Fostering Design Patterns Education: An Exemplar Inspired in the Angry Birds Game Franchise

DIOGO SOARES DA CRUZ SILVA, Rio de Janeiro State University (UERJ), Brazil
MARCELO SCHOTS, Rio de Janeiro State University (UERJ), Brazil
LETICIA DUBOC[*], La Salle - Universitat Ramon Lull, Spain

Design patterns aim at easing the reuse of design solutions, saving time in solving recurring problems and supporting code maintainability. However, teaching design patterns is challenging. One of the reasons lie in the students' difficulties, not only in understanding this concept in a high level of abstraction, but also in realizing how to effectively use design patterns. The use of games in software engineering education has been successfully enabling students to understand some contents more easily, besides acting as a motivational factor. However, the few existing approaches for teaching design patterns using game-oriented methods focus mostly in the pattern implementation, ignoring the fact that students need to first recognize the scenarios in which patterns can be used. In this sense, this paper proposes and evaluates an educational material that illustrates the use of design patterns using game scenarios from the Angry Birds franchise. An evaluation with students indicates an overall positive feedback, also pointing out some difficulties and barriers in abstraction levels and knowledge transferring between domains. There were also some suggestions for improvements.

CCS Concepts: • **Social and professional topics → Software engineering education**; • **Software and its engineering → Design patterns**.

Additional Key Words and Phrases: Design patterns, software engineering education, game development

ACM Reference Format:
Diogo Soares da Cruz Silva, Marcelo Schots, and Leticia Duboc. 2019. Fostering Design Patterns Education: An Exemplar Inspired in the Angry Birds Game Franchise. 1, 1 (April 2019), 20 pages. https://doi.org/10.1145/3364641.3364660

1 INTRODUCTION

Software engineering education is sometimes criticized for not properly preparing students for industry. Among the deficiencies of new graduates, a recurrent complaints is their difficulty in solving problems, which is directly linked to the lack of knowledge about existing solutions that could be applied to other contexts [13].

One of the most well-known problem solving strategies in software engineering is design patterns, i.e., generic, reusable solutions to recurrent problems in software design [7]. Most works on teaching design patterns focus on implementations through illustrative examples (e.g., [6]). Although this is undeniably relevant, recognizing scenarios in

[*]Also with, Informatics and Computer Science Dept. (COMPUT)Rio de Janeiro State University (UERJ).

Authors' addresses: Diogo Soares da Cruz Silva, MARVEL-SE, Informatics and Computer Science Dept. (COMPUT), Rio de Janeiro State University (UERJ), R. São Francisco Xavier, 524, 6025-A, Maracanã, Rio de Janeiro, RJ, Brazil, 20550-900, di_scruz@hotmail.com; Marcelo Schots, MARVEL-SE, Informatics and Computer Science Dept. (COMPUT), Rio de Janeiro State University (UERJ), R. São Francisco Xavier, 524, 6025-A, Maracanã, Rio de Janeiro, RJ, Brazil, 20550-900, schots@ime.uerj.br; Leticia Duboc, Department of Engineering, La Salle - Universitat Ramon Lull, Quatre Camins, 30 - Sant Jaume Hilari Building (2nd floor), Barcelona, Catalonia, Spain, 08022, l.duboc@salle.url.edu.

which such patterns can be applied is crucial for effective learning. Yet, studies point out that students find it difficult to determine which patterns to use in each situation, often misusing them [9, 12].

Motivation is a key element for the learning process. When it comes to design patterns, demonstrating its importance and usefulness in the software development can motivate students [8]. Many works have been proposed to improve the quality of software engineering education and make learning more attractive. Claypool and Claypool, for instance, propose using computer game-based projects, arguing that the fun factor makes classes more dynamic, arouses students' interest, and holds their attention, helping them to assimilate concepts through scenarios they are acquainted with [5].

This work presents an exemplar for teaching design patterns, inspired in the world-famous Angry Birds franchise, with the intent to ease the first contact of students with the concept of design patterns. It also presents its evaluation with Computer Science students, who were asked to identify opportunities of applying the patterns to their coursework projects. We aim to answer the following Research Questions (RQs):

RQ1: Does the use of the Angry Birds game help and motivate students to learn design patterns?
RQ2: Is the Angry Birds exemplar useful in identifying practical situations for applying design patterns?
RQ3: Is the Angry Birds exemplar helpful when modeling the selected design patterns?

The game was chosen according to the following criteria: game success, fun and ease of gameplay, number of editions, and variety of game elements (see Section 3). It is worth noting that there are certainly other subjects equally suitable for teaching design patterns, some of them outside of the gaming realm. We intend to assess the suitability of the Angry Birds exemplar rather than arguing that it is "better" than any other. Besides, we are not aware of an equivalent material using a world-famous game.

The remainder of this paper is organized as follows: Section 2 introduces the concept of design patterns and discusses the importance of teaching them. Section 3 presents the Angry Birds franchise. Section 4 exemplifies some design patterns in the Angry Birds context. Section 5 presents an evaluation of our work. Section 6 presents related works, and Section 7 presents some final remarks.

2 DESIGN PATTERNS

Design patterns help solving common problems on software development by specifying the decomposition of objects and their relationships [7]. They have been widely used in the software industry [3, 17]. The familiarity with design patterns allows developers to find solutions for recurrent problems faster and provide a common vocabulary among developers [1]. One of the most widespread set of design patterns is the one proposed by the Gang of Four (GoF) [7], composed by 23 patterns classified into three categories:

Creational: provide mechanisms for creating objects;
Structural: aggregate new functionalities to objects; and
Behavioral: handle objects' behavior, allowing the communication and the sharing of responsibilities.

2.1 Learning Design Patterns

Learning design patterns is an important step in the formation of Computer Science professionals [14]. To novice software developers, understanding where to apply such patterns can be particularly challenging. In a study conducted by Pillay [12], 22 undergraduate students had design patterns classes during six weeks, and most of them presented difficulties in determining which patterns to use in which situations. Ghazarian [9] observed that most students in a software engineering course misused design patterns.

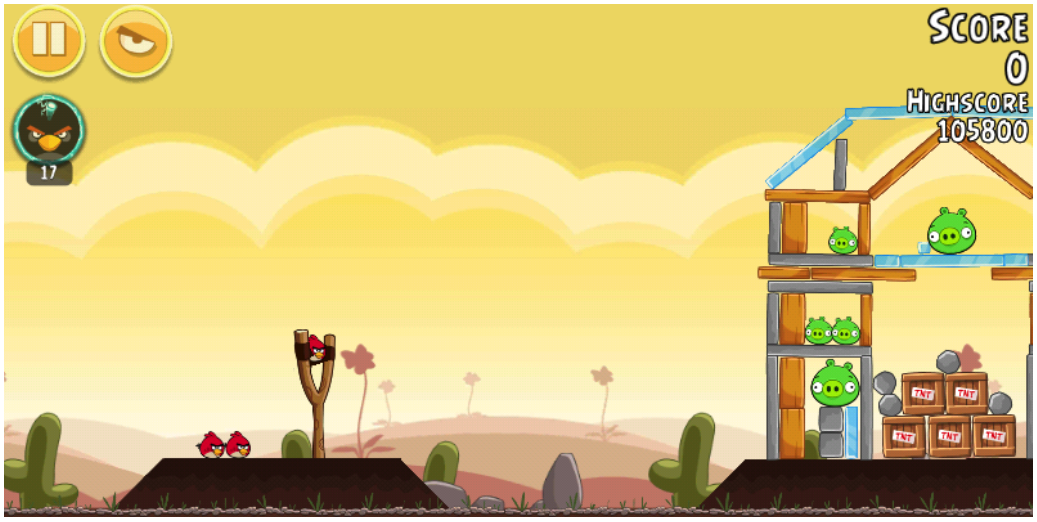

Fig. 1. A scenery in the classical version of Angry Birds

Design patterns are strongly linked to the concept of abstraction, i.e., the ability to group common features in a group of entities, ignoring details that do not belong to the problem domain. Despite its importance in object-oriented development, learning abstraction has been of great difficulty for undergraduate students [12]. According to Koster [10], games might ease the understanding of what abstraction is, due to their graphical appeal which allows people to see what was abstracted on their mind. Some works along these lines will be discussed in Section 6.

