Teaching Design Patterns

Article · June 2010

CITATIONS
9

READS
581

1 author:

N. Pillay
University of Pretoria
117 PUBLICATIONS **595** CITATIONS

SEE PROFILE

Teaching Design Patterns

Nelishia Pillay
School of Computer Science
University of KwaZulu-Natal
KwaZulu-Natal, South Africa
+27 33 2605644
pillayn32@ukzn.ac.za

ABSTRACT
Design patterns form part of most Computer Science (CS) undergraduate curricula. Research has shown that design patterns are both difficult to learn and teach. This paper presents the results of a survey conducted to identify the difficulties experienced by students learning design patterns for the first time. In addition to this it discusses teaching methodologies that can be used in the instruction of a course on design patterns to assist students to grasp the concepts more easily. Given the teaching and learning difficulties identified in the study, the paper proposes a way forward.

Categories and Subject Descriptors
K.3.2 [**Computer and Information Science Education**]: Computer Science Education

General Terms
Experimentation.

Keywords
Design patterns, learning difficulties, teaching methodologies.

1. INTRODUCTION
Design patterns play a prominent role in software design in industry and are thus an essential component of the CS undergraduate curriculum. The philosophy behind design patterns is to document the experiences of software developers in designing software systems and reuse solutions previously found to design solutions to new problems. The main aim of design patterns is to produce code that can be reused and is easily extensible ([1], [2], [3], and [4]). Students find design patterns difficult to learn [4]. Design patterns are also difficult to teach [5]. The paper firstly describes the difficulties experienced by students in learning design patterns in section 2. A survey was conducted in a third year Software Design course to identify such learning difficulties. The paper then describes teaching methodologies for the instruction of design patterns in section 3 Design pattern projects are described in section 4. Section 5 examines the incorporation of design patterns in other Computer Science courses. Given the learning and teaching difficulties identified in the study, section 6 proposes a way forward.

2. LEARNING DIFFICULTIES
According to Clancy [6] and Wick [7] students find design patterns complex and difficult to learn. In order to identify the difficulties experienced by students a survey was conducted in a third year Software Design course. The course forms part of a three-year B.Sc degree with a major in Computer Science. Twenty two students were enrolled for the course. The design patterns component was taught over six weeks. The students were given three lectures and one three-hour practical per week. During lectures each design pattern was presented in theory followed by an example of the implementation. These were reinforced by exercises which the students had to work through during the practical. The students also had to complete a semester project. The project required them to implement the game Sudoku. As part of the design of the project they had to describe which design patterns they would use, substantiate their choice of patterns, and specify how they would be implemented.

The text book used for the section on design patterns is "Head First Design Patterns" [8]. This book was chosen due to the instructional methodology incorporated into presenting the design patterns. The book presents design patterns in a light and fun manner which facilitates the learning process. Students found the textbook to be "very interesting" and "text book examples were easy to follow and fun".

At the end of the section on design patterns the students were required to complete a questionnaire which asked them what difficulties they experienced with learning design patterns, what difficulties they experienced with the design pattern practicals and to rate the design patterns from most difficult to learn to least difficult. The main difficulty experienced by students was determining when to use which design pattern. Some of the students also stated that design patterns were difficult to understand and it took a while to grasp the different concepts. Students also felt that the use of design patterns increased the complexity of the design and code which led to confusion in some cases. A majority of the students experienced problems with understanding the *iterator* design pattern while they experienced the least difficulty with the *singleton* design pattern. The practicals basically required students to implement the design patterns studied in lectures for different problems, thus they were told which design patterns to use.

Students generally found the practicals helped them to understand design patterns as a result of implementing them. The project required them to decide on which patterns to use. Students felt that the gap between the practicals and the project was too big, and that practicals should include exercises requiring them to determine which design patterns to apply. The main aim of this study was to ascertain the learning difficulties experienced by students and thus there was no evaluation of the approach taken in teaching the course.

3. TEACHING DESIGN PATTERNS

According to Preiss [9] design patterns are harder to teach than to learn. Antonio et al. [10] and O' Cinneide et al. [11] state that lecturing in the traditional manner is insufficient for teaching design patterns. A problem-based approach in which students experiment with design patterns is more appropriate [11]. Students need to see patterns in use to grasp the different concepts [12]. Three main categories of teaching methodologies are relating design patterns to introductory programming patterns, teaching design patterns in-the-small and in-the-large, and teaching design patterns in a particular context. These are discussed below.

