Category: Software & Systems Design

Patterns in the Field of Software Engineering

Fuensanta Medina-Domínguez
Carlos III Technical University of Madrid, Spain

Maria-Isabel Sanchez-Segura
Carlos III Technical University of Madrid, Spain

Antonio de Amescua
Carlos III Technical University of Madrid, Spain

Arturo Mora-Soto
Carlos III Technical University of Madrid, Spain

Javier Garcia
Carlos III Technical University of Madrid, Spain

INTRODUCTION

In the mid 1960's, the architect Christopher Alexander (1964) came up with the idea of Patterns, as "a solution to a problem within a defined context" and developed this concept. He explains, in a very original way, his ideas of urban planning and building architecture, using patterns to explain the "what", "when", and "how" of a design.

Alexander invented a Pattern Language that is the fundamental to good building and city designs, and describes it in a collection of repetitive schemas called *patterns*.

In Computer Science, software is susceptible to conceptual patterns. Consequently, Ward Cunningham and Kent Beck, used Alexander's idea to develop a programming pattern language composed of five patterns as an initiation guide for Smalltalk programming. This work was presented at the Object-Oriented Programming, Systems, Languages & Applications Conference (OOPSLA) in 1987.

In the early 1990's, Erich Gamma and Richard Helm did a joint research that resulted in the first specific design patterns catalog. They identified four patterns: Composite, Decider, Observer, and Constrainer patterns.

According to many authors, OOPSLA '91 highlighted the evolutionary process of design patterns. The synergy between Erich Gamma, Richard Helm, Rala Johnson, and John Vlissides (better known as the "Gang of Four" or GoF) and other reputable researchers (Ward Cunningham, Kant Beck or Doug Lea) definitively launched the study of and research into Object Oriented Design Patterns.

At the same time, James Coplien, another software engineer, was compiling and shaping a programming patterns catalogue in C++, which was a significant advance in the implementation phase in software development. Coplien's catalog was published in 1991 under the title "Advanced C++ Programming Styles and Idioms".

Between 1991 and 1994 the concept of pattern design was discussed at international congresses and conferences. All of these encounters culminated in OOPSLA '94. The GoF took advantage of this event to present their compilation (Gamma, Helm, Johnson & Vlissides, 1995). This publication, considered at that time as the best book on Object Orientation, compiled a 23-pattern catalog, founding the basis of patterns design.

The number of pattern-related works, studies and publications in general, but especially in design, has exponentially grown since. However, the different research groups being born must be cataloged into three fundamental paradigms:

- Theoretical approximations to the software pattern design concept and pattern languages. Coplien's work (Coplien, 1996; 2004; Coplien & Douglas, 1995) stands out in this field.
- Analysis and compilation of software applications design patterns. Rising's efforts (Rising, 1998; 2000) and Buschmman (Buschmann, Meunier, Rohnert, Sommerlad & Stal, 1996; Buschmann, Rohnert, Stal & Schmidt, 2000) are included in this classification.
- The study of special purpose patterns, like antipatterns (Brown, 1998).

As has been explained, the pattern concept has clear origins, and an important value as a reuse tool. The main

problem is that the word pattern has been used almost for everything, thus losing its original meaning. The goal of this work is to go back to the definition of patterns and present how software engineering is working with this concept.

The remainder of this chapter is structured as follows. Section two provides both, formal and informal definitions of pattern as well as the formats used to describe them. Section three presents a classification of existing patterns in the field of software engineering. In section four, the authors describe their conclusions and present the future trends in section five. A selection of key terms is defined at the end of the chapter.

BACKGROUND

Pattern Definition

The knowledge and use of pattern improves communication between the designer and the developer. According to Erich Gamma et al. (1995):

"Designers know that you do not have to solve each problem starting from scratch….you must reuse solutions which previously worked. When you find a good solution, you must use it continuously. This experience makes you an expert."

Although software engineers knew about design patterns, it was a tremendous boost for them when design patterns were systematized and categorized by four engineers Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, known as the Gang of Tour (GoF):

They schematized 23 software design patterns through templates and used the Unified Modeling Language (UML) to describe them. They also provided examples of implementation written in Smalltalk and in C++. These patterns were later written in oriented-object language such as Java (Cooper, 1998).

FORMAL DEFINITION OF SOFTWARE PATTERNS

In this section, the most significant definitions of the software pattern have been gathered.

The first ever definition, is the one proposed by Alexander:

"A recurring solution to a common problem in a given context and system of forces" (Alexander, 1979).

Less literary but more concrete is the one proposed by Riehle and Zullighoven (1996):

"A pattern is the abstraction from a concrete form which keeps recurring in specific non-arbitrary contexts."

Nevertheless, the most precise one many authors followed is that of Gabriel (1998):

"Each pattern is a three-part rule, which expresses a relation between a certain context, a certain system of forces which occurs repeatedly in that context, and a certain software configuration which allows these forces to resolve themselves."

Gabriel defined concepts that make up the terminology of software patterns:

- Forces System: a set of objects and restrictions that have to be satisfied by the application, such as portability, flexibility, reuse, and so forth.
- Software Configuration: a set of design rules to be applied to solve the problem forces.

James Coplien (1996) enumerated the requirements that a "good" pattern has to carry out:

- Solve a problem: the patterns capture solutions, not principles or strategy.
- Provide tested solutions: the patterns show neither theories nor speculations. Simple solutions are not provided.
- Describe a relation: the patterns describe systems, structures and mechanism. They do not provide a simple module.
- Have a human component: the best patterns have to be useful.

PATTERN DESCRIPTION

Patterns must be described formally so that their content is available to all. A pattern format is a template with sections, a formal structure that eases learning, comparison among other patterns and their use. There are different formats for describing patterns such as: the Alexander, GoF and canonical formats.

The Alexander Format

Alexander explained his patterns, in a narrative style, in terms of problem to be solved, described the context in which the pattern is applied and the proposed solution. So, each Alexander's pattern is described according to the following elements:

- Name
- Problem
- Context
- Forces
- Solution