3 THE ANGRY BIRDS FRANCHISE

The first Angry Birds game [16] was launched in December 2009, initially created for the iOS platform. As of today, it is a franchise composed of 16 games. The original game tells the story of wingless birds, which use a slingshot to take revenge on green pigs that stole their eggs. The basic game elements are described as follows:

Pigs: vilans that hide in fortress and must be destroyed.
Blocks: create the fortresses and protect the pigs and must be destroyed to reach the pigs.
Birds: use a slingshot to destroy blocks and pigs.
Slingshot: used by the player, through the finger or the mouse, to throw the birds and destroy pigs and their fortresses.
Scenery: elements (slingshot, birds etc.) composing a game stage.
Stage: period in which the player starts to interact with the scenery until there are no more birds.
World: a set of sceneries/stages.

The gameplay is simple: a shot is defined by the strength and firing angle with which the slingshot releases the bird, aiming to collide it with the fortress where pigs hide, destroying both the structures and the pigs. The goal is to destroy all the pigs in a stage to unlock the next one. The game scenery also has elements to help the player, such as indestructible platforms the bird ricochets to reach places that cannot be reached in a straightforward shot. Figure 1 shows the scenery of a stage in the game.

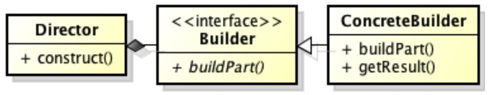

Fig. 2. The Builder pattern

The Angry Birds franchise has been chosen due to its enormous success (so that students would probably know the game), as well as its ease and fun gameplay. In addition, the game has several editions, offering a good scenario for discussing how to reuse common game elements and features without re-implementing them from scratch. Finally, the variety of game elements, interactions, and visual effects provide some clue into the complexity of the game development, allowing the identification of a number of design patterns.

4 GOF PATTERNS AND ANGRY BIRDS

This section presents part of the developed exemplar to teach design patterns[1]. We found 20 possible implementations out of the 23 GoF patterns [7] in the game franchise. We neither had access to the game's source code nor attempted to reverse engineer it. We only present possible pattern implementations, for didactic purposes. Due to space constraints, we illustrate three of them as follows.

4.1 Builder

The Builder pattern [7] separates the construction of a complex object from its complete representation, so that the same construction process can create different representations. Its elements are depicted in Figure 2 and described as follows.

Director is the constructor of the object who passes the information for object creation to the `Builder`.
Builder is an interface that creates parts of the final object through the `buildPart()` method.
ConcreteBuilder is a implementation of the `Builder` interface that returns the created object through the `getResult()` method.

In Angry Birds, sceneries can be created by using this pattern, as they share the same basic elements: background, birds, pigs, blocks etc. An external tool can build sceneries through a file scripts to ease the creation of stages, as shown in Figure 3. When the player selects a game stage, the corresponding file is read and sent to the `StageReader` class (representing the `Director` in the original pattern diagram), through the `createStage (File f)` method. The `StageReader` reads the file, generating a list of tokens that contains information about size, color, rotation, etc that may be used for the game physics' calculations. As the tokens are verified, they are sent as parameters to the `ConcreteStageBuilder` class (i.e., an implementation of the `Stage Builder` interface) through its corresponding methods, `buildPig()`, `buildBlock()`, `buildBird()`, or `buildBackground()`, which are then responsible for instantiating the objects and adding them to the stage. At the end of this process, the `getStage()` method from the `StageBuilder` is called by the `StageReader`, receiving the complete `Stage` object.

4.2 Observer

The Observer pattern [7] creates one-to-many relationships, so that an object notifies others when changes are made in its state. Figure 4 represents the class diagram of the pattern, composed of:

[1]The exemplar can be accessed at https://goo.gl/G12cwL

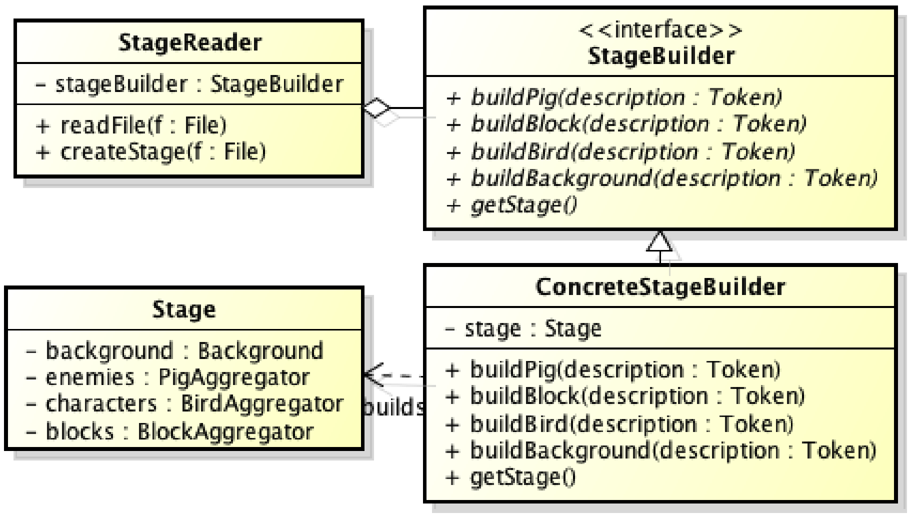
StageReader
- stageBuilder : StageBuilder
+ readFile(f : File)
+ createStage(f : File)
<<interface>>
StageBuilder
+ buildPig(description : Token)
+ buildBlock(description : Token)
+ buildBird(description : Token)
+ buildBackground(description : Token)
+ getStage()
Stage
- background : Background
- enemies : PigAggregator
- characters : BirdAggregator
- blocks : BlockAggregator
builds
ConcreteStageBuilder
- stage : Stage
+ buildPig(description : Token)
+ buildBlock(description : Token)
+ buildBird(description : Token)
+ buildBackground(description : Token)
+ getStage()

Fig. 3. Application of the Builder pattern

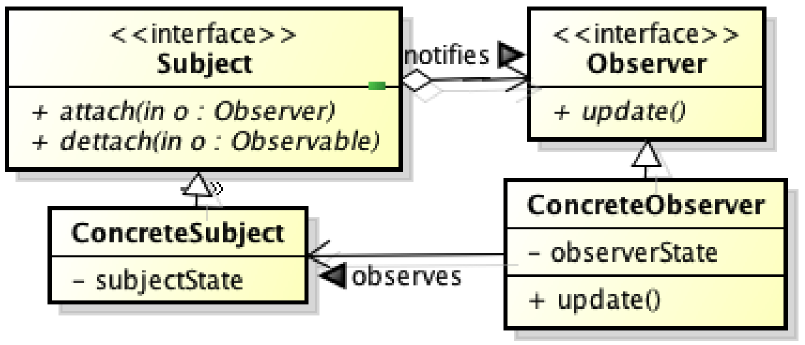

Fig. 4. The Observer pattern

Observer is an interface to be implemented by objects that should be *listeners* of (i.e., notified of) events in a `Subject`.

Subject is an interface whose implementations dispatch events of interest for the `Observer`. It contains methods to add and remove `Observers` from its the list of listeners and to notify the `Observer` objects about events they are interested in.

ConcreteObserver implements the `Observer` interface, keeping a reference of the `Subject` and implementing the method for updating its internal state, invoked as the Subject state changes.

ConcreteSubject implements the `Subject` interface, containing the states that matter to its list of *listeners* (the `Observers`), and implementing the method that notifies them about states change.

In Angry Birds, when a block or a pig is destroyed, an event is dispatched. The event induces changes in the state of some objects, e.g., how some screen elements are displayed. For instance, when the player destroys blocks and/or pigs, the game adds score points.

