*Int. J. Engng Ed.* Vol. 24, No. 4, pp. 717–728, 2008
Printed in Great Britain.

0949-149X/91 $3.00+0.00


# Teaching GoF Design Patterns through Refactoring and Role-Play*

GUILLERMO JIMÉNEZ-DÍAZ, MERCEDES GÓMEZ-ALBARRÁN and
PEDRO A. GONZÁLEZ-CALERO
*Dept. de Ingeniería del Software e Inteligencia Artificial, Universidad Complutense de Madrid, C/ Prof. José García Santesmases s/n. 28040. Madrid, Spain. E-mail: gjimenez@fdi.ucm.es*

*In order to fully understand the implications of object-oriented design patterns, students need to consider alternative designs to a problem and to analyse these solutions in terms of coupling, cohesion and extensibility. Lecture-based approaches to teaching design patterns do not provide students with the insights needed unless they already have experience in object-oriented design. In this paper we present an approach to teaching design patterns that promotes active learning and makes students participate in refactorings through role-play sessions. We describe two experiments that demonstrate student acceptance and present promising results on the effectiveness of the approach.*

**Keywords:** pattern-directed refactoring; active learning; role-play; object-oriented design pattern learning

## INTRODUCTION

PROFESSIONAL SOFTWARE DEVELOPERS address recurring design challenges every day. To solve them, they resort to their own experience, applying some variation of a solution that has worked well in a similar situation in the past. Some records of this professional knowledge about software design have been abstracted and named as design patterns. In industry, the use of design patterns has a great impact on the way software is developed. Through the years, they have been widely used and applied to very different kinds of software applications [1–4]. Design patterns are created from these experiences and they capture the essential parts of a software design. They also provide a common design vocabulary for sharing design information among developers [5].

Although there are many different types of software patterns (procedural patterns, distributed processing patterns, anti-patterns . . .) our interest centres on the object-oriented design patterns catalogued by Gamma, Helm, Johnson and Vlissides, called the Gang of Four (GoF) design patterns [6], which are an indispensable body of knowledge for any Computer Science or Software Engineering student. An object-oriented design pattern comprises: the pattern's name; the problem, which defines the context where the pattern can be applied; the abstract solution that the pattern defines; and the consequences and drawbacks of adopting the solution defined by the pattern. The solution provided by an object-oriented design pattern identifies the classes that participate in the solution, their roles, responsibilities and collaborations and how they interact to distribute responsibility. The abstract solution is often accompanied by an example of the pattern implementation in a concrete programming language.

Design patterns are also a learning aid for novice developers, who learn skills and design concepts that are independent from current technology and use good practices described in the shared vocabulary of a professional development team. In recent years, design patterns have become so central to Computer Science and Software Engineering curricula [7, 8] that several works have advocated their use in introductory courses in object-oriented programming [9–13]. Lecturing is one traditional teaching style followed in order to explain design patterns to the students. Our experience teaching design patterns has revealed that the lecture-based teaching approach, by itself or combined with the use of isolated examples (such as the ones in [6]) and study groups that read about patterns and discuss their advantages and disadvantages, is quite effective for graduate students. These students have enough experience in object-oriented design to appreciate the ideas behind the patterns and to relate them to problems previously faced.

However, the teaching approaches previously mentioned are quite ineffective for an undergraduate audience. These students lack the experience needed to understand the subtleties and implications of the design ideas given by the patterns. Undergraduate students tend to measure the correctness of a program by evaluating its behaviour rather than the qualitative properties of its design. They do not pay attention to some impor-

* Accepted 25 April 2008.

tant aspects that determine the quality of a good design, such as scalability, robustness to changes or its degree of flexibility and extensibility [14].

Moreover, they do not feel confident with sketchy real world examples such as those described in [6]. For example, they do not completely understand either the TCP connection protocol employed to motivate the State pattern, or the line-breaking algorithms that motivate the Strategy pattern. Undergraduate students simply assume the design pattern as an obvious solution, without reflecting on it and without analysing the advantages it provides with regard to other alternatives.

The problems encountered when teaching object-oriented design patterns to undergraduate students can be summarized in the following two main ideas: a simple example about how to implement a pattern is not enough to ease the comprehension of the pattern, and undergraduate students do not have enough design experience to appreciate the patterns. As stated in [15], the real problem when learning design patterns is not only how to implement them, but to understand the problem, to decide which design pattern solves it and to know which are the implications of applying this pattern, in order to evaluate if another pattern fits better.

Despite the quantity of work in teaching design patterns, this area still presents interesting pedagogical challenges to instructors. Evidence of that fact is the workshop series 'The Killer Examples for Design Patterns and Object First', which have been held annually at the ACM SIGPLAN International Conferences on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA) since 2002 [14]. Most research works on teaching design patterns consider that they are taught more effectively when they are presented in real world case studies. A case study is a complete software application that includes a narrative explanation of its development process and questions to engage students to analyse, judge and evaluate different issues from its design [16]. A wide diversity of case studies appear in design pattern literature: the elevators of a building [17], a maze [18], a musical composition project [19], a computer game [20] or a presenter [14]. Other works have even proposed the use of professional object-oriented frameworks [21] as complex generic software systems whose architecture can be explained in terms of design patterns. We agree with the use of real world case studies. However, although a framework is generally an example of good design ideas, we do not agree with the use of object-oriented frameworks as initial case studies when teaching novice students because of their high complexity. For instance, even a framework developed with pedagogical goals in mind such as JHotDraw [22] consists of more than 180 classes, where patterns are combined and a class intervenes in different instantiations, playing different roles in the same pattern, or in different patterns. We postpone the use of frameworks until the students have a basic knowledge of the patterns.

