been recognized as a mean that could be useful in solving some fundamental problems that occurred inside of object-oriented paradigm such as problems of designing and reusing applications code and libraries. There were a growing number of people supporting this belief ([2], [3], [4], [6], [7], [8], [10], [12], [17], [20]).

The main idea behind design patterns is to support the reuse of design information, thus allowing developers to communicate more effectively [23].

Regarding these new paradigms and technologies, the following questions have arisen [23]:

- Amongst the many different design patterns that are being discovered, are any related to each other? What are the characteristics of such a relationship?
- Do two patterns address a similar problem area?
- Is it possible to combine two design patterns?
- What are the criteria for classifying design patterns into categories?

As we can see, the main questions are how to recognize all existing and developing patterns that are emerging, how to find the right pattern for the right problem and how to understand connections between patterns.

There are a few publications that refer to this question and the most important and familiar is the Design Patterns: Elements of Reusable Object-Oriented Software [11] that introduced the design patterns in the first place. Although the classification given in this book is very detailed and answers some of the mention questions there is still a lack of understanding about connections and relationships between patterns and how to find the right pattern for some particular problem and to be sure that that very pattern is the best for the job. In this paper one possible solution to these unanswered questions will be presented.

Another problem that has emerged is education of novices about these new concepts, namely about design patterns. There is a need to recognize the problems that occur in this type of education and to refer to its possible solutions. Answers to these questions are presented in the second part of this paper.

2 Design patterns

Design pattern is a general and reusable solution to some problem that is common in the area of software design (problem-solving and planning for a software solution). It is a template that tells how to solve some particular problem and can be use in a variety of situations.

Design pattern is general in nature, not specific which means that it cannot be translated directly into program code. It shows objects, relationships, interactions, etc. that lead the developer to code particular solution for some occurring situation. There are also other types of patterns such as architectural patterns but design patterns deal exclusively with the problems of software design. Generally, it can be said that design patterns are true and tried solutions to reoccurring software design problems under a particular context.

The very idea of patterns originated as an architectural concept by Christopher Alexander (1977-1979). Later, Kent Beck and Ward Cunningham started to explore the idea of patterns usage in programming. Their idea was presented later in the annual ACM conference OOPSLA in 1986. In 1994 GoF (Gang of Four) - Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides wrote a book titled Design Patterns: Elements of Reusable Object-Oriented Software [11] which had a great influence on the popularity of design patterns.

Design patterns provide tested and reliable solution to commonly occurring problems and eliminate the possibility of some major problems that can occur when developer design on their own without any guidelines. Using design patterns makes it easier to understand program code to everyone that is familiar with the particular design that is used.

In spite of many advantages there is some criticism of design patterns.

Some of the critics and problems are:

- Inclusion of additional levels of indirection
- Unlike components, patterns don't provide reuse
- Some argue that design patterns don't differ significantly from other forms of abstraction

3 Design patterns classification

In the book "Design Patterns: Elements of Reusable Object-Oriented Software" [11] there is a defined catalog which contains about 20 design patterns. These patterns are grouped into 3 groups (Creational Patterns, Structural Patterns and Behavioral Patterns).

The structure of all design patterns mentioned in [11] is given in the figure 1 [23].

All patterns and their relationships are defined through documentation that consists from following parts as defined in [11]:

- Design patterns are described in documentation that makes it easy to use them. It describes the context in which the pattern is used, elements and relationships inside of context that the patterns deals with and tried to resolve and also some particular solutions suggestion.
- This documentation consists of several sections. They are not always the same but there is one commonly format that is described in the book Design Patterns: Elements of Reusable Object-Oriented Software which was mentioned before. Of particular interest are the Structure, Participants, and Collaboration sections which describe a prototypical micro-architecture that developers copy and adapt to their particular designs to solve the recurrent problem described by the design pattern.

All sections of this format of documentation are given in [11].

In this sections structure, there is a part called Related Patterns that describes possible relationships between patterns. Design patterns catalogue, as described in [11], shows classification that is based on two criteria: jurisdiction (class, object, compound) and characterization which is already mentioned (creational, structural, behavioral). These relationships are described in detail and every one of them appears to be different and this is the fact that brings to conclusion that there is a need to classify these relationships in some matter to ease one in finding right pattern(s) for some specific domain of work.