In progressing from a novice programmer to an experienced one, a programmer develops certain patterns which are evoked in similar situations. In order for students to understand the rationale behind design patterns, design patterns can be introduced by reminding students of patterns that they developed when novice programmers. Proulx [13] presents such patterns for an introductory programming course. For example, a design pattern for inputting data from a user would be *prompt the user – read the data- verify-repeat*. Similarly, a pattern for repetition would be *initial setup-verifying loop condition-loop body-loop condition update-post mortem*. On a higher level patterns are developed for conversions in general, e.g. from Farenheit to Celsius, converting centimeters to metres. As an exercise students can be required to come up with such patterns of their own, e.g. a pattern for recursion.

According to Dewan [14] design patterns must be presented in-the-small and in-the-large. A common error made is that design patterns are only taught in-the-small and not in-the-large. Patterns must firstly be presented in-the-small. Each design pattern should be presented by describing the design pattern, the motivation for each design pattern, examples of use and implementation and a set of smaller exercises for each pattern ([1], [6], [13], [15]). The benefit of using design patterns can be emphasized by requiring students to extend a program that has not used design patterns ([10] and [15]). The students will experience how difficult it is to change a program that has not been designed using design patterns. After illustrating the difficulty of adapting the program, the students will be required to incorporate the use of design patterns and then make the necessary changes. Students should then be presented with larger projects to work on. Initially students must be told which design patterns to implement [7]. This must be followed by a project requiring the students to decide on which patterns to use when designing the system and implement the corresponding design. It is also important that the application domain used is one that the students understand well so as to remove any complication when incorporating the use of design patterns [7].

According to Christensen [1] design patterns should not be taught in isolation but collectively. If design patterns are taught individually students see them as independent patterns used to solve independent problems. Usually more than one design pattern is needed for a particular domain. Students often miss the point that more than one pattern maybe needed to solve a problem.

Weiss [4] and Hamer [5] emphasize the importance of studying design patterns in the context of a project. The choice of the larger project that the students must work on throughout a design patterns course is important. The following section describes different projects that have been successfully used for this purpose.

4. DESIGN PATTERN PROJECTS

The project must be motivating and the application domain must either be one that students are familiar with or be easily understood ([7], [12] and [15]). This section describes some of the projects that have been used in design pattern courses.

Christensen [1] presents the JHotDraw environment for students to experiment with the observer and adapter patterns. This is essentially an environment for developing 2-D editors. This domain also illustrates how to combine patterns. The project used by Pecinovsky et al. [3] involves the drawing of geometric shapes. The patterns used in the project include the servant, crate, observer, bridge, strategy, state and proxy patterns. Weiss [4] requires students to initially implement a simple payroll system. Students have to then extend this project to include more employees and additional taxes. Design patterns used include the factory, singleton and observer design patterns. The game of life has also been used as a design pattern project [7]. Antonio et al. [10] use the set of "abstract strategy games", namely, *n-in-the-row* problems, including connect-four, Complica, and Gravity. The design patterns required include the model-view-controller, observer, strategy, template, factory and abstract factory patterns. Hamer ([5] and [12]) uses a musical composition project to teach design patterns. The assignment requires the use of context-free grammars for musical composition. Design patterns that need to be used are the composite, decorator, factory, and visitor patterns. Dewan [14] requires students to incrementally create a spreadsheet program. This involves programming scanners and evaluators. Design patterns used in this project include factory, façade, iterator and the composite design patterns. Stuurman et al. [15] use the Jabberpoint program to teach design patterns. This program is used to present and create slides (like the Microsoft PowerPoint program). Schriener et al. [16] look at the efficiency of using different design patterns for a Sudoku program. The use of the model-view-controller, observer and the iterator patterns are analyzed for this domain. Nguyen et al. [17] require students to use design patterns in creating a recursive descent parser. Nguyen et al. [18] use a domain that students are familiar with, namely, sorting. This required the use of the template and strategy design patterns.

The following section discusses how design patterns can be of benefit in other Computer Science courses.

5. USING DESIGN PATTERNS IN OTHER CS COURSES

Paterson et al. [19] suggest that design patterns are used to teach the Java API. The Java API implements a number of design patterns. Patterns can therefore be used to teach Java. If students understand patterns they will be able to understand the design of the Java API.

According to Lang et al. [20] design patterns can help students understand object-oriented design principles. Thus, design patterns can be used in the instruction of object-oriented design and programming.