Apart from this, students learning design patterns should feel the pain of a design which has some pitfalls in order to appreciate the convenience of applying a concrete design pattern [23]. We agree with [19] that design patterns are not 'solutions in search of a problem' like data structures. Instead, design patterns are a technique to generate solutions in software design, so students should experience their use in practice. For this reason, we agree with those approaches that promote a more active participation of the student in the design of the case study, such as the teachlets described in [24].

This article describes an innovative approach to teaching design patterns to undergraduate students which has been successfully applied throughout several academic years at the Complutense University of Madrid (Spain). Our learning-by-doing approach helps students not only to gain insight into the how and the why of design patterns but, more importantly, it helps students learn to apply them. Following the axiom 'Good design comes from experience, and experience comes from bad design' [10], our students learn to use GoF design patterns by collaboratively identifying pitfalls in existing designs, analysing the potential alternatives and their advantages and drawbacks, and achieving a better solution by means of a pattern-directed refactoring of the initial design. In order to promote active learning [25, 26] and student participation, the analysis and comprehension of alternative designs are developed through role-play sessions. Several authors have emphasized the importance of identifying the roles that intervene in a design pattern and their relationships and collaborations in order to distribute responsibilities [9, 15]. Therefore, in our approach each student performs the role of an object within the case study in order to understand better the collaborations among the objects by simulating the execution of a use case.

The next section describes the pedagogical tools used to teach design patterns when following the pedagogical approach described in the section that follows it. The section Experience reports illustrates the application of this teaching approach during the last two years in our institution. This section also contains the evaluation of the experience from two different points of view: the subjective evaluation from the students, and the effectiveness evaluation according to the student grades. This article concludes with a proposal of future extensions of the approach according to the conclusions extracted from the experience evaluation.

## TAKING A GLANCE AT THE PEDAGOGICAL TOOLS USED

According to the principles and challenges of our approach, in our classes we make use of two pedagogical tools: pattern-directed refactoring episodes and role-play sessions.

*Refactoring episodes*

Refactoring is a technique widely used in software development communities for cleaning up and improving software structure in an efficient and controlled manner. A refactoring is a change in the structure of software to make it easier to understand and cheaper to modify without changing its observable behaviour [27].

Instead of applying primitive refactorings like 'Rename method' or 'Move method' [27], we propose that the students follow a pattern-directed refactoring strategy to yield a better software design. Some examples of pattern-directed refactorings appear in [28]. Some research works describe the application of primitive refactoring methods in introductory computer science courses [29, 30] but we do not know about the application of pattern refactorings with pedagogical purposes.

Presenting design patterns as a result of a refactoring process is not only useful from a pedagogical point of view, taking the student from simple obvious solutions to more elaborated ones, but is also in line with modern methodologies for software engineering such as Extreme Programming [31] and, in general, agile methodologies [32].

Instead of careful up-front design that intends to foresee and facilitate future changes in a system, agile methodologies propose incremental design where sophisticated design patterns emerge in the process of continuous refactoring as new functionality is implemented in the system.

When we give the students the 'design pattern'-hammer, we run the risk that every software application looks like a nail. Instead of using patterns in a design from scratch, we use a pattern-directed refactoring strategy, which provides the opportunity to apply design patterns in order to enhance existing designs. This strategy, when controlled by the instructor, alleviates the students' impulse to overuse design patterns and lets the students realize that including unnecessary sophistications in a design is a waste of time.

*Role-play sessions*

In order to notice the changes and improvements better as a result of refactorings, as well as the internals of the initial naive designs employed in our classes, our students work collaboratively in role-play sessions.

Role play [33] is a kind of active learning where participants learn complex concepts—hard to understand by means of abstract explanations—while they simulate a scenario. In this scenario each participant plays a predefined role. When applied to object-oriented design, each actor in the role-play session plays the role of an object. The participants act out (part of) the software application in a predefined list of execution scenarios. During the role-play, participants interact with each other, learning from themselves, the other participants and the roles played [34].

In our case, the instructor properly selects the set of scenarios and CRC cards gather the information about the classes involved.

CRC cards [35] are a technique widely used in responsibility-driven design. A CRC card represents a Class and it contains information about class Responsibilities and Collaborators. A responsibility is what a class knows and what it can do, i.e. the class' properties and methods. A collaborator is a class that helps carry out a responsibility.

CRC cards provide valuable support for discussing and evaluating an object-oriented design in a collaborative way. After creating the responsibilities of each class solely, designers can make use of the corresponding CRC cards when simulating how classes interact to achieve a certain functional requirement. This simulation forces participants to evaluate a design solution and to discover alternatives to enhance it.

To conclude the presentation of the tools supporting our role-play sessions, we refer to role-play diagrams (RPDs), which the instructor uses to track the role-play sessions. An RPD is a semi-formal representation of a scenario execution in an object-oriented application that captures objects' states [34]. The ability to track each object state is essential in our approach because we need to represent the initial values of the attributes and how the values change throughout the role-play simulation.

Instead of using CRC cards and RPDs, we could employ the piece of source code that represents the scenario execution performed in a role-play session. We believe that students get distracted by implementation details when employing source code, while CRC cards and RPDs gather the essential information to follow a role-play simulation.

## A HIGH-LEVEL DESCRIPTION OF THE TEACHING APPROACH

In recent years we have taught courses on object-oriented design patterns for undergraduate Computer Science students. The high-level learning objectives in these courses are:

- understanding the notion of design pattern and learning a subset of GoF design patterns [6];
- learning to apply patterns to solve real problems;
- understanding the consequences of applying a certain design pattern.

The high-level course schema runs as follows:

- We first introduce the notion of object-oriented design patterns, and background concepts such as software reuse, coupling and cohesion.
- One or more iterations of the following process is made:
  - Providing the students with some background into some GoF design patterns following a lecture-based approach.