4 Classification of design patterns relationships

In order to classify design patterns relationships the main issues that have to be addressed had to be identified. Two main issues can be recognized in order to create an adequate relationship classification: direction of relationship (unidirectional, bidirectional) and strength of relationship. To efficiently address these issues a classification between pairs of design patterns (X, Y) that states the nature of these issues was formed [23].

This classification includes the following categories [23]:

- X uses Y in its solution – this means that when X builds a solution one part of the problem is similar to problems that are solved by Y
- X is similar to Y – X and Y address a similar type of problem
- X can be combined with Y – X can be combined with Y to solve a problem

This classification of relationships between pairs of design patterns is shown in figure 2 [23].

By using this classification, a developer can better decide what patterns to use. He can see which patterns are similar and which of them are the best for some particular problem. He can also find the best pattern to solve some subproblem looking at the patterns that the main patterns X can use. Another advantage is the possibility of combining different patterns that can be combined in order to solve some more or less complex problem for which none of the patterns alone has the right answer.

5 Combined classifications

When looking at described classifications there is a room for further improvement. One general combined classification that uses all existing and adds some new ideas could be composed. What is obvious is that this classification should address both details of each pattern and also each type of relationships between them also in detail. As a first level groups of patterns should be used as units. They should be described with all possible aspects and relationships between then should be stated and explained.

In this way user would easily, as it would in the next lower levels, find where to look for pattern for some problem but also where to find similar patterns or patterns that can be combined and joined in solving a problem. So this level would be the first step in searching for complete problem solution. The types of relationships that would denote links between pattern groups (that are virtually the same as the ones between patterns) could be the following (taking in consideration that A is the first group of design patterns and B the second group):

- A pattern(s) uses B pattern(s) in its solution
- A has similar pattern(s) to all (some) patterns from B
- Pattern(s) from A can be combined with pattern(s) from group B

These relationships could be divided into even more detailed way. This is discussed below.

At the second level we look at patterns. Each pattern should also be explained in detail with all relevant aspects. Along with this each relationships between patterns has to be defined and well described in order to make tracking for all necessary patterns for some particular situation as easy as possible.

Types of relationships that can be found here are a more detailed version of those used between pattern groups. Relationships that would be used in this case are (X represents pattern 1 and Y pattern 2):

- X might/must use Y in its solution in degree (strength graduation) Z
- X is similar to Y in degree Z
- X can be combined with Y in degree Z

Using these 2 levels of a general classification could be made that would enable user to find each of patterns easier and in this way it would also make the possibility of wrong choices and solutions less likely. This all is just another challenge that will probably be addressed in the future research.

5 Design patterns in education

Another problem mentioned is the problem of education about design patterns. Popularity of design patterns in software industry established a need to incorporate design patterns in higher education institutions. Although the need was recognized there was no consensus on what patterns to include in this education or what level of knowledge to provide in educational programs of this kind. All that was agreed upon was that early exposure to design patterns and its principles would benefit the students [5], [9], [16], [21].

Various learning techniques and models were proposed in the last years in literature and in various conferences [9], [14], [19], [22]. The aim to find the best techniques for learning about design patterns became a goal of some conferences like OOPSAL, ECOPS and its mirror in Europe. These conferences focused on finding the best examples for educational purposes [1].

When looking at all these efforts a need has occurred to perform an exploratory study that would show what are concrete problems in higher education that arise when talking about design patterns. Researchers Della, Clark and Wick believed that the abstract nature of patterns makes them difficult to comprehend among novices [9], [22]. It was also noticed that the difficulty to comprehend design patterns is connected to lack of knowledge and experience about object-oriented technology [21].

Descriptions of patterns as given in [18] have been frequently criticized as to simple and incomplete, not representing real situations in real world [15], [22]. As notices, two main problems that novices encounter when dealing with design patterns are [13]:

- lack of experience in designing software
- not encountering enough recurring design problems
- not enough experience with translating design pattern to code for implementation

Based on all this a need occurred for proper study that would investigate the causes of problems among novices in learning design patterns.

A research of this kind was done in [13] and as a result a list of most common problems in education of novices about patterns was formed. These problems are:

- Selection-based errors
- Application specific errors

Selection-based errors refer to capability of novices to identify applicable patterns to some particular problem.