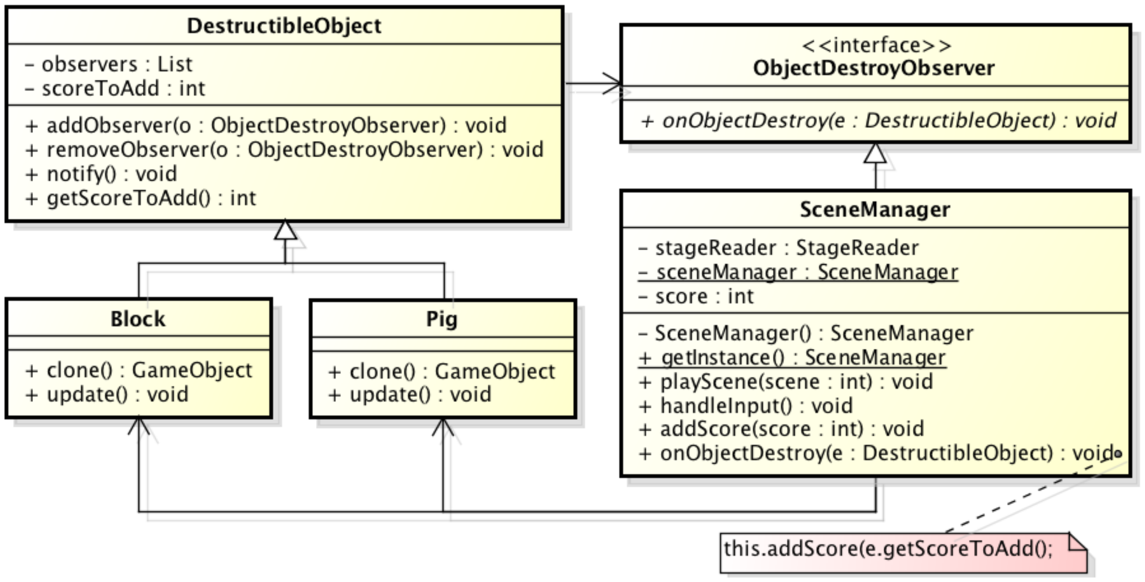
DestructibleObject
- observers : List
- scoreToAdd : int
+ addObserver(o : ObjectDestroyObserver) : void
+ removeObserver(o : ObjectDestroyObserver) : void
+ notify() : void
+ getScoreToAdd() : int
<<interface>>
ObjectDestroyObserver
+ onObjectDestroy(e : DestructibleObject) : void
SceneManager
- stageReader : StageReader
- sceneManager : SceneManager
- score : int
- SceneManager() : SceneManager
+ getInstance() : SceneManager
+ playScene(scene : int) : void
+ handleInput() : void
+ addScore(score : int) : void
+ onObjectDestroy(e : DestructibleObject) : void
Block
+ clone() : GameObject
+ update() : void
Pig
+ clone() : GameObject
+ update() : void
this.addScore(e.getScoreToAdd();

Fig. 5. Application of the Observer pattern

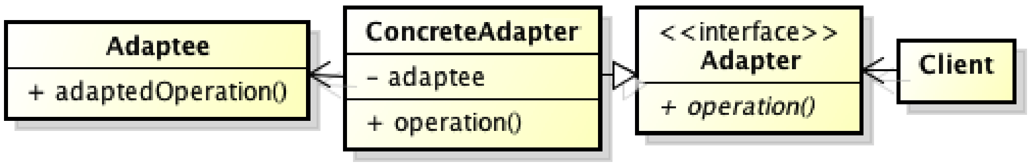

Fig. 6. The Adapter pattern

The Observer can be implemented by dispatching notifications to all the elements in the scene interested in "knowing" if a block or pig has been destroyed. Figure 5 illustrates the diagram of this implementation, followed by the description of its classes.

In order to reduce coupling between the Pig and Block elements (`DestructibleObject`) and the class responsible for managing objects and scene behavior (`SceneManager`), a notification can be sent to the `SceneManager` when a pig or block is destroyed, which will handle the calculation and show the total score of the player.

4.3 Adapter

The Adapter pattern [7] allows the communication between two incompatible interfaces, without changing the class that should be adapted. Figure 6 shows the class diagram for the pattern.

Client keeps an instance of the `Adapter` interface.
Adaptee is the class that must be adapted to the domain/problem.
Adapter (or *Target*) is an interface that defines domain-specific methods (the `operation()`), to be invoked by the `Client`.

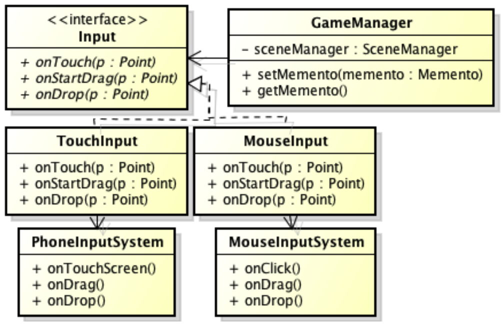

Fig. 7. Application of the Adapter pattern

ConcreteAdapter implements `Adapter` and has an instance of `Adaptee`, invoking `adaptedOperation()` from the `operation()` method, making the necessary adaptations.

Due to its success, a desktop version of the Angry Birds game was launched for Windows PCs. In order to reuse its main elements, it would be necessary to adapt the original mobile touch input to mouse interactions. Figure 7 represents a possible application of this pattern, followed by an explanation of the model elements.

The `PhoneInputSystem` and `MouseInputSystem` are the classes to be adapted. They are part of the data input system for the platform (mobile and desktop), i.e., they are equivalent to the `Adaptee` class in the original pattern diagram. The `TouchInput` and `MouseInput` (`ConcreteAdapter` in the original diagram) implement the `Input (Adapter)` to encapsulate the methods from `PhoneInput System` and `MouseInputSystem`, respectively. Thus, the `GameManager` does not need to know who consumes the input data – it only keeps an instance of the `Input` that is compatible with the platform used.

These three design patterns were among the ones taught to undergraduate students to evaluate our proposal.

5 EVALUATION OF THE EXEMPLAR

The evaluation of the proposed exemplar used the three research questions (posed in Section 1) as a basis: *(RQ1) Does the Angry Birds exemplar help and motivate students to learn design patterns?? (RQ2) Is the Angry Birds exemplar useful in identifying practical situations for applying design patterns? (RQ3) Is the Angry Birds exemplar helpful when modeling the selected design patterns?*

5.1 Context

The study took place at a public Brazilian university, with third-year Computer Science undergraduate students. It was performed as part of a 20-week long course on Systems Analysis and Design. The course applied project-based learning as methodology [4], and also took inspiration from the social-constructivism [20] and significative learning [2] theories. Students progressively worked on course-long projects of their choice and deliver (every 1 to 2 weeks) increments of the analysis, design, and implementation of their systems. These projects were used as the target systems in which the student should apply the design patterns learned through the Angry Birds exemplar. This approach is aligned with the listed theories due to, among others, the development of competence for transferring knowledge among different domains.

As we did the study towards the end of the course, the students had a good understanding of their domains and a reasonable design of their systems. Participation (except for questionnaires) was mandatory, since the content was on the syllabus of the course. The study involved 26 students, who filled out a characterization questionnaire and an signed informed consent prior to the study.

In the characterization questionnaire filled out before the experiment, all of them stated they expected to graduate in the current or following year. We next summarize the characteristics taken from multiple-choice questions, to which students could provide more than one answer. Hence the percentages do not add up to 100%.

When asked about their object-oriented (OO) experience, 57.69% of the students said that they had attended an object-oriented development course; 53.85% read materials about OO development; 23.07% developed OO software in the context of a course; 19.23% developed OO software for personal use, 19.23% developed OO software for industry, and 7.69% never developed OO software.

With respect to their experience with game development: 26.92% students had no knowledge on the subject, 65.38% read about game development, 15.38% developed games in a course, one (3.85%) developed games for personal use, and one (3.85%) developed games for industry. Some had more than one type of experience.

We also asked to which extent they knew Angry Birds: 80.77% of them had played it, 11.54% were currently playing it, and 7.69% saw people playing it but never played it, and one (3.84%) stated he/she did not know the game.

