Frameworks: Putting Design Patterns into Perspective

Henrik Bærbak Christensen
Department of Computer Science
University of Aarhus
DK–8200 Aarhus N
Denmark
hbc@daimi.au.dk

ABSTRACT

Design patterns have made a strong impact on how object-oriented software is designed, implemented, and communicated in industrial projects. Teaching patterns is therefore of great importance. Patterns, however, have a natural catalogue-like nature, that easily misleads us into to lecturing them one at a time. This leaves an impression of patterns as isolated solutions to independent problems. We present our experience with trying to counteract this problem by using a well-engineered framework, JHotDraw, as a case study in how patterns work together to define a flexible and compositional software system of high quality.

Categories and Subject Descriptors

D.1.5 [Software]: Object-oriented Programming; D.2 [Software]: Software Engineering; D.2.2 [Software]: Design Tools and Techniques; K.3 [Computing Milieux]: Computers and Education

General Terms

Design

Keywords

Design patterns, frameworks, JHotDraw, reuse

1. INTRODUCTION

The notion of design patterns has had a major impact on the way object-oriented software is contemplated, designed, and realized over the last decade. As such, teaching both the underlying ideas of design patterns as well as providing the students with theoretical and practical experience with a range of the most often used patterns, is important in computer science education.

Design patterns, however, have one major drawback in a teaching context, namely their 'catalogue'-like quality. If not careful, you easily end up in a ‘daily grind’: “today we take observer, tomorrow composite, and next week strategy”. This is almost as mind-numbing and boring as reciting recipes from a cookbook one at the time. This is in stark contrast to the excitement that patterns caused when they first entered the software engineering stage.

Another problem with “reciting” design patterns is the danger that students perceive patterns as independent solutions to independent problems. This problem is made even worse because pattern structure is usually documented using UML class diagrams which of course describe structure using classes. However, design patterns are not just a set of classes! Patterns define roles and responsibilities of collaborating objects. While we can tell this to our students, chances are they will miss the point unless we demonstrate it to them.

Several authors have reported their experiences with design patterns in an educational setting [1, 2, 8, 10] and for instance the “killer example” workshops [7] demonstrate the continued effort to provide sound teaching material on the subject. In this paper, we argue in favour of presenting well-designed frameworks as exemplary use of design patterns. So far the subject of frameworks is sadly overlooked in teaching but we have found that introducing frameworks to students serves many important pedagogical points that will be outlined. The main point with respect to design patterns is that a framework makes it clear that design patterns work together, and that patterns really define roles.

We present observations and experience using a particular framework: JHotDraw, originally designed by Thomas Eggenschwiler and Eric Gamma, in a ‘programming in the large course’ taught at Department of Computer Science, University of Aarhus. We believe that other well-engineered frameworks may serve to illustrate pedagogical points concerning design patterns and framework design. However, we will focus on JHotDraw as we have been using it in our teaching.

2. PATTERNS AND FRAMEWORKS

In the course we first present a set of design patterns individually with accompanying exercises. The set consists of patterns that are found in the JHotDraw framework. We then turn to the subject of frameworks. Gamma et al. defines a framework as a set of cooperating classes that make up a reusable design for a specific class of software[3].

Frameworks are complex software systems, both structurally and not least behaviourally. JHotDraw [5] is an example of a medium-sized framework for developing 2-D semantic drawing editors. Several demonstration programs are available that build Petri diagram editors, UML class diagram editors [6], and ordinary 2D graphical figure editors based upon the framework.

JHotDraw 5.1 consists of more than 180 classes. A UML class diagram only showing the central abstractions is shown in figure 1. This is a design of much higher complexity than any single design pattern.
We use this figure in our teaching and ask the students in class the following question: How on earth are we going to understand this system? Are there any road-maps that will help us to know what is going on?

The answer is “yes”, and design patterns provide this road-map. The JHotDraw is basically a model-view-controller architecture with some additions. The diagram in figure 2 shows the same structural diagram but with the indication of what parts of the diagram that plays a role in the MVC patterns, along with other important patterns.