Application specific errors are divided into three sub-categories:

- Mapping of pattern participants
- Misrepresentation errors
- Incomplete design

Mapping of pattern participants refers to process of incorporating design patterns into the initial class diagram. The mapped or generated class diagram should match that of the suggested structure.

Misrepresentation errors are a common errors when one represents an interface or an abstract class as a concrete class or does some other similar misrepresentation.

Incomplete design means that one has left out some operations or even classes for overall solution design.

All this points out that a proper leadership and guidance, as well and new and improved models and examples of patterns learning, are needed and this topic today is more actual that ever.

If we look carefully at each of problems mentioned we can see that we need a complete and overall approach to whole educational process in order to solve them. What needs to be incorporated into mentioned educational process is a set of steps of which each has to be addressed in great detail.

The first step is to include proper lections and exercises about software design. There is a great importance for novices to learn the basics of designing as it is the very first level in understanding the whole concept of design patterns. After this first step is achieved in satisfactory manner the next step can be undertaken.

The next step would be exposure of novices to a large series of example problems that would present them a very need to use something as design patterns. In this step they would experience various design problems and goal would be for them to conclude that there is a fair amount of recurring design problems that need to be unified and solved in one place that is one pattern.

Finally, the novices would be presented with concepts of design patterns as a way to solve recurring problems that they encountered in previous step. Along with understanding the concepts of design patterns novices need to be exposed to a large number of examples in which they need to connect proper design pattern to some problem and also they need to master the whole development process from choosing a design to completely implementing it trough concrete computer code.

These steps would as such address all detected problems that occur among novices regarding education about design patterns. The key aspect in all steps are examples and then more examples as experience is the only true way to go in order to achieve complete understanding of theoretical and practical aspect of working with design patterns.

6 Conclusion

In this paper two problems were considered: a problem of a more practical classification that would enable easier finding of right pattern for the right problem and a problem of educating novices about patterns. Both of these problems are very actual and need some practical solutions.

Regarding the first problem a classification of design patters relationships has been presented as a mean to understand patterns better and to find right pattern(s) easier by thinking about their relationships rather than just of patterns themselves. For the second problem a most reoccurring problems with patterns education were mentioned and discussed. This is just a basis for more research on this field and more practical solutions that are so needed in this particular field.

References

[1] Alphonce, C., Casperson, M., Decker, A.: **Killer "killer examples" for design patterns**, Proceedings of the 38th SIGCSE technical symposium on Computer science education, March 7 – 11, Covington, Kentucky, USA, 2007, pp. 228-232.

[2] Beck, K.: **Patterns and software development**, Dr. Dobb's Journal, Vol. 19, No. 2, 1993, pp. 18 - 23.

[3] Beck, K., Johnson, R.: **Patterns generate architecture**, Proceedings of the 8th European Conference on Object-Oriented Programming, July 4 - 8, Bologna, Italy, 1994, pp. 139-149.

[4] Buschmann, F.: **Rational architectures for object-oriented software systems**, Journal of Object-Oriented Programming, Vol. 6, No. 5, pp. 30–41., 1993.

[5] Clancy, M., Linn, M.: **Patterns and Pedagogy**, ACM SIGCSE Bulletin, Vol. 31, No. 1, pp. 37-42., 1999.

[6] Coad, P.: **Object-oriented patterns**, Communications of the ACM, Vol. 35, No. 9, pp. 153–159., 1992.

[7] Coplien, J.: **Advanced C++: Programming Styles and Idioms**, Addison - Wesley, Reading, MA, 1993.

[8] Coplien, J.O.: **Generative pattern languages: An emerging direction of software design**. C++ Report, No. 6, 1994.

[9] Della, L., Clark, D.: **Teaching Object-Oriented Development with Emphasis on Pattern Application**, Proceedings of the Australasian conference on Computing education, Melbourne, Australia, 2000, pp. 56-63.

[10]Gamma, E., Helm, R., Johnson, R., Vlissides, J.: **Design patterns: Abstraction and reuse in object-oriented designs**, Proceedings of the 7th European Conference on Object-Oriented Technology, July 26-30, Kaiserslautern, Germany, 1993, pp. 406–431.

[11]Gamma, E., Helm, R., Johnson, R., Vlissides, J.: **Design Patterns: Elements of Reusable Object-Oriented Software**, Addison-Wesley, 1995.