5.2 Choice of Study Design

We faced a few decisions when designing the evaluation. We briefly discuss the main alternatives considered and the rationale for the chosen design. We took a few considerations into account:

1. We created the exemplar using a world-known game in order to ease the students' first contact with the concept of design patterns;
2. The study could take up to two classes with 1 hour and 40 minutes each, due to the course planning for the semester;
3. During the study, students should identify situations for applying patterns in their projects, model the solutions, and fill up questionnaires while their impressions were fresh in their minds;
4. Whatever the alternative adopted, the study would be subject to threats to validity. These are discussed in Section 5.4.
5. We could not compare with previous years, as design patterns had been added to the syllabus for that course on that year.

The first alternative was to divide students into two groups, control (receiving a lecture on design patterns using "classic" examples, such as the ones on the GoF Design Patterns book [7]) and experimental (in a lecture using the Angry Birds exemplar). This would allow a comparison in the perception and learning of both groups. To this end, the classes should be given simultaneously by different lecturers, to avoid students exchanging information about the study. However, for our small population, this design would be subject to two important threats: the difference in the student's abilities and in the lecturer's teaching style. Another reason why we did not feel comfortable with this choice was that the groups would be exposed to different teaching styles, one possibly better than the other. The impossibility to dedicate more than two classes for the study would not allow us to exposing both groups to both teaching styles.

The second alternative was to first introduce the concepts of design patterns using "classic" examples and then with the Angry Birds exemplar, using follow-up questionnaires after each class, so that students would have something to compare our exemplar with. We chose not to adopt this design for the following reasons: (i) The explanations, the identification of the situations, the modeling, and the follow-up questionnaires should fit within the two classes. This would either give the students too little time for these tasks or reduce even more the number of patterns presented. (ii) It would be difficult to evaluate whether the ease on students' learning was due to the repetition of the concepts in two different scenarios or by the exemplar itself. (iii) The comparison between the two approaches ("classic" examples vs. Angry Birds) would be highly dependent on the choice of the examples themselves. For example, the examples described by Gamma et al. are appropriate, but they handle very distinct domains. It would be difficult to assess whether the differences in impressions would be due to the variety vs. consistency of the examples, the familiarity of students with the domains (or lack thereof), or other factors. (iv) We wanted the exemplar to be the students' first contact with design patterns, so giving other examples beforehand would be inconsistent with our objectives.

The third design option was to introduce design patterns concepts with the Angry Birds exemplar for all students. The advantages of this approach are: (i) the exemplar would be the students' first contact with design patterns; (ii) their learning would be more directly related to the educational material being evaluated; (iii) all students would be exposed to the same teaching style; (iv) students could be exposed to a greater number of patterns, having more time to work on the tasks. On the other hand, the students would have nothing to compare the Angry Birds exemplar with, not allowing us to evaluate whether the exemplar was better than other examples or whether what helped the student was the Angry Birds exemplar (or simply the use of concrete examples). These drawbacks had a relative importance to us, as in this experiment we were more interested in evaluating the suitability of exemplar for teaching rather than whether it was better than other materials. Hence, despite these drawbacks, we felt that this design was best fit to our purposes. Other studies in the area used similar approaches [11].

The teaching material was created and applied by the first and the second authors. The third author was the lecturer of the course at the time. Students were all well-versed with design patterns. The experiment was designed by all authors (including the classes timing, the evaluation questionnaire and the grading criteria), but the third author did not get involved in the analysis of results to avoid bias and so that students would not be afraid to criticize their own course lecturer. The second and the third authors carefully evaluated the class design - the former had thought the subject in a different course before, and the latter was familiar with the students and their abilities. The first author was chosen to give the class, being the most familiar person with the material, and he was not a lecturer at the university, so students would not have preconceived ideas or be afraid to provide true feedback.

5.3 Study Description

We carried out the study over two days, during the Systems Analysis and Design class times. It consisted on teaching the students selected design patterns and asking them to identify situations in which these patterns could be applied on their coursework project and model their solutions. Students did the exercise individually.

Since the class time frame was limited to 1 hour and 40 minutes, we could not present all the design patterns at once; thus, we selected 2 of each category (creational, structural, and behavioral), based on our pre-identification of candidate situations for the application of such patterns in the context of the students' coursework. The chosen patterns were: Singleton, Builder, Observer, Strategy, Adapter, and Decorator.

In the first day, we used a presentation of slides to explain the basic concepts of design patterns, the motivation and benefits of their use, and a brief history of GoF patterns. We advised them to think in advance (during the explanation of each pattern) whether and how such pattern could fit into their coursework projects. Then, we explained each aforementioned design pattern in its original form, followed by an example of its use in the context of the Angry Birds games. Each explanation lasted about 5-6 minutes, and students were given another 5-6 minutes for trying to identify and textually describe (in a sheet of paper) one or more usages of that pattern in their coursework. We repeated this process for each selected pattern. We answered students questions during the explanations, but not during the exercises. We also gave students a printout of the slides. Finally, we collected the exercises and asked them to fill out a follow-up questionnaire about their experience.

We asked students to explain their modeling rationale to facilitate the corrections. Design is a creative process, and diverging solutions may just represent different design alternatives (i.e., there is no “correct” modeling solution, but a number of coherent ones) [18].

On the second day, students were handed back their work from the previous class and the printout of the slides from the first class. Each pattern was quickly reviewed by showing its Angry Birds example, followed by 10 minutes to create diagrams for the situations identified earlier. We repeated this process for each pattern. In sequence, students were given around 20 minutes to complete or correct the situations identified or their diagrams. As in the first day, we only answered students questions during the explanations. In the end, the students filled out another follow-up questionnaire.

We asked students to answer questionnaires after each class to make sure that their impressions were still fresh when collecting their opinions about their experience (note that modeling is just attempted on the second class). Additionally, to incentivize students’ participation and engagement, they were told that their grades on the exercises could give them extra points on the final course grade.

After the classes, the authors of the paper who were not lecturers in the course graded the students’ answers. The students were given two grades for each design pattern: one for the identification of situations to which the design pattern could be applied and another one for the modeling of such situation. When a student had attempted a second answer (in the second round of the second class), we considered that answer. Grades vary from A to D, being A the highest and D the lowest (when the situation/diagram was incomplete, incorrect, or not done at all). To avoid bias, the grading was done without looking at the questionnaires answers.

We performed the data analysis using a spreadsheet to collect answers to the questionnaires and generate charts. One author created a codebook and used it to code the answers to the open questions in the questionnaire using the Saturate tool [15]. The other authors reviewed and discussed both the codebook and the coding. We calculated percentages according to the codes assigned to the remarks (e.g., if the student mentioned that "the game is well-known and helped the understanding of design patterns", we assigned the codes "well-known game" and "eased understanding"). The summarized findings were discussed among all the authors.

5.4 Limitations and Threats to Validity

Our exemplar has some limitations. Most importantly, we elaborated the design patterns on the Angry Birds game based on our knowledge of the game, without access to its source code. Thus, one cannot assert that the patterns mentioned in this work are actually implemented in the game, especially in the way they are described. Besides, we did not find situations that refer to the patterns Bridge, Composite, and Interpreter, but that does not mean that they are not present. Finally, the game may present other uses of the patterns.

We identified the following threats to validity [19] in our study:

Construct validity threats: In our study, we wanted to understand whether the proposed exemplar helped and motivated students to learn design patterns. Our analysis relied heavily on two follow-up questionnaires, whose answers reflected students’ perceptions about their learning, which might differ from their actual learning. To reduce this threat, we have also graded the students' exercise and compared their perceptions to their results. It is important to note that the lecturer did not grade the exercises, to avoid bias due to preconceptions about students' abilities.

Another point worth discussing is that questionnaires and exercises were not anonymized. This imposes a threat because students might not have answered them truthfully, since one of the authors was their lecturer and they could be afraid of any negative influence in their grades. This decision was taken for the following reasons: (i) to facilitate the execution of the study, as we needed to quickly hand back the exercises to the students in the second class; (ii) to facilitate the data analysis, by comparing the characterization and follow-up questionnaires with their exercises; (iii) because any alternative form of anonymizing the answers (e.g., an identification code) was unlike to convince them, as the lecturer was familiar with their coursework projects.