- Making the students actively participate in the task of iteratively improving the naive design of a case study by means of applying appropriate design patterns. This way students:
  - learn to identify evidences of bad designs. These evidences are called *code smells* (a hint that something has gone wrong somewhere in your code) [27];
  - face a real design problem and appreciate the potential of design patterns in real contexts;
  - become familiarized with the application of design patterns, facilitating future instantiations of them in different contexts; and
  - discover the benefits and consequences of applying a specific design pattern.

If few hours are available for the course, say up to 20 hours, just one iteration should be done where a selection of patterns is first described and then put into practice. In this case, the course should cover the most representative patterns of each category (creational, behavioural, and structural). As the number of hours available increases, more iterations can be done by grouping related patterns and covering their theoretical implications in more detail and putting them into play through role-play sessions where different patterns are tried in order to solve the same problem.

Regarding the methodology for the practical collaborative design sessions, once the initial naive design of the case study is presented at a high level in the course, the instructor leads each iteration in this pattern-directed refactoring strategy, following the next steps:

1. Select a functional requirement. The instructor selects a functional requirement to add to the case study or an existing requirement to improve.
2. Analyse the attempted solution ('Before' design). The instructor specifies several use cases (also known as execution scenarios) and students simulate them using the CRC cards corresponding to the classes initially provided. These role-play sessions help the students to become familiarized with the design. The instructor depicts each role-play simulation using RPDs. These diagrams keep the students from getting lost while they are performing a role or watching the simulation. After the simulations, although the classes seemed to be appropriate to the proposed case study so far, students realize that the new requirements force them to revise the design in order to improve it.
3. Propose a new solution ('After' design). The students are responsible for identifying and applying a design pattern that can improve the previous design. This refactoring consists of selecting a design pattern and instantiating it in the case study. In this context, instantiating a design pattern means relating the classes and methods within the 'Before' design to the roles defined by the design pattern. Students can also add new interfaces and classes or modify the existing ones, if needed.
4. Analyse the 'After' design. Starting from the new set of classes that result from applying the design pattern, students again simulate the execution scenarios. During these role-play sessions, students can also detect additional modifications to the general pattern derived from the case study. These simulations are employed to confirm that the application of the design pattern preserves the behaviour of the case study and resolves the pitfalls encountered in the 'Before' design. Once more RPDs serve to track the simulations.
5. Reflect on the pattern. Finally, the students compare the 'After' design and the 'Before' design in order to realize the benefits of applying the design pattern selected. The instructor also stresses the consequences, benefits and drawbacks of the pattern applied. If students choose several patterns during the proposals stage, the instructor and the students discuss their application in the case study instead of the one selected.

*A sample session: introducing the* Prototype *pattern*

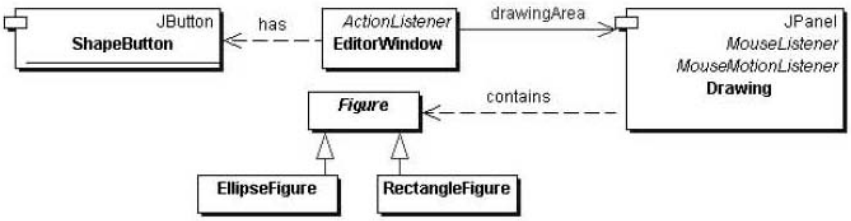
In this section we describe a session that follows the teaching approach described above and lets students learn to use the *Prototype* pattern, a creational pattern that promotes the use of a prototypical instance to create a kind of product objects [6]. This pattern hides the different kinds of products that the client who uses this object knows. Furthermore, the type of products can be changed at run-time simply by changing the prototypical instance employed to create the products.

To perform the refactoring and role-play sessions, we have selected a simple drawing editor as the discussion case study. The resulting drawing editor will contain a set of creation tools to generate predefined figures (rectangles, ellipses, polygons . . .) and a selection tool to select, move and resize them. We supply the students with an initial version of the drawing editor classes. The editor consists of the following sketchy classes (Fig. 1(a)):

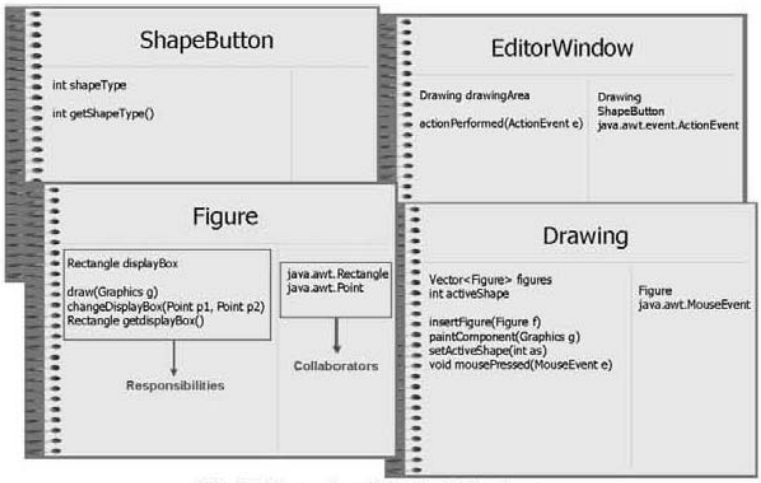
- *EditorWindow*. It coordinates the application.
- *ShapeButton*. It serves to decide the type of figure that the user creates.
- *Drawing*. It contains the figures created.
- *Figure* and its subclasses, *FEllipse* and *FRectangle*. They represent the type of figures that the user can create in the editor.

The scenario description is the following:

> The user wants to create a new figure in the drawing editor. Figure creation is made in two steps. First the user selects the type of figure by clicking one of the buttons available in the toolbar (Rectangle button and Ellipse button). In this scenario the user clicked




(a) Initial classes for the drawing editor.