[12]Johnson, R.: **Documenting frameworks using patterns**, Proceedings of Conference on Object Oriented Programming Systems Languages and Applications, October 18 – 22, Vancouver, British Columbia, Canada, 1992, pp. 63 – 76.

[13]Masita, A. J., Shahrul, A. M. N.: **The difficulties of Using Design Patterns among Novices: An Exploratory Study**, Proceedings of fifth International Conference on Computational Science and Applications, August 26-29, Kuala Lumpur, Malaysia, 2007, pp. 97-103.

[14]O'Cinneide, M., Tynan, R.: **A Problem-Based Approach to Teaching Design Patterns**, Annual Joint Conference Integrating Technology into Computer Science Education, Leeds, United Kingdom, 2004, pp. 80 - 82.

[15]Ouyang, Y.: **Explaining Design Patterns Through One Application**, Proceedings of 32nd ASEE/IEEE Frontiers in Education Conference, November 6 – 9, MA, USA, 2002, pp. 6-11.

[16]Porter, R., Calder, P.: **Patterns in Learning to Program – An Experiment?**, Proceedings of the sixth conference on Australasian computing education, Dunedin, New Zealand, 2004, pp. 241 - 246.

[17]Pree, W.: **Meta-patterns: A means for describing the essentials of reusable o-o design**, Proceedings of the 8th European Conference on Object-Oriented Programming, July 4 - 8, Bologna, Italy, 1994, pp. 150-162.

[18]Reed, D.: **Incorporating problem-solving patterns in CS1**, Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education, February 26 – March 01, Atlanta, Georgia, USA, 1998, pp. 6-9.

[19]Schmolitzky, A.: **A laboratory for teaching OO language and design concepts with teachlets**, Proceedings of Conference on Object Oriented Programming Systems Languages and Applications, October 16 – 20, San Diego, California, USA, 2005.

[20]Shaw, M.: **Heterogeneous design idioms for software architecture**, Proceeding of the Sixth International Workshop on Software Specification and Design, Software Engineering Notes, October 25 – 26, Como, Italy, pp. 158–165.

[21]Weir, S. B.: **An investigation of significant technical factors affecting the learning and using of design patterns**, Master Thesis, Utah State University, USA, 2006.

[22]Wick, M. R.: **Teaching Design Patterns in CS1: A Closed Laboratory Sequence based on the Game of Life**, Proceedings of the 36th SIGCSE technical symposium on Computer science education, February 23-27, St. Louis, Missouri, USA, 2005, pp. 487-491.

[23]Zimmer, W.: **Relationships between design patterns**, Pattern languages of program design, ACM Press/Addison-Wesley, 1995.

Appendix A.