Yet, for us, it was more important to propose to the students a project they were familiar with than to anonymize questionnaires. To reduce this threat, we assured them that their lecturer would not correct the exercises or have access to their individual, nominal answers to the questionnaires (only the summarized and anonymized results), and that their answers would not influence their current grades (instead, they could earn extra points). Despite the assurances given, students may have been reluctant to give sincere feedback. Finally, although we reinforced the importance of doing their best on the exercise, students who did not need extra points may have been less engaged.

Internal validity threats: Our study asked students to apply the patterns on their coursework project. Their ability to do so depends on a number of factors. For instance, students were working on different projects, all in the same domain but with varying functionality, and this might have influenced their perceptions and grades. In fact, one student stated later that it was difficult to find the application of patterns in his/her particular project. As stated by Ghazarian [9], this situation can lead students to try to force a pattern to a problem that may not actually benefit from it. To reduce this threat, we studied all projects in advance for selecting the patterns to be taught in the study. Yet, our perception of the coursework projects might contrast with the students'. Other potential influence factors include the students' previous domain knowledge and experience with OO development, their ability to assimilate new knowledge, and their ability to work under time pressure. Some factors were circumvented taking the characterization questionnaire into account when interpreting the results.

External validity threats: Our study was performed with 26 undergraduate Computer Science students in a single university. Specific characteristics of the students' curriculum framework (such as the heavy load on Math-related subjects in the first two years, leaving other Computer Science subjects to be taught later in the course) limit the generalization of the results. Furthermore, there are different ways to present the same exemplar, e.g., with more time and detail. Despite these particularities, we believe that our proposal can also be helpful for other universities and/or students. Yet, further experiments are required to confirm that.

Conclusion validity threats: Due to the sample size, we are not able to perform any statistical tests or advanced analysis that would allow a cause-effect assessment. Thus, we assume this as a major threat to our experiment that we are not able to circumvent.

5.5 Results and Discussion

The analysis uses the data collected from the two follow-up questionnaires[2] and the grades given to the exercises, contrasted with the characterization questionnaire when applicable. The results are presented according to the research questions listed previously.

[2]All questionnaires can be found in https://goo.gl/G12cwL

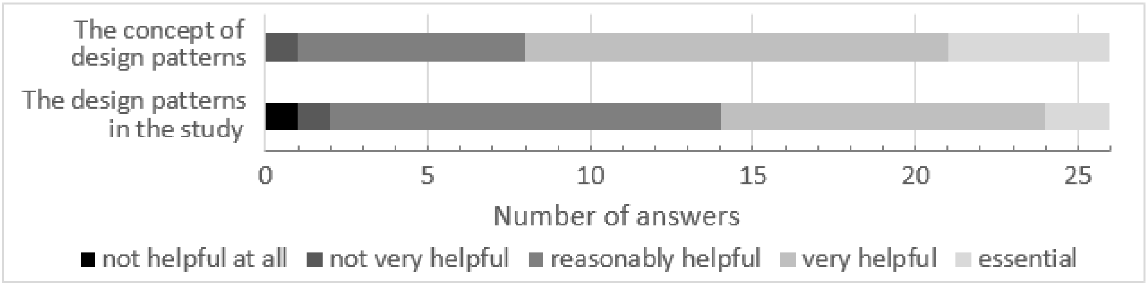

Fig. 8. Helpfulness of the Angry Birds game to understand design patterns

RQ1: *Does the use of the Angry Birds game help and motivate students to learn design patterns?*

In the follow-up questionnaire, the 26 students were asked two questions about their understanding of design patterns. The first one looked into whether they believed that the Angry Birds game helped them to understand the concept of design patterns (i.e., reusable solutions to recurrent problems). The answers are depicted in Figure 8. Nearly half of the students (12, 46.15%) answered that the exemplar was at least "very helpful" to that end. An equal number of students felt that the game was "reasonably helpful". Only two students stated that the game was of little or no help to understand the concept of design patterns. The second question asked whether the game helped to understand the explained patterns. All students except one (25, 96.15%) felt that the game was at least "reasonably helpful" to that end. The remaining student felt that the game was of little help. From the answers provided, most students perceived that Angry Birds helped them to understand design patterns.

In order to find out whether the Angry Birds game motivated the study of design patterns, we started by looking into the students' perceptions on the subject (Figure 9). All students except one considered design patterns "very helpful" or "essential" to software development. In addition, when asked what they thought of design patterns as a topic in the course, most students (20, 76.92%) considered the subject "interesting", while one student was indifferent to the topic. None of them said the topic was "boring" or "unbearable", but the student who declared indifference might have had this opinion, being reluctant to say so.

We also asked if students thought that the Angry Birds exemplar helped to understand the utility of design patterns (i.e., how practical they are), and their benefits (i.e., what are the gains of applying them) (see Figure 9). Even though more than half of the students (15, 57.69%) stated that the game was at least "very helpful" to understand the design patterns benefits, a considerable number (11, 42.31%) thought the game has helped "reasonably" or "a little". We could not identify whether this was due to the oral explanation, the exemplar itself, or the game domain (which may be difficult for some to grasp). With respect to understanding the utility of design patterns, all students stated that the game was at least "reasonably helpful" to this end, most of them (20, 76.92%) stating that it was "very helpful" or "essential".

We also asked some open questions, discussed as follows. The totals (and percentages) do not add up to 26 (100%) because each student's answer can comprise more than one aspect.

When asked what they liked the most and the least in using Angry Birds to teach design patterns, over half of the students (14, 53.85%) stated that the game eased the understanding of the topic. They felt that the examples were good and made the explanation clearer. One student said that the class were more dynamic. 13 (50%) liked the choice of Angry Birds because it is well-known (7, 26.92%), easy (3, 11.54%) and has a visual appeal (3, 11.54%). One student said

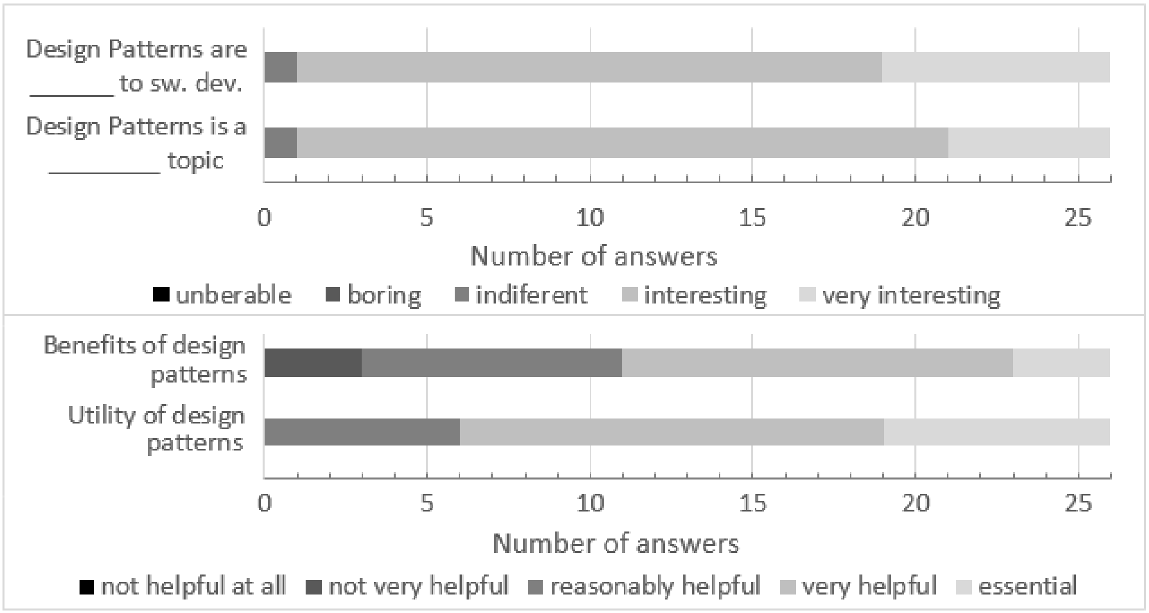

Fig. 9. Students' perceptions on the use of Angry Birds

it was interesting to think about the design of a computer game. Some (4, 15.38%) praised the use of a system from a known domain.