Thus, knowledge of the MVC patterns is the key to understanding both the structural as well as dynamical aspects of JHotDraw. For instance, MVC states that the view(s) should redraw when being notified about state changes in the model component. This behavioural structure is evident in JHotDraw where the central model abstraction, interface Drawing, maintains a list of DrawingChangeListeners that are notified whenever the drawing model component is changed. The central view component, DrawingView, is of course a DrawingChangeListeners.

MVC is basically an example of the Observer pattern. We will use the terminology defined in GoF [3]: the subject is the object that contains the state information while an observer is an object that must keep its own internal state synchronized.

2.1 Subject as Observer

The observer pattern is central in JHotDraw. JHotDraw is a semantic drawing editor meaning that graphical figures that have semantic bindings on each other are supported. As a simple example consider the association line connecting two classes in a UML diagram: moving one class has implications on the association line that must constantly be updated and redrawn to keep the two class figures connected. Any JHotDraw graphical object is a Figure instance and this interface also plays the role of subject in the observer pattern maintaining a list of FigureChangeListener objects that are notified upon state changes. This supports the aforementioned association line’s redrawing as it can subscribe to position changes in the two classes it connects. The nice catch is that the Drawing instance also simply subscribes to figure changes in all figures that it contains—so that it knows when to fire change events to its views. Thus Drawing is an example of a “double observer pattern”: it plays the role of subject with respect to DrawingViews and at the same time the role of observer with respect to Figures. This is elegant, simple to program, and ensures graphical consistency.

In a teaching setting it allows us to show how powerful the Observer pattern really is—and to emphasize that the terms ‘subject’ and ‘observer’ are really roles in the observer pattern, not just class implementations to be copied from a pattern book.

2.2 Abstractions having Several Roles

This latter point is even more emphasized when we look at the individual components in more detail. In figure 3 is shown the structure of the JHotDraw model component with design pattern roles superimposed.

The Figure abstraction is actually playing a role in four different patterns: observer, composite, decorator, and adapter. Here JHotDraw display another unique, pedagogical, feature, namely that it brings “visual” life to these patterns. If you look at figure 4 you will see an ordinary TextFigure but below that is a deco-
2.3 Other Patterns Used

The adapter pattern is used in JHotDraw through the use of handles—the rectangular boxes that appear when a figure is selected. For instance, a text figure has, when selected, a special handle that allows the font size to be changed simply by dragging it (not shown). Thus, mouse move events are converted into invocations of the text figure's setFontSize method which is an example of the adapter pattern.

JHotDraw uses the well known idea of graphical tools to define the operations to perform on the drawing surface. The tool palette is the set of buttons on the left hand side of figure 4. You select a tool to instantiate a rectangle, a text, or some other figure; and other tools to connect figures, add a border, etc.

The tools act as the controller component of MVC, but looking closer other patterns appear. The selection tool internally uses a state pattern to define the actual operation to perform based upon what is clicked (a figure (move) or its handles (resize)).

2.4 Framework Hotspots

An important question that we pose to the students is: “Why have the developers introduced all these complex interactions?” If you study the event sequence for a simple mouse drag you find that the request is delegated through three or four objects before a figure is actually manipulated.

The answer is that each delegated method call presents a well defined spot to intervene: a hotspot. That is, framework programmers can define new types of tools and/or handles that customize the framework to his or her particular needs, and ask the framework to delegate to/through these custom-made objects instead.

This is of course the key point in a framework, but it also demonstrates the fundamental delegation pattern at work [4].

3. PEDAGOGICAL POINTS

We have outlined a number of important aspects of design patterns that are easily missed by students unless they are demonstrated in a context: that the a single class may participate in many different patterns or even plays different roles within the same pattern; that patterns are often combined; and that framework hotspots may be defined by delegation-based patterns. We have used JHotDraw as example as it is well-engineered, flexible, and uses patterns strictly “by the book”. Other well-engineered frameworks may serve just as well to convey these pedagogical points, however JHotDraw has the advantage of being visual and therefore some patterns, like composite and observer, can directly be observed “in action”.