Nguyen [21] use design patterns in a data structures course. Design patterns are used in algorithms for lists, trees, queues, traversing container structures and conversions between data structures. Design patterns used include the state, singleton, strategy, iterator and adapter patterns.

Wick et al. [22] use the template and strategy design patterns in a course on genetic algorithms.

6. A WAY FORWARD

From the previous sections it is evident that students experience difficulty in understanding and applying design patterns. Teaching design patterns is also not an easy task. In summary, in order to help students relate to design patterns they need to be reminded of the patterns that they have developed as novice programmers, e.g. repetition patterns. Each design pattern must be presented describing the pattern, purpose of the pattern, examples of uses and implementation. Students must then be given exercises to work on for the individual patterns. It must be made clear that design patterns are not used in isolation and usually more than one design pattern is needed to solve a problem. Exercises should also include existing code which does not use design patterns, with new requirements. This is used to illustrate how difficult it is to modify such code. The code must then be extended to incorporate the use of design patterns and make the necessary changes. Exercises requiring the students to implement specified design patterns for particular applications are also needed. Assignments requiring students to choose which design patterns to apply for specific applications are essential. This must be done on both a small and large scale. The application domains used for exercises and projects must be areas that students are familiar with or can easily understand. Design patterns can also play an important role in the teaching of other CS topics such as object-oriented programming and data structures.

From the above summary it is apparent that learning design patterns requires more than one level of cognitive reasoning. These levels of reasoning are usually developed at different stages of the CS undergraduate curriculum. Thus, a single course on design patterns may not be sufficient for students to grasp the necessary concepts. This section proposes teaching design patterns throughout the CS undergraduate curriculum. To illustrate how this can be achieved the author uses the current CS undergraduate curriculum at the above university. Table 1 lists the core courses that students wishing to major in Computer Science must complete.

Table 1. CS Undergraduate Curriculum

Year	Semester 1	Semester 2
First	Introduction to Computer Science	Computer Programming
Second	Object-Oriented Programming	Data Structures
Third	Operating Systems Computability&Automata Theory	Software Design Computer Networks

The first year modules concentrate on developing students' problem solving skills and teach procedural programming in Java. Topics covered include the standard programming structures, such as, input, output, variable declarations and types, repetition, modularization, recursion and arrays. Both these first year modules can be used to get students used to thinking and problem solving in terms of patterns. Patterns presented by Proulx [13] can be used for this purpose. Students can also be required to come up with their own patterns for problem solving.

The latter part of the second semester module in first year introduces object-oriented design and programming. The first semester course in second year continues with these concepts on a larger scale. Design patterns can be introduced in this course. Once students are familiar with the OOP principles and implementation of them, the benefit of design patterns can be illustrated. Each design pattern can be introduced and students must complete exercises on the implementation of each pattern. Previous program solutions created by students can be extended and improved to include design patterns and meet additional requirements. All assignments at this level should specify what design patterns to use.

The data structures course in the second semester of second year can be used to further develop students' understanding of design patterns. The use of design patterns to create more flexible data structure algorithms can be illustrated. As in the first semester course, at second year level students should not be required to decide which design patterns to apply, but should be shown which ones are suitable and be required to implement them.

At third year level students should acquire the skills requiring the highest level of cognitive reasoning, namely, determining which design patterns to apply. At this stage students should have developed both a knowledge and understanding of the different design patterns, and see the benefit of using design patterns. Initially, students should be given smaller problems requiring them to choose appropriate design patterns during design and implement the designs. This should be followed by large scale projects. One of the tasks of such a project would be to identify which design patterns to use and substantiation of each choice. In addition to applying design patterns in a course on Software Design, students can also be required to include the use of design patterns in the design and implementation of solutions to problems in other third year level CS courses such as Operating Systems, Computer Networks or Artificial Intelligence.

7. CONCLUSION

Design patterns are difficult to learn, requiring different cognitive levels of reasoning. The paper highlights the learning difficulties generally encountered by students. The standard approach to lecturing has proven to be insufficient and a problem-based approach is necessary. Students need to experience the use of design patterns both in-the-small and in-the-large. The paper also presents suggested teaching methods that can be used to improve the learning and teaching of design patterns. The knowledge and skill required to successfully use design patterns is best developed incrementally. Thus, the paper proposes teaching design patterns throughout the CS undergraduate curriculum instead of presenting them in a single course at third year level. The paper has illustrated how this can be attained.