Most students (18, 69.23%) declared that they would not change anything in the approach. Some (5, 19.23%) would like more examples for each pattern, within and outside the games context. Indeed, presenting one usage of a pattern would not be enough to help depicting it as generic and reusable. Two students suggested minor changes in the slides, and one did not answer the question.

Finally, we asked them to rate, in a scale of 1 to 5 (being 1 the highest level of dissatisfaction and 5 the highest level of satisfaction), how they liked the idea of using games in a software engineering course. Results are shown in Figure 10. The majority of students (25, 96.15%) chose the top two ratings. Only one student showed indifference, attributing the rating 3, and none has shown dissatisfaction. The students were then asked to use the same scale to indicate how they would recommend the use of this material in others classes or universities. Again, nearly all students (25, 96.15%) responded with the top two ratings, while one student remained indifferent.

In summary, most students declared to understand the benefits of design patterns and found it interesting and relevant to software development. They also liked the use of a game to illustrate the application of design patterns (with a good acceptance of Angry Birds), and would recommend it. Because these perceptions show an overall positive evidence of the students' motivation, we believe that RQ1 can be answered positively for our sample.

RQ2: *Is the Angry Birds exemplar useful in identifying practical situations for applying design patterns?*

We started by asking how much the Angry Birds game helped to understand in which situations patterns could be applied. As Figure 11 shows, most students (20, 76.92%) found that the game was at least "very helpful" to that end. Some students (4, 15.38%) thought it offered a reasonable help, while 2 (7.69%) felt it offered little help.

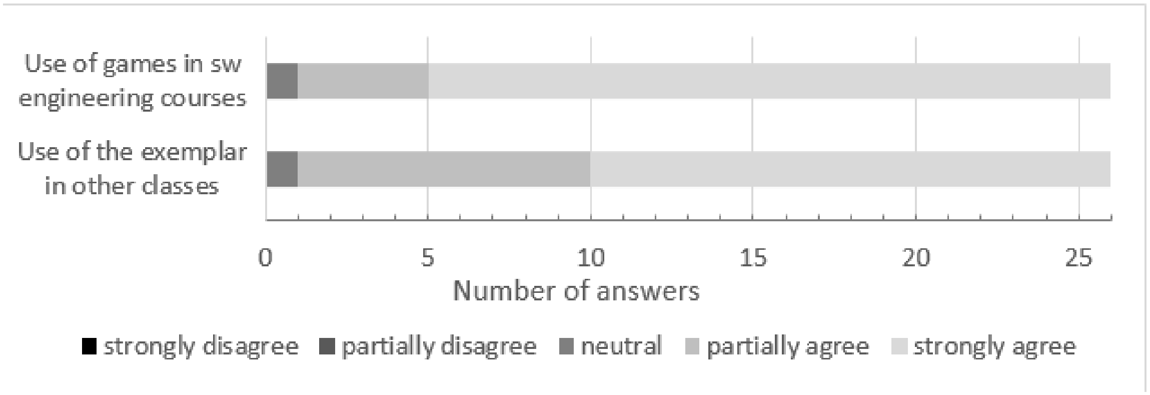

Fig. 10. Use of games in software engineering training

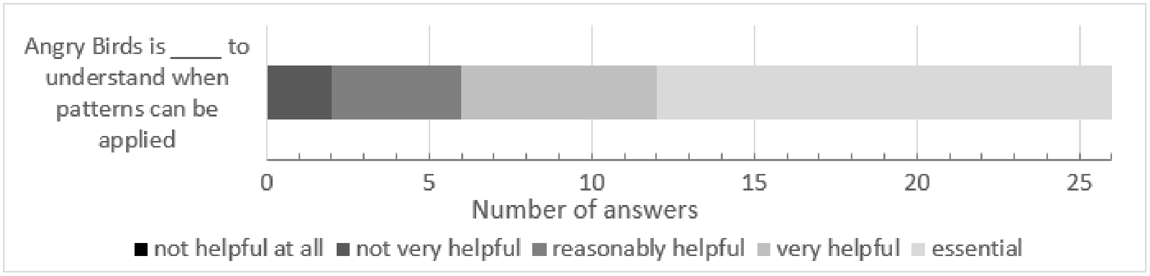

Fig. 11. Use of Angry Birds to understanding situations in which design patterns can be applied

It is interesting to compare the students' perceptions with their grades in the exercise (the grades were transposed to an ordinal scale, being A=3, B=2, C=1, and D=0). The grades show that most students were able to identify patterns properly: 14 students (53.85%) graded 60% or more (two of them achieving 100% and another 94.4%). Other 9 students (34.61%) graded between 40% and 60%, and 3 students (11.54%) achieved less than 40%. The grades show a "mismatch" between their perception and their actual understanding. A possible reason for this is that students were asked to identify situations in a very short time frame (5-6 minutes), right after an equally short explanation of the pattern.

We also asked the students to rate three of the elements used to introduce each design pattern: the oral explanation, the conceptual diagram of the pattern, and the diagram instantiated with the Angry Birds exemplar. As it can be seen in Figure 12, about half of the students (14, 53.85%) considered the oral explanation at least "very helpful" to identify scenarios in which a design pattern could be applied. On the other hand, 12 (46.15%) stated that the explanation was "not very helpful" or was "not helpful at all". This was somehow surprising, since the oral explanation aimed at clarifying the concept and usage of each pattern. However, a number of students commented afterwards that the explanation was hard to understand (some could not hear it loud enough) or was too short. One student added that he/she needs a strong theoretical explanation beforehand in order to understand the practice.

Almost half of the students (12, 46.15%) thought that the conceptual diagram was at least "very helpful" for identifying the application scenarios of the design patterns, while most of them (14, 53.85%) found it was "not very helpful" or "not helpful at all". However, the vast majority of students (24, 92.31%) considered the diagram instantiated with the Angry

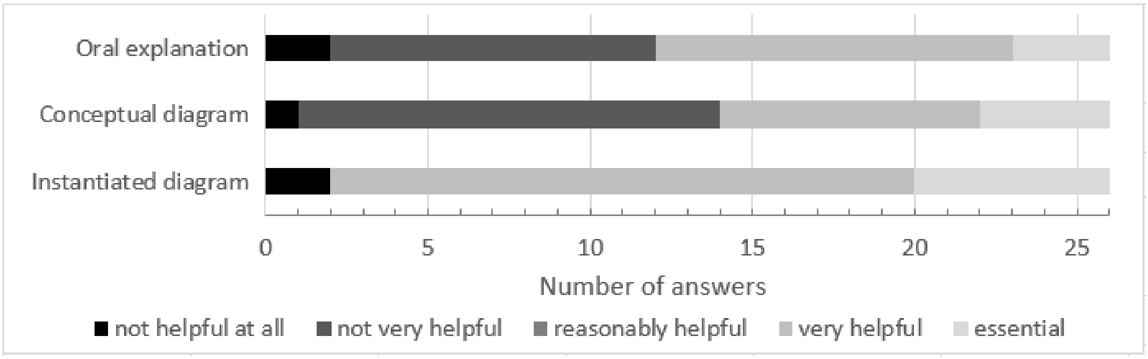

Fig. 12. Helpfulness of elements in the identification of situations to apply design patterns

Birds "essential" or "very helpful" for the same end. An explanation for these results is that the instantiated diagram provides a concrete example of the pattern in context, and tends to be more useful to illustrate situations of use than the conceptual diagram. Only 2 students (7.69%) thought that the Angry Birds exemplar did not help at all (although they praised the approach in their answers to other questions).