(b) CRC cards of the initial classes.

Fig. 1. 'Before' design: The documentation provided to perform the initial role-play session.

on the Ellipse Figure button. Then, the user clicks on the canvas and the application creates an ellipse, whose display box has the upper left corner of the mouse cursor's position. The first mouse click is received by the *EditorWindow* class through the message *ActionPerformed* while the second click is sent to the *MousePressed* responsibility of the *Drawing* class.

Figure 2 shows a simple sample of the scenario.

The content of the classes is reflected onto CRC cards, as we can see in Fig. 1(b). Using these classes, the students should simulate the following scenarios:

- What happens when the user pushes the button to select the type of figure.

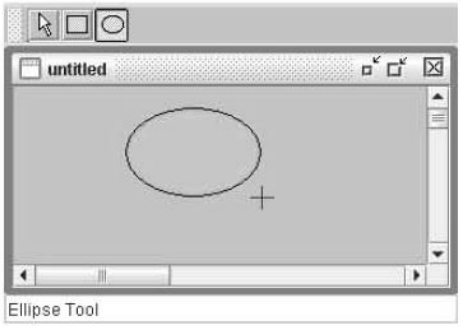



Fig. 2. Intended behaviour for the scenario.

- How to create the figure after the user has selected the Rectangle figure button.
- How to create the figure after the user has selected the Ellipse figure button.

The main goal of the first scenario is to understand that each tool button contains the type of figure that will be created later and to see how the type is propagated from the *ShapeButton* class to the *Drawing* class. The second and the third scenario are more important because the students will understand how to use the type stored in the *Drawing* class to create the suitable figure.

For example, we start the second scenario with the *RECTANGLE_TYPE* value stored in the instance of the *Drawing* class. When instructor invokes the *MousePressed* method with a parameter called *oEvent* (which contains coordinates where the user has clicked on the canvas), the participant responsible for performing the role of a *Drawing* object explores its responsibilities in order to create the figure. First, he or she looks up its *activeShape* responsibility. If it contains the *RECTANGLE_TYPE*, they will create an instance of the *RectangleFigure* class. Then they will request the *oEvent* parameter for the point where the user has clicked. Once the *Drawing* object has this information, they delegate on the *RectangleFigure* participant to change its display box using the responsibility *changeDisplayBox*. Once the *Figure* has updated the display box and

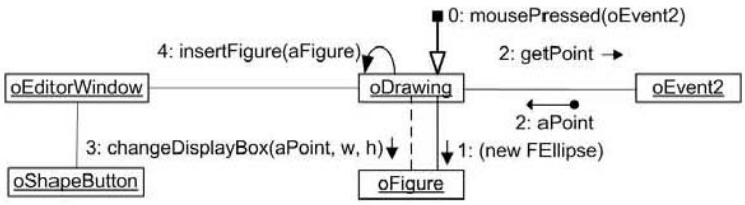



Fig. 3. RPD created during the role-play session after the last use case.

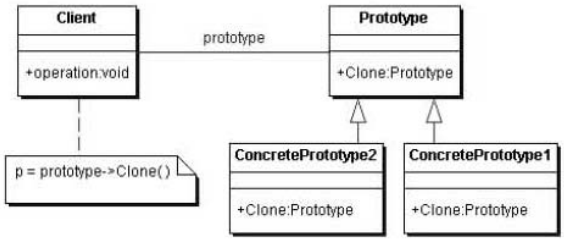



Fig. 4. Structure and roles of the *Prototype* pattern.

the control flow returns to *Drawing*, they add the new figure to itself using the responsibility *insertFigure*. After that, the scenario concludes. During the simulation, the instructor completes an RPD (Fig. 3) that records the student interactions.

The same scenario is repeated with the *ELLIPSE_TYPE* stored in the *Drawing* object. In this scenario, students realize that it is necessary to employ a conditional expression to decide whether to create a *RectangleFigure* or an *EllipseFigure* object.

The simulation of the last two scenarios reveals that the initial version of the drawing editor classes forces the inclusion of conditional expressions in the *MousePressed* responsibility in the Drawing class in order to decide the type of an object. These conditional statements are wrongly replacing the use of polymorphism. Moreover, these conditional structures usually spread over the whole source code, complicating software scalability and maintenance. Now it is time to refactor the 'Before' design by using a design pattern.

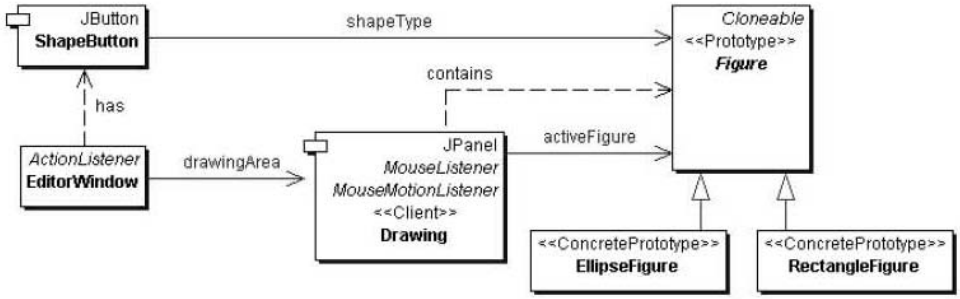
At this moment, students enter a discussion about which design pattern should be applied. According to the light background into design patterns provided in a previous lecture-based session, the main candidates are the *Prototype* and the *Factory Method* patterns. The instructor decides to apply the first one and leave the discussion of the second one until after completing the current session. Now, the students should identify the roles of the selected pattern (see Fig. 4) in the editor design. *Drawing* performs the Client-role, while *Figure* will perform the Prototype-role. Finally, the subclasses of Figure will perform the ConcretePrototype-roles. Next, the instructor provides the new set of CRC cards to continue with the learning session. The new classes provided and their CRC cards are presented in Fig. 5.