8. REFERENCES

[1] Christensen, H. B. 2004. Frameworks: Putting Design Patterns into Perspective. *SIGCSE Bulletin inroads*, 36, 3 (September 2004), 142-145.

[2] Nguyen, D., and Wong, S. B. 2000. Design Patterns for Lazy Evaluation. *SIGCSE Bulletin inroads*, 32,1(March 2000), 21-25.

[3] Pecinovsky, R., Pavlickova, L. and Pavlicek, L. 2006. Let's Modify the Objects-First Approach into Design-Patterns-First. *SIGCSE Bulletin inroads*, 28, 3 (September 2006), 188-192.

[4] Weiss, S. 2005. Teaching Design Patterns by Stealth. *SIGCSE Bulletin inroads*, 37, 1 (March 2005), 492-494.

[5] Hamer, J. 2002. A Musical Approach to Teaching Design Patterns. *SIGCSE Bulletin inroad*, 34, 3 (June 2002), 197.

[6] Clancy, M.J., and Linn, M. C. 1999. Patterns and Pedagogy. *SIGCSE Bulletin inroads*, 31, 1(March 1999), 37-42.

[7] Wick, M. R. 2005. Teaching Design Patterns in CS1: A Closed Laboratory Sequence Based on the Game of Life. *SIGCSE Bulletin inroads*, 37, 1(March 2005), 487-491.

[8] Freeman, E., and Freeman E. 2004. *Head First Design Patterns*. O'Reilly.

[9] Preiss, B. R. 1999. Design Patterns for the Data Structures and Algorithms Course. *SIGCSE Bulletin inroads*. 31, 1 (March 1999), 95-99.

[10] Antonio, M., Jimenez-Diaz, G., and Arroyo, J. 2009. Teaching Design Patterns Using a Family of Games. *SIGCSE Bulletin inroads*, 41, 3(September 2009), 268-272.

[11] O' Cinneide, M., and Tynam, T. 2004. A Problem-Based Approach to Teaching Design Patterns. *Inroads SIGCSE Bulletin*, 36, 4(December 2004), 80-82.

[12] Hamer, J. 2004. An Approach to Teaching Design Patterns Using Musical Composition. *SIGCSE Bulletin inroads*, 36, 3(September 2004), 156-160.

[13] Proulx, V. K. 2000. Programming Patterns and Design Patterns in the Introductory Computer Science Course. *SIGCSE Bulletin inroads*, 32, 1(March 2000), 80-84.

[14] Dewan, P. 2005. Teaching Inter-Object Design Patterns to Freshman. *SIGCSE Bulletin inroads*, 37, 1 (March 2005), 482-486.

[15] Stuurman, S., and Florjin, F. 2004. Experiences with Teaching Design Patterns. *SIGCSE Bulletin inroads*, 36, 3 (September 2004), 151-155.

[16] Schreiner, A. T., and Heliotis, J.E. 2008. Sudoku – A Little Lesson in OOP. *Inroads SIGCSE Bulletin*, 40, 2 (June 2008), 44-47.

[17] Nguyen, D. Z., Ricken, M., and Wong, S. 2005. Design Patterns Parsing. *SIGCSE Bulletin inroads.* 37, 1 (March 2005), 477-481.

[18] Nguyen, D., and Wong, S. B. 2001. Design Patterns for Sorting. *SIGCSE Bulletin inroads*, 33,1 (March 2001), 263-267.

[19] Paterson, J. H., and Haddow, J. 2004. A Proposed Design Patterns Extension for the BlueJ IDE. *SIGCSE Bulletin inroads.* 36, 3(September 2004), 278.

[20] Lang, J. E., Bogovich, M. R., Barry, S. C., Durkin, B. G., Katchmar, M. R., Kelly, J. H., McCollum, J. M., and Potts, J. M. 2001. Object-Oriented Programming and Design Patterns. *SIGCSE Bulletin inroads*, 33, 3(December 2001), 68-70.

[21] Nguyen, D. 1998. Design Patterns for Data Structures. *SIGCSE Bulletin inroads*, 30, 1(March 1998) 336-340.

[22] Wick, M. R., and Phillips, A.T. 2002. Comparing the Template Method and Strategy Design Patterns in a Genetic Algorithm. *Inroads-SIGCSE Bulletin*, 34, 4 (December 2002), 76-80, ACM.