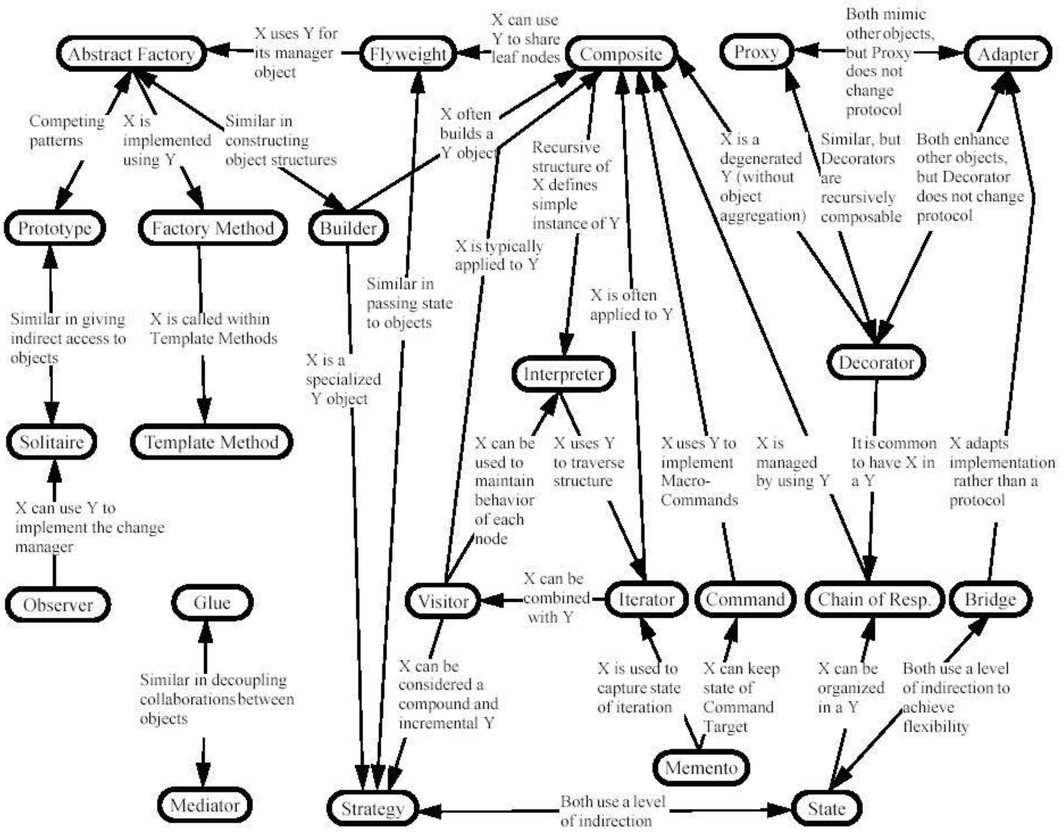
Abstract Factory
X uses Y for its manager object
Flyweight
X can use Y to share leaf nodes
Composite
Proxy
Both mimic other objects, but Proxy does not change protocol
Adapter
Competing patterns
X is implemented using Y
Similar in constructing object structures
X often builds a Y object
Recursive structure of X defines simple instance of Y
X is a degenerated Y (without object aggregation)
Similar, but Decorators are recursively composable
Both enhance other objects, but Decorator does not change protocol
Prototype
Factory Method
Builder
X is typically applied to Y
Similar in passing state to objects
X is often applied to Y
Similar in giving indirect access to objects
X is called within Template Methods
X is a specialized Y object
Interpreter
Decorator
Solitaire
Template Method
X can be used to maintain behavior of each node
X uses Y to traverse structure
X uses Y to implement Macro-Commands
X is managed by using Y
It is common to have X in a Y
X adapts implementation rather than a protocol
X can use Y to implement the change manager
Observer
Glue
Visitor
X can be combined with Y
Iterator
Command
Chain of Resp.
Bridge
Similar in decoupling collaborations between objects
X can be considered a compound and incremental Y
X is used to capture state of iteration
X can keep state of Command Target
X can be organized in a Y
Both use a level of indirection to achieve flexibility
Memento
Mediator
Strategy
Both use a level of indirection
State

Figure 1. Design patterns catalogue structure

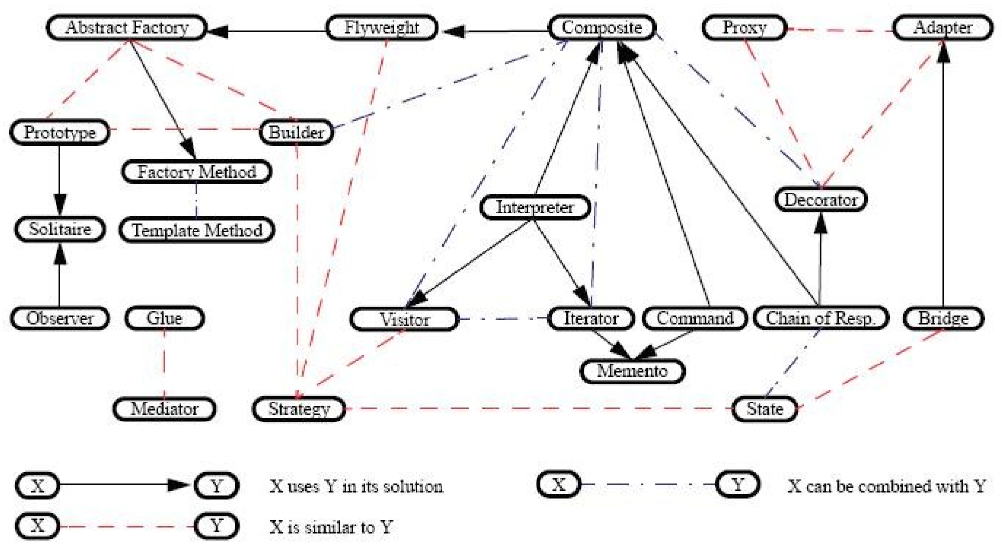

Figure 2. Design patterns relationships classification