The students again perform the scenarios described above. During the simulation, the instructor will point out that the use of the *Prototype* pattern allows the removal of the conditional statements from the *Drawing's MousePressed* responsibility because, both the *ShapeButton* and the *Drawing* class will store the prototypical instance that will be copied to create new figures. Furthermore, adding a new type of figure is quite simple: create a new subclass of *Figure*. So, the previously mentioned *code smell* has disappeared.

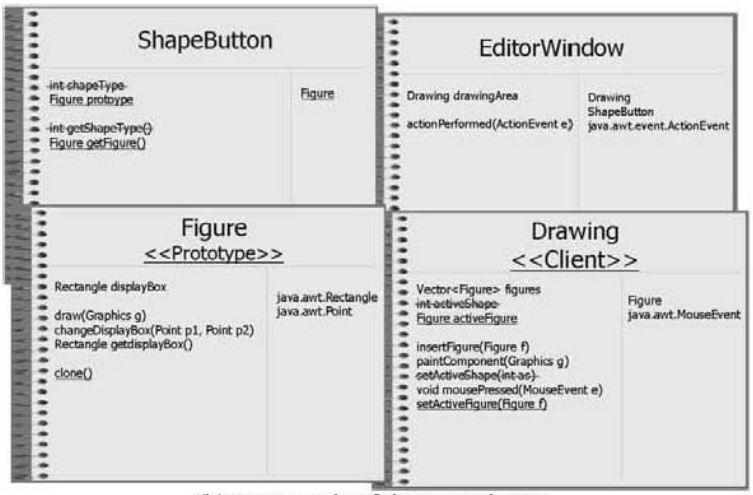
After completing the scenarios, the instructor explains the decision to apply the *Prototype* pattern instead of the *Factory Method*. Although both patterns will resolve the problem, the first one is easier to apply for this case study because of the need to create a hierarchy of *Creator* and *ConcreteCreator* classes to instantiate the *Factory Method* pattern. The instructor concludes the session by stressing some drawbacks of this pattern, such as the fact that its application needs to implement the *Clone* operation correctly. This operation becomes tricky with classes that contain complex structures.

## EXPERIENCE REPORTS

We have applied the teaching approach described in two academic years: 2005–2006 and 2006–2007. To date, 61 undergraduate Computer Science students have participated in two editions of a 25-hour seminar on Design Patterns. Owing to the large number of students in each edition and the low number of objects that intervene in the scenarios only a few students actively participated in each scenario execution while the others observed the performance. However, throughout the seminars, all students were engaged in at least one scenario execution. Each role-play session lasts around one hour, except the first one. Students do not feel confident with these active techniques so the first session usually takes approximately one hour and a half. During the role-play sessions, every student has a set with all the CRC cards employed in the role-play, except the actors, who only have the CRC card of the object that they perform. All the material employed in the sessions is available at http://gaia.fdi.ucm.es/projects/virplay/ (in Spanish). These experiences have given us invaluable insights as to how to




(a) New classes for the drawing editor.




(b) CRC cards of the new classes.

Fig. 5. 'After' design. The editor design after instantiating the Prototype pattern on the 'Before' design classes.

best organize the contents and involve the students in the teaching process.

All the students who participated in our design pattern courses have completed 50% of their studies in Computer Science. Students should complete a Pretest on object-oriented programming in Java and design patterns in order to evaluate their previous knowledge. The results indicate that they all have an upper-intermediate level of knowledge of object-oriented programming using the Java programming language –a mean score of 6.4 out of 10– and little or no knowledge of design patterns –a mean score of 3.4 out of 10.

Experiences are grouped in the following two experiments.

*First experiment: setting the approach in motion*

This experiment was exploratory and occurred during summer 2006 in a Design Patterns seminar. 24 students attended the seminar.

The course was organized as follows:

- During the first 20% of the course, we provided the students with some background regarding design patterns (following the explanations and small isolated examples included in [6]), their benefits and drawbacks, isolated from a software application.
- The following 70% of the course time involved the students in a pattern-directed refactoring approach combined with role-play. Before the refactorings, we introduced students to the pedagogical techniques and tools used –CRC cards, role-play activities and RPDs– and we sketched the requirements and the initial design of the drawing editor that the students are going to redesign.
- In the remaining 10% of the course we presented the students with a real-world software application whose design relies heavily on design patterns. This application is JHotDraw [22], a framework to create graphics editors, which can be considered an extended and more professional version of the drawing editor developed by the students.

Goals

The main goal of this experiment was to gain an insight into how well or poorly received the learning approach is by the students. We aimed to evaluate the students' reaction to the role-play sessions and to measure the students' opinions about using CRC cards and RPDs instead of

source code to understand how an application works internally. The results of this experiment allowed us to consider more complex experiments where role-play sessions could be used in parallel with traditional approaches, knowing that the role-play approach would be acceptable to the students.

Experimental results

Students completed several questionnaires evaluating the learning approach. We can stress that the students found the experience very motivating. They highlighted the relevance of understanding design patterns as part of object-oriented design learning. They especially appreciated the experience of putting them into practice in the design of a drawing editor. The overall answers about the quality and usefulness of the Design Patterns seminar were positive and the average score was 4 points out of 5. Students considered the use of the drawing editor as a more useful discussion context than the isolated examples presented in the first part of the seminar (4.6/5 in contrast with a 3.3/5). Students positively assessed the techniques used: CRC Cards, 4.4/5; Role-play, 4/5; RPDs 4.2/5. Finally, students felt confident using CRC Cards and RPDs to understand design patterns and they did not consider it necessary to replace them with the source code of the application.