In an open question to indicate what helped the most in identifying situations to apply the patterns, most students (18, 69.23%) consistently answered that the instantiated diagram was more useful. Some students (4, 15.38%) preferred the oral explanation, and 2 (7.69%) found the conceptual diagram more helpful. One student did not answer this question, and another provided a different answer. Besides these answers, we also obtained 3 answers from different students (1 from the latter student) mentioning other elements that were considered much useful: the student's own description of the patterns (made during the explanation), a parallel with the Angry Birds, and the game screenshots. Comparing with the answers to the question on the helpfulness of each element, all the students who considered the conceptual diagram as the most helpful element also considered the instantiated diagram at least "very helpful".

Finally, we asked students what they found more difficult: to identify the situations in which to apply the patterns or to model them. Just above half of the students (15, 57.69%) found more difficult to identify the situations. Reasons varied greatly: the uniqueness of the situations, difficulty in gaining an initial perception of the application, difficulty in finding an application in the coursework project, lack of time, excess of information, insecurity etc. One student mentioned that the game helped to illustrate the application of the pattern, but he/she found it difficult to switch domains. Finally, 4 students (15.38%) found it difficult to choose between similar patterns. This might be due either to problems during the explanation of the patterns or to the situations chosen to illustrate them.

In summary, although some students had issues in identifying situations to apply the patterns, most of them found the Angry Birds exemplar (in particular, the instantiated diagram) helpful to this end. Thus, we assume RQ2 can be answered positively, but we acknowledge that our evidence is not as strong as it is for RQ1.

RQ3: *Is the Angry Birds exemplar helpful when modeling the selected design patterns?*

To answer this question, we asked students to rate the same three elements (the oral explanation, the conceptual diagram, and the instantiated diagram) with respect to their helpfulness on the modeling of the design patterns. Figure 13 presents the results. Above half of the students (16, 61.54%) claimed that the explanation was "not very helpful" or "not helpful at all" when it comes to modeling. One possible reason may be because understanding the interactions

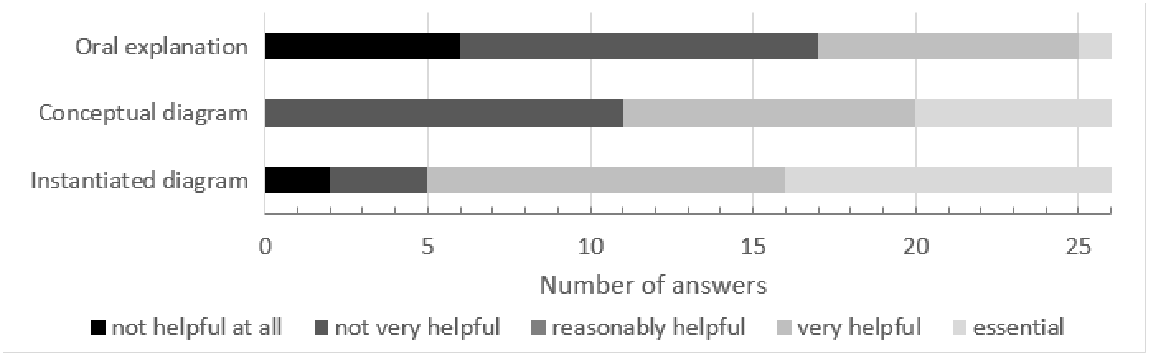

Fig. 13. Helpfulness of elements in modeling design patterns

between the diagram elements is not trivial. Another factor that might have contributed to this was already mentioned for RQ2: some students reported difficulty in hearing the oral explanation.

While 15 students (57.69%) said that the conceptual diagram was at least "very helpful" for modeling, 11 (42.31%) stated that it did not help that much. On the other hand, the majority of students (21, 80.77%) found that the diagram instantiated with the Angry Birds examples was at least "very helpful", while the remaining students (5, 19.23%) thought it helped a little or not at all. This result indicates the importance of exemplars for helping students in modeling the taught concepts (the "learning by example" approach).

When asked to choose the most helpful element to model the design patterns among the aforementioned ones, the majority of the students (18, 69.23%) asserted that the instantiated diagram was more helpful, followed by 6 (23.08%) who preferred the conceptual diagram. No student chose the oral explanation in this case. The 2 remaining students felt that none of these elements were particularly useful: one of them chose his/her own description of the patterns made during the explanation, while another one felt that the most helpful element was the game screenshots. These results indicate that the Angry Birds exemplar may have had a positive impact on helping students modeling the design patterns.

Regarding the difficulties in finding situations for applying the pattern against modeling it (asked previously), less than half of the students (11, 42.31%) found that modeling was harder than finding out where to apply the patterns. Some of them (3, 11.54%) admitted difficulties due to their lack of OO practice. Moreover, 2 students found it hard to understand how to translate the methods from the pattern/exemplar diagram to their own. One student said that although he/she could see a clear parallel between the situation identified and the explanation given, he/she could not find a fit when modeling. This student stated that he/she did not have much programming experience, which might have caused such difficulty. Students also mentioned other difficulties, such as insufficient examples, little time, insecurity, and bad explanation of the design patterns. One student did not explain his/her difficulties.

Curiously, in contrary to their perceptions, the students' grades showed that they had more difficulty in modeling than in identifying the situations in which to apply the patterns. This can be due to several reasons, including the limited time given and the complexity of some patterns. Many students created a model very different from the conceptual diagram/exemplar, demonstrating that they really did not understand the model. Only 7 students (26.92%) graded 60% or more in the modeling. One student who achieved 100% in the identification part also achieved 100% in the modeling.

He/she has experience in OO development in the industry. A similar percentage of students (6, 23.07%) stood between 40% and 60% of the grade. The remaining half (13) achieved less than 40% of the grade.

To complement our perception, we analyzed the number of students who had higher grades in the identification (20 students), the ones who had higher grades in modeling (one student), and the ones who had similar grades in both phases (5 students). This confirmed that, for this sample, the identification phase was easier. Yet, we must consider that design patterns is a complex subject, and students ideally should be given more time to reflect and discuss, especially in group. They performed the modeling under considerable time pressure and without any preparation study.

We emphasize that results contain some confounding factors. Students were involved in distinct projects with distinct degrees of functionality (some were in the same project, but no communication was allowed during the study), which may have hindered the application of patterns. Their backgrounds may also have strongly influenced their results (e.g., students with good OO modeling or development skills were more likely to identify appropriate situations and produce better models). Yet, we did not find an explicit correlation between background/training and the students' grades.

In summary, most students found the Angry Birds exemplar helpful for modeling the design patterns. Yet, a considerable amount of them felt difficulties in doing so, as revealed by their grades. Thus, we conclude that RQ3 has the weakest positive answer based on our sample and the previous RQs.

5.6 General Observations

Some students made informal comments about the experiment. Positive comments made after the first day of the experiment include "very important subject", "excellent approach", "this was a good experience", and so on. One student stated that "examples like this are usually not given in the courses". The only negative remark, mentioned by two students, was the lack of time to understand and apply the patterns (limited by the class time frame).

After the first day, students raised questions such as "what are the most used patterns?" and "how to apply patterns in any programming language?". We found this to be a good result, since they got engaged in the topic. Some students asked for more patterns using Angry Birds (provided by e-mail after the experiment). In the second day, students provided more positive comments: "excellent/good/interesting/attractive approach", "the game helped to understand the patterns when the definition was not enough", and "[aligning] theory and practice helps in learning". Two students complained about the lack of time.

Many students made suggestions for improvement. One asked for a broader scope to apply the patterns, since the coursework constrained the pattern application scope. Another student asked for more than one application of the same patterns within the Angry Birds game, while another one asked for examples of combinations of patterns.One student suggested to introduce the Angry Birds scenario before the pattern – a good remark, as it represents a more natural flow of explaining the problem before the solution.

Some students also provided feedback regarding the content of the exemplar and the current lack of source code examples. One student stated, "I felt the necessity of code samples to better understand the application of some more complicated design patterns". Another student agreed, stating that the study "lacked some more specific examples, even simpler ones, using source code". On the other hand, a student stated, "an abstraction of this type, as applied to the understanding of design patterns, is very didactic. Looking at source code does not always help understanding the pattern".