Our experience with the approach is positive. It is clear at the exams that the students have understood the role aspect of patterns. Students also like JHotDraw for its flexibility and visual
appeal. However, the framework is not very well documented and its complexity introduces a steep learning curve. For short courses it would be interesting to identify smaller frameworks with a less steep learning curve that still illustrate the pedagogical points identified.

Frameworks have a number of other characteristics that make them appealing in a teaching context, and we therefore would like to make teachers aware of:

1. **Student motivation.** A framework defines the skeleton of an application that can be customized by an application developer. This changes the focus radically. If students must program everything from scratch, then the workload and complexity simply rule out making programs that in any respect compares to the fancy and appealing programs that they are used to from e.g. the Windows platform. Prime numbers printed in a shell are not that spectacular. However, a framework provided by the teacher can provide the “bells and whistles” that makes the effort invested by the student look more appealing or “professional”. Talking business language, the “return on investment” is simply greater for the student. In JHotDraw it is possible to make relatively simple exercises that are quite impressive.

2. **Developing is a reuse business.** We find it is important to teach students that “software development” is not just a question of producing code but more a question of finding reusable assets. We must train students to stop trying to reinvent the wheel, and make them comfortable with the idea of reusing high quality software.

3. **First introduction.** Frameworks have already made an impact on how industrial software is developed in a cost efficient and reliable way. Large frameworks like Java Swing, Remote-Method-Invocation (RMI), and Enterprise Beans serve as backbone in large systems. We there find it is important that we teach students the basic concepts and techniques relating to frameworks.

4. **Object concepts.** Good object-oriented frameworks are unique examples of just how strong a paradigm object-orientation is. Looking behind the scenes of good frameworks shows how careful modelling of domain concepts and the use of interfaces, polymorphism, and delegation makes a piece of software highly flexible and demonstrates the power of low coupling and high cohesion. It is simply a brilliant case study to learn from.

5. **Complexity.** Kristen Nygaard, one of the fathers of object-orientation, often emphasized that object-orientation’s main force is in its ability to tackle complex problems—thus we must provide students with problems of some complexity in order to motivate object-orientation.

6. **Backbone for compulsory project.** Our course includes a large programming assignment that is part of the exam. For two years the domain has been designing and programming a backgammon game. JHotDraw is used in one of the last deliveries where students are requested to equip their domain model of backgammon with a graphical user interface using JHotDraw. Thus it serves both as a case study as well as a vehicle for making the graphical user interface much faster than would otherwise be possible.

4. **CONCLUSION**

We have in this paper argued that teaching design patterns presents some pitfalls that may hinder students’ seeing their full potential. Our point is that patterns are descriptions of roles of collaborating objects—this may not become very obvious to students if we do no demonstrate this point in practice.

We have described how we have found the introduction of the high-quality 2D graphical editor construction framework JHotDraw a good case study for making this point clear to students. JHotDraw relies on several of the patterns that are obvious choices in teaching: observer, composite, adapter, model-view-controller. It uses several of them, however, in elegant ways that reinforces the central point of patterns as descriptions of roles and interaction patterns, not as classes.

We have also outlined a set of pedagogical reasons why frameworks are an important topic to teach in its own right. Frameworks are examples of reusing design as well as code—whereas design patterns are only design reuse. Thus it serves to strengthen the view on programming as a process of reusing as well as coding, not just coding.

Frameworks have a lot of benefits that have already made a major impact on how industrial software is developed as well as the cost-efficiency and reliability of industrial software. We have yet to see the same impact in teaching. We hope that our idea of using frameworks as a way to put patterns into perspective will make more teachers adopt them as important topics.

**Acknowledgements**

The figures used in this paper are orginally from a presentation by Mira Mezini.

5. **REFERENCES**