*Second experiment: settling the approach and analysing its potential*

The second experiment occurred during autumn 2006. Thirty-seven students attended the second edition of the Design Pattern seminar. We changed the course organization slightly in order to evaluate the pedagogical efficiency of our approach. After the lecture-based background about design patterns, students were divided into two groups. The control group (CG) completed two design iterations through an approach where the instructor did not promote the collaboration of the students: the instructor presented the pattern-directed refactorings in a lecture-based manner. The experimental group (EG) completed these two design iterations by combining pattern-directed refactoring and role-playing. Later, we regrouped the students and they completed the remaining iterations through refactoring and role-playing.

Goals

We developed this experiment with the aim of estimating the pedagogical efficiency of our approach. In order to measure it, students completed two different tests: Test 1 was completed after the first two design iterations; Test 2 was completed at the end of the experiment. We have employed the grades of these tests:

- to compare the knowledge acquired by the students in CG with the students in EG;
- to measure if the students in CG enhanced their knowledge when they participated in role-play sessions;
- to evaluate if students in EG understood the application of design patterns better than the students in CG.

Moreover, the students completed several questionnaires to evaluate the learning approach. These questionnaires let us:

- corroborate the student's acceptance of the learning approach obtained in the first experiment;
- know if the students preferred participating in the role-play sessions rather than observing the role-play;
- know the CG students' preferences between a lecture-based refactoring class and a role-play-based one;
- know if the students considered the refactoring task useful in learning design patterns.

Experimental results

As in the first experiment, students evaluated our learning approach by means of several questionnaires. Once more we can stress that the experience was very motivating for the students. As in the previous experiment, students appreciated the experience of putting the design patterns in practice. They considered that the seminar is suitable for obtaining overall knowledge about patterns (4.7 over 5 score), their intention (4.4/5), structure (3.6/5) and consequences (3.8/5). They confirmed the usefulness of the case study instead of isolated examples (4.5/5 in contrast with 3.9/5). Most students who participated in the lecture-based refactoring preferred the role-play-based approach (66%), and the students who participated in the role-play sessions considered participation more useful than the simple observation of a role-play (3.7/5, where 5 is 'Participating in role-play sessions is significantly more useful than observing it'). Finally, students positively evaluate the use of CRC cards (3.9/5), Role-play (3.7/5) and RPDs (4.2/5) and most students did not miss the application source code instead of CRC cards (70%) or RPDs (81%).

To test different aspects of our approach, we have used the SPSS statistical software package [36] to run standard procedures to determine whether the differences between two samples are significant enough to conclude that the samples belong to different populations. We apply the t-test when both samples are normally distributed and the Wilcoxon signed-rank test when they are not. To determine whether a sample is normally distributed we use the Shapiro–Wilk test.

We divided the students into two groups by means of a random process. The analysis of the Pretest results revealed that there were no significant differences between them and, according to those results both groups, EG and CG, could be considered as samples of the same population.

Table 1. Average grades obtained by the students after completing Test1 and Test2

| Tests | | Experimental Group (EG) | | | Control Group (CG) | | | Total | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | **Mean** | ***N*** | **Std. deviation** | **Mean** | ***N*** | **Std. deviation** | **Mean** | ***N*** | **Std. deviation** |
| Test1 | Total | **7.6316** | 19 | 1.46099 | **7.3889** | 18 | 1.19503 | 7.5135 | 37 | 1.32543 |
| | Practical questions | **7.0526** | 19 | 1.80966 | **6.8889** | 18 | 1.84355 | 6.9730 | 37 | 1.80257 |
| Test2 | Total | **7.7895** | 19 | 1.08418 | **8.0000** | 18 | 0.90749 | 7.8919 | 37 | 0.99398 |
| | Practical questions | **7.3026** | 19 | 1.19774 | **7.7083** | 18 | 1.07187 | 7.5000 | 37 | 1.14109 |

Table 2. Average differences between grades obtained in Test1 and Test2

| | Group | *N* | Mean | Std. deviation | Std. error |
|---|---|---|---|---|---|
| Total difference | EG | 19 | **0.1579** | 1.21395 | 0.27850 |
| | CG | 18 | **0.6111** | 1.53925 | 0.36280 |
| Practical question difference | EG | 19 | **0.2500** | 1.33073 | 0.30529 |
| | CG | 18 | **0.8194** | 1.93992 | 0.45724 |

Test1 contained ten questions, five about theoretical issues of design patterns and five about putting into practice and recognizing design patterns. We compared the results of Test1 obtained for each group as well as the results obtained considering only the practical questions. As we can see in Table 1, the EG obtained better results in both Test1, as a whole, and the practical questions. However, according to the Wilcoxon signed-rank test, the differences are not significant.

After regrouping the students and concluding the drawing editor redesign, they completed Test2. This test included eight questions about applying and identifying design patterns. Again, we compared the results between groups and, to our surprise, the CG sharply improved their grades above those in EG, as shown in Table 1. Nevertheless, the Wilcoxon signed-rank test determined that, according to the results of Test2, EG and CG still belong to the same population. We also analysed the differences between the increase from Test1 to Test2 in EG and CG (see Table 2) to conclude that, according to the t-test, the difference is not statistically significant.

Finally, we measured the consequences of participating in a role-play session instead of acting as a mere observer of the simulation. To do that, we again analysed the results by splitting the questions into two different groups for every student: (1) questions about the patterns that the student has actively participated in, and (2) questions about the patterns where the student has only observed the simulation. 100% of the students correctly completed all the questions about patterns where they played a role (a 10.00 average grade), while only 50% of the students correctly completed all the questions about patterns where they did not play any role, resulting in an average grade of 8.85. A t-test lets us finally conclude that there is a remarkable difference between observing a role-play session and participating in it.