6 RELATED WORK

Claypool and Claypool [5] propose the use of computer games-based projects for software engineering education, arguing that the "fun factor" makes classes more dynamic and draw the students' attention. Their proposal is divided in ten modules for integrating digital games within the software engineering curriculum. They cover the basic concepts of software and game development, such as software design using UML, development management and planning, prototyping, design patterns focused on games, coding, and testing. The class is divided in teams, and each team must develop a different game. All the planning phases require all the teams, because an intention is to develop reusable code, and each team must reuse components developed by the other teams.

The main difference to our study is that their work take into account many aspects of software engineering simultaneously, including the game coding, so that there is an underlying risk of overlooking some important development steps. In addition, they do not provide enough support and evidence on the effectiveness of their approach for teaching design patterns. Our work, by focusing in the identification of the possible design patterns applications in the context of games, can stimulate and ease the teaching and learning processes, allowing students in not only grasping concepts, but also having in mind situations in which they can be applied. We also hope that, by having a better understanding of design patterns, the student may recognize them more easily.

Gestwicki and Sun [8] developed EEClone, a Java game that uses the State, Facade, Observer, Strategy, Visitor, and Singleton GoF patterns. The goal of the authors was that, during the semester, students could develop different games, using EEClone as a starting point. Through their methodology, three main goals were set: (i) students' understanding of object-oriented design and modeling; (ii) identification and application of specific design patterns addressed in the experiment; and (iii) ability to communicate about design patterns and software design in an efficient way.

The authors performed four evaluations to measure students' learning along the course: (i) initial exams, for assessing the students' previous knowledge about design, (ii) presentations to the class, for students to present the program and explain the significant changes made in the design and game implementation, (iii) supervisor observations regarding the learning improvements throughout the course, and (iv) the final exam, which evaluated the learning and knowledge about design patterns. In the end of the semester, students had a good knowledge of design patterns and OO design.

The main difference between our work and Gestwicki and Sun's [8] lies in learning process. Their strategy handles learning through observation/customization of design pattern implementations, while our teaching material supports learning design patterns through possible usages in a well-known game, stimulating students to identify similar situations in different contexts. We believe that the identification of situations is a stage that should precede the implementation – if a student knows the patterns but does not know where to apply them, the learning might not be effective.

7 FINAL REMARKS

This work offers a teaching material to illustrate the use of design patterns in a didactic and ludic way, and presents an evaluation of its use. The exemplar allows students to observe how to apply design patterns using a well-known game as an illustration scenario. The exemplar and the questionnaires used in the evaluation are openly available online.

Results indicate that students were overall satisfied with the proposed exemplar. To them, the use of games in teaching software engineering sounds promising due to their familiarity with the subject matter (even if they have never developed a game). The choice of the Angry Birds franchise also received a positive feedback. These results are encouraging, especially considering that they received a very limited first exposition to the design patterns topic. The exemplar has proven to be useful as the first contact of students with design patterns. Yet, there are other subjects, both within and outside the gaming context, that can be equally good. Our aim is to contribute to the software engineering teaching community with a ready-to-use, familiar, fun, and validated exemplar.

7.1 Future Work

We have several plans for future work: (i) perform further evaluations of our exemplar, with a greater number of classes in the courses of Systems Modeling, Software Engineering, and OO Programming, to obtain more evidence with respect to usefulness and efficacy. (ii) Identify more applications of the same patterns within Angry Birds and provide examples of pattern combinations (in response to the students' feedback). (iii) Augment examples with auxiliary resources (e.g., source code and sequence diagrams). (iv) Explore a more collaborative scenario, in which students can work together for solving problems, experiencing different games and other scenarios in a creative fashion. (v) Use open source implementations based on Angry Birds [3], allowing students to practice design patterns implementation. (vi) Explore "side effects" of design patterns, such as the drop in application performance [7]. (vii) Use games to teach other software engineering concepts. We are currently working on item *iii* and also tackling replicability.

ACKNOWLEDGMENTS

We thank Evandro Macedo, Rodrigo Bello and Anny Mayara for their feedback on the patterns. We thank CNPq and FAPERJ for the financial support. The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement no. 712949 (TECNIOspring PLUS) and from the Agency for Business Competitiveness of the Government of Catalonia.

REFERENCES

[1] Ellen Agerbo and Aino Cornils. 1998. How to Preserve the Benefits of Design Patterns. In *13th ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications (OOPSLA)*. 134–143.
[2] David Paul Ausubel, Joseph Donald Novak, Helen Hanesian, et al. 1968. Educational psychology: A cognitive view. (1968).
[3] K. Beck, R. Crocker, G. Meszaros, J. Vlissides, J. O. Coplien, L. Dominick, and F. Paulisch. 1996. Industrial experience with design patterns. In *18th International Conference on Software Engineering (ICSE)*. 103–114.
[4] Phyllis Blumenfeld, Elliot Soloway, Ronald Marx, and Joseph S. Krajcik. 2011. Motivating Project-Based Learning: Sustaining the Doing, Supporting the Learning. *Educational Psychologist* (2011), 369–398.
[5] Kajal Claypool and Mark Claypool. 2005. Teaching Software Engineering Through Game Design. In *10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE)*. 123–127.
[6] Elisabeth Freeman, Eric Freeman, Bert Bates, and Kathy Sierra. 2004. *Head First Design Patterns.* O'Reilly Media, Inc.
[7] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. 1994. *Design Patterns: Elements of Reusable Object-oriented Software.* Addison-Wesley, Boston, MA, USA.
[8] Paul Gestwicki and Fu-Shing Sun. 2008. Teaching Design Patterns through Computer Game Development. *Journal on Educational Resources in Computing* (2008), 1–22.
[9] A. Ghazarian. 2012. Work in Progress: Transitioning from Novice to Expert Software Engineers through Design Patterns: Is It Really Working?. In *Frontiers in Education Conference (FIE)*. 1–2.
[10] Raph Koster. 2013. *Theory of Fun for Game Design* (2nd ed.). O'Reilly Media, Inc.
[11] D. M. Nascimento, K. Cox, T. Almeida, W. Sampaio, R. A. Bittencourt, R. Souza, and C. Chavez. 2013. Using Open Source Projects in software engineering education: A systematic mapping study. In *2013 IEEE Frontiers in Education Conference (FIE)*. 1837–1843. https://doi.org/10.1109/FIE.2013.6685155
[12] Nelishia Pillay. 2010. Teaching design patterns. In *Southern African Computer Lecturers Association Conference (SACLA)*. 1–4.
[13] A. Radermarcher, G. Walia, and D. Knudson. 2014. Investigating the skill gap between graduating students and industry expectations. In *36th International Conference on Software Engineering (ICSE 2014)*. 291–300.
[14] IEEE Computer Society. 2014. *Guide of the Software Engineering Body of Knowledge - SWEBOK, version 3.0.* IEEE Press.

[3] See https://github.com/estevaofon/angry-birds-python for an example

[15] Jonathan Sillito. 2013. Simple collaborative qualitative analysis. http://www.saturateapp.com
[16] Rovio Entertainment Ltd. 2015. Angry Birds. http://www.rovio.com/en/our-work/games/view/1/angry-birds
[17] Dirk Riehle. 2011. Lessons learned from using design patterns in industry projects. In *Transactions on Pattern Languages of Programming II*, James Noble and Ralph Johnson (Eds.). Springer-Verlag, Berlin, Heidelberg, 1–15.
[18] Marcelo Schots, Claudia Rodrigues, Cláudia Werner, and Leonardo Murta. 2010. A Study on the Application of the PREViA Approach in Modeling Education. In *XXIX International Conference of the Chilean Computer Science Society (SCCC)*. Antofagasta, Chile, 96–101.
[19] Forrest Shull, Janice Singer, and Dag IK Sjøberg. 2008. *Guide to advanced empirical software engineering*. Vol. 93. Springer.
[20] Lev Semenovich Vygotsky. 1980. *Mind in society: The development of higher psychological processes.* Harvard University Press.