*Lessons learned*

We are pleased with the high success that our teaching approach has had in the participants in our courses. Students' evaluations show that they feel motivated by a learning experience that involves them in their learning process. Students' motivation increases when they are involved in the design of an application and they collaboratively reflect on how to improve an acceptable but naive design.

Regarding pedagogical results, grades in Test1 show that our motivating teaching approach is more effective than the lecture-based approach, although the results in Test1 do not seem to have statistical significance. We have also confirmed that participating actively in role-play sessions is more effective than observing them. Furthermore, according to the results obtained when analysing the differences between grades in Test1 and Test2 in students from the CG, we can conclude that these students better assimilate concepts in design patterns after attending and participating in role-play sessions. First passive iterations help the student to understand better the design problem and its context in such a way that next interactive iterations become more valuable. Anyway, we should corroborate this hypothesis with future experiments.

## FUTURE EXTENSIONS

The experiences described have prompted us to reflect on the organization of courses whose main goal is to teach design patterns in depth. Lecture-based sessions should be kept to provide the students with some background on design patterns and include some refactoring episodes developed by the instructor. These sessions should be complemented with pattern-directed refactoring combined with role-play sessions, as described in this

paper. This approach strengthens student knowledge in design patterns. Moreover, it gives students the opportunity to put them into practice as a method of achieving more flexible designs, instead of forcing them to generate isolated instances of patterns in the development of an application.

The approach presented in this article gives the students deep design-level knowledge about design patterns. However, as well as feeling confident in using patterns during design, it is worth it for students to know the implementation details of the patterns. Just like the 'Implementation' section described for each design pattern in [6], the implementation of the application redesigned during the rest of the course would complete student knowledge about design patterns. This task would force the student to implement design patterns in a concrete programming language. Owing to seminar time limitations, our students do not implement the application resulting from applying the patterns in a concrete programming language. In compensation for this, we provide them with the resulting Java code as a final documentation element of the seminar. However, as the number of hours available increases, we consider it suitable that students devote part of the time to implementing design pattern uses in a concrete programming language. We are already planning longer seminars in this sense. Currently, we are working on the development of a software tool for supporting the creation of the CRC cards. The instructor will also employ this tool during the refactoring sessions for RPD creation and the modification of CRC cards.

Finally, we should be aware that in recent years much effort has been focused on complementing conventional structured courses (based on lectures and practical laboratory work) with tools and environments that students can use alone or in a collaborative way. In this sense, we are developing tools to support the teaching approach described herein [37]. The main drawback of collaborative role-play design sessions is that of resources: it is better to participate in the play than to watch it, so, ideally, in the class there should just be the number of students needed for the role-play. The technological alternative would be to let the students play in a network-enabled collaborative environment where the human tutor could be helped or even substituted by a virtual one. This work builds upon ViRPlay-3D [38], a tool that gathers our previous work in the understanding of object-oriented application behaviour where the student visualizes and interacts with other virtual actors in a role-play simulation that performs the execution of a piece of code. The tool for the creation of CRC cards and role-play mentioned above will also serve as an authoring tool for the generation of the simulation scenarios performed in the virtual environment.

*Acknowledgements*—This work is supported by the Spanish Committee of Education & Science project TIN2006-15202-C03-03 and it is partially supported by the Comunidad de Madrid Education Council and Complutense University of Madrid (consolidated research group 910494). We would also like to thank all of the participants in the experience for their time and feedback, especially Marco Antonio Gomez for his help in carrying out the experiments and Javier Arroyo for his support with the statistics.

## REFERENCES

1. D. Schmidt, A family of design patterns for flexibly configuring network services in distributed systems, in *Proc. International Conference on Configurable Distributed Systems*, Annapolis, Maryland, (1996), pp. 124–135.
2. D. Schmidt, Using design patterns to develop reusable object-oriented communication software, *Communications of the ACM*, **38**(10), 1995, pp. 65–74.
3. M. Vokac, Defect frequency and design patterns: an empirical study of industrial code, *IEEE Transactions on Software Engineering*, **30**(12), 2004, pp. 904–917.
4. M. Hahsler, A quantitative study of the adoption of design patterns by open source software developers, in *Free/Open Source Software Development*, S. Koch, Ed. IGI Publishing, Wien, Austria, (2005), pp. 103–123.
5. K. Beck, R. Crocker *et al.*, Industrial experience with design patterns, in Proc. 18th International Conference on Software Engineering, Berlin, (1996), pp. 103–114.
6. E. Gamma, R. Helm *et al.*, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley Professional, Massachusetts, (1995).
7. Computing Curricula 2001, *Computer Science, The Joint Task Force on Computing Curricula,* ACM / IEEE Computer Society, 15 December 2001.
8. Software Engineering 2004, *Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering*, The Joint Task Force on Computing Curricula. ACM / IEEE Computer Society, 23 August 2004.
9. E. Wallingford, Toward a first course based on object-oriented patterns, in *Proc. 27th SIGCSE Technical Symposium on Computer Science Education*, Philadelphia, Pennsylvania, (1996), pp. 27–31.
10. O. Astrachan, G. Mitchener *et al.*, Design patterns: an essential component of CS curricula, in *Proc. 29th SIGCSE Technical Symposium on Computer Science Education*, Atlanta, Georgia, (1998), pp. 153–160.
11. V. K. Proulx, Programming patterns and design patterns in the introductory computer science course, in *Proc. 31st SIGCSE Technical symposium on Computer Science Education*, Austin, Texas, (2000), pp. 80–84.

12. C. Alphonce and P. Ventura, Object orientation in CS1-CS2 by design, in *Proc. 7th Annual Conference on Innovation and Technology in Computer Science Education*, Aarhus, Denmark, (2002), pp. 70–74.
13. R. Pecinovský, J. Pavlícková *et al.*, Let's modify the objects-first approach into design-patterns-first, in *Proc. 11th Annual Conference on Innovation and Technology in Computer Science Education*, Bologna, Italy, (2006), pp. 188–192.
14. C. Alphonce, M. Caspersen, *et al.*, Killer 'killer examples' for design patterns, in *Proc. 38th SIGCSE Technical Symposium on Computer Science Education*, Covington, Kentucky, (2007), pp. 228–232.
15. C. Chambers, B. Harrison, *et al.*, A debate on language and tool support for design patterns, in *Proc. 27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Boston, Massachusetts, (2000), pp. 277–289.
16. M. J. Clancy and M. C. Linn, Patterns and pedagogy, in *Proc. 30th SIGCSE Technical Symposium on Computer Science Education*, New Orleans, Louisiana, (1999), pp. 37–42.
17. C. Nevison and B. Wells, Teaching objects early and design patterns in Java using case studies, in *Proc. 8th Annual Conference on Innovation and Technology in Computer Science Education*, Thessaloniki, Greece, (2003), pp. 94–98.
18. C. Nevison and B. Wells, Using a maze case study to teach object-oriented programming design patterns, in *Proc. 6th Australasian Computing Education Conference*, Dunedin, New Zealand, (2004), pp. 207–215.
19. J. Hamer, An approach to teaching design patterns using musical composition, in *Proc. 9th Annual Conference on Innovation and Technology in Computer Science Education*, Leeds, (2004), pp. 156–160.
20. P. V. Gestwicki, Computer games as motivation for design patterns, in *Proc. Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education*, Covington, Kentucky, (2007), pp. 233–237.
21. H. Bærbak-Christensen, Frameworks: Putting design patterns into perspective, in *Proc. 9th Annual Conference on Innovation and Technology in Computer Science Education*, Leeds, (2004), pp. 142–145.
22. JHotDraw, JHotDraw Website, WWW: http://www.jhotdraw.org/.
23. B. Venners, How to use design patterns. A conversation with Erich Gamma, Part I, in *Leading-Edge Java*, (2005).
24. A. Schmolitzky, A laboratory for teaching object-oriented language and design concepts with teachlets, in *Proc. 20th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, San Diego, CA, (2005), pp. 332–337.
25. P. A. Kirschner, J. Sweller *et al.*, Why minimal guidance during instruction does not work: An analysis of the failure of constructivist, discovery, problem-based experiential and inquiry-based teaching, *Educational Psychologist*, **41**(2), 2006, pp. 75–86.
26. Active Learning in Engineering Education, http://www.ale.tudelft.nl/.
27. M. Fowler, *Refactoring: Improving the Design of Existing Code*, Addison-Wesley Professional, (1999).
28. J. Kerievsky, *Refactoring to Patterns,* Addison-Wesley Professional, (2004).
29. S. Smith, S. Stoecklin *et al.*, An innovative approach to teaching refactoring, in *Proc. 37th SIGCSE Technical Symposium on Computer Science Education*, Houston, Texas, (2006), pp. 349–353.
30. S. Stoecklin, S. Smith, *et al.*, Teaching students to build well formed object-oriented methods through refactoring, in *Proc. 38th SIGCSE Technical Symposium on Computer science education*, Covington, Kentucky, (2007), pp. 145–149.
31. K. Beck and C. Andres, *Extreme Programming Explained: Embrace Change*, 2nd edn, Addison Wesley, (2004).
32. J. Shore and S. Warden, *The Art of Agile Development*, 1st edn, O'Reilly Media, Inc., (2007).
33. J. Bergin, J. Eckstein *et al.*, Patterns for gaining different perspectives, in *Proc. 8th Conference on Pattern Languages of Programs*, Monticello, Illinois, (2001).
34. J. Börstler, Improving CRC-card role-play with role-play diagrams, in *Proc. 20th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*, San Diego, CA, (2005), pp. 356–364.
35. D. Bellin and S. Suchman-Simone, *The CRC Card Book,* Addison-Wesley, Massachusetts, (1997).
36. SPSS: Statistical Package for the Social Sciences, SPSS Inc., www.spss.com.
37. G. Jiménez-Díaz, M. Gómez-Albarrán *et al.*, Pass the ball: game-based learning of software design, in *Entertainment Computing—ICEC 2007*, vol. 4740, *LNCS*, L. Ma, M. Rauterberg and R. Nakatsu, Eds. Springer, Berlin, (2007), pp. 49–54.
38. G. Jiménez-Díaz, M. Gómez-Albarrán *et al.*, Software Behaviour understanding supported by dynamic visualization and role-play, *SIGCSE Bulletin*, **37**(3), 2005, pp. 54–58.

**Guillermo Jiménez-Diaz** is a teaching assistant in the Department of Software Engineering and Artificial Intelligence (DISIA) at the Complutense University of Madrid, Spain. He is a member of GAIA, the Group of Artificial Intelligence Applications at the UCM. His research interests include programming education and virtual learning environments. He has a Master's in learning methods to teach how to use frameworks and his Ph.D. relates to learning object-oriented programming in virtual environments.

**Mercedes Gómez-Albarrán** is an associate professor in DISIA at the Complutense University of Madrid, Spain. She holds the post of Academic Secretary at the Computer Science School of this University. She is a member of GAIA. Her main research interests

are virtual learning environments, innovative techniques and tools for programming teaching, and case-based teaching systems. She received her Ph.D. in Computer Science from the UCM.

**Pedro A. González-Calero** is an associate professor in DISIA at the Complutense University of Madrid, Spain. He leads GAIA. His main research interests are virtual learning environments, case-based reasoning, and serious applications of video games. He received his Ph.D. in Computer Science from the UCM.