The Need for Killer Examples for Object-Oriented Frameworks

Michael E. Caspersen and Henrik Bærbak Christensen
Department of Computer Science, University of Aarhus
DK–8200 Aarhus N
Denmark
{mec, hbc}@daimi.au.dk

Abstract

In this paper, we argue in favor of introducing object-oriented frameworks as an important topic in our software engineering teaching. Frameworks provide a basis for students to build interesting and impressive programs even with small programming effort at the introductory level. Frameworks are excellent examples of the use of design patterns and software engineering principles to serve as case study at the advanced level. And frameworks convey the important pedagogical point that developing software means reusing software just as much as producing code. However, counting the number of publications and literature about teaching frameworks, it seems that few has realized the potential—it is an underutilized technology. We therefore suggest that killer examples are highly needed and present some experiences with two frameworks that we have found useful in our teaching.

1 Introduction

The Call for Papers state that *Object orientation is an excellent approach to managing the complexity of large, real-world, software systems.* We agree, but add that there is an even better approach: *Let someone else do the job* ☺ .

Software reuse, after decades of unfulfilled promises, is beginning to become true in the form of *object-oriented frameworks*. Industrial developers can build large, complex, software systems that are reliable and computational efficient because they do not build from scratch: they reuse the vast effort invested into for instance the Java 2 Enterprise Edition, Java Swing, or Java Remote Method Invocation (RMI) framework.

Thus, being a successful developer today is no longer just a question of being a good programmer, but just as much a question of understanding complex interaction patterns in third party frameworks and being able to design in accordance with its guidelines.

This changes the skill set that we need to teach students—or rather—we need to teach new skills in addition to the old ones.

This leads us back to the opening statement: *Let someone else do the job*. We strongly feel that a developer/student should tackle a new problem/assignment first by asking: “What software can I reuse to solve this problem?” rather than “What algorithm should I program?” or “Which design patterns should I use?”

We therefore propose that the scope of the “Killer Examples” workshop is broadened to cover killer objects-first-, design patterns-, *and* framework examples. The reason is that articles and literature that treat the topic of how to teach framework concepts, techniques, tools, and examples are extremely slim. We take this as a strong indication that teachers have not yet seen this as an important topic to teach—it is simply an *under-utilized technology* in teaching. This is really a shame as it is a topic that deserves our attention. Thus, we need to get the message across to teachers and one way is to provide a base of good examples of simple, yet powerful, frameworks as well as teaching material that focus on the conceptual foundation of object-oriented frameworks.

2 Why Object-Oriented Frameworks?

We have above stated that reuse is important to teach. Why, then, in the form of object-oriented frameworks? And—what benefits does teaching frameworks have for the students, for the teacher, and for object-orientation? We have found quite a few:

1. *Student motivation.* A framework defines the *skeleton of an application that can be customized by an application developer* [2]. This changes 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 is 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.

2. *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 modeling of domain concepts, use of polymorphism, and the use of design patterns 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.

3. *Developing is a reuse business.* As mentioned in the introduction, it is important to teach students that “software development” is not just a question of producing code. We must train students to stop trying to reinvent the wheel, and make them comfortable with the idea of reusing high quality software.

Thus frameworks can serve in teaching in several areas. First, at an introductory level, it serves as a black box that makes even a small student effort into an rather impressive program. Second, the black box can be opened to show how good programs are structured.

Designing frameworks is fairly hard, and this is probably part of the reason for the under-utilization of the technology. However, more and more software components with framework characteristics see the light of day, and it is therefore important that students as soon as possible are exposed to the technology.

Some argue that framework technology is too advanced a technology to be incorporated in CS1. We do not agree. We have taught a course, "Introduction to Object-Oriented Programming", ten times over the last five years, and our experience tells us the opposite: it is possible and very fruitful to teach frameworks in CS1. As a result of our approach the students gain a clear conceptual understanding of the notion of frameworks (design, inversion of control and hotspots) and most importantly that they have fun working with it – primarily because it is possible with a minor programming effort to produce appealing and impressive applications/applets.

In the next two sections, we will present our experience with teaching frameworks in two different contexts. For our CS1 teaching, we have developed a very simple framework, *Presenter*, that we present as a potential "killer example" of a simple yet rich framework. For our CS2 teaching, we have used JHotDraw as illustration of sound design principles and the clever use and combination of design patterns.

3 The Presenter Framework: A CS 1 Killer Example

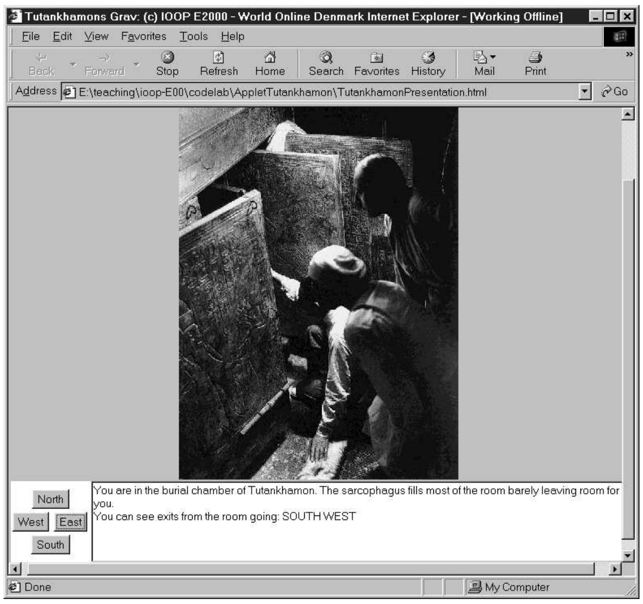
When deciding upon covering frameworks in our CS1 course a number of requirements was formulated: It should illustrate the basic principles of frameworks (inversion of control and hotspots); it should be simple for students to use; it should be flexible in the sense that a number of sensible instantiations should be possible; it should be fun, challenging, and visual.

The (killer) example we have constructed and used is a *presenter framework*. The presenter framework facilitates construction of multi-media presentations of a domain where the compass-directions are a suitable metaphor for user navigation. (So far "multi-media" is limited to images and text but it is possible to extend it to movies and sound.)

We introduce the presenter framework about two thirds way into our semester long course and do it through a specific instantiation, namely a multi-media presentation of the tomb of Tutankhamen, the pharaoh whose tomb was miraculously found rather intact in 1922 by Howard Carter [1].

In fig. 1 is shown a screen snapshot of the Tutankhamen tomb presentation. Using the four buttons marked with the compass directions the user can navigate around the chambers of the tomb. In each chamber the user is presented with a picture taken during the original opening of the tomb along with some explanatory text. It is our experience that the concrete instantiation—moving around a tomb with pictures from the original opening—grabs the imagination of the students.

The Tutankhamen's tomb instantiation also allows us to underline an important software engineering principle, namely separating model/domain code and user interaction code. We build a small object-oriented model of the domain with classes: *chamber* (having exits, an image and a description) and *visitor* (having an association with a specific room and a `move` method). As the user interaction code is completely


Tutankhamons Grav: (c) IOOP E2000 - World Online Denmark Internet Explorer - [Working Offline]
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print
Address E:\teaching\ioop-E00\codelab\AppletTutankhamon\TutankhamonPresentation.html
Go
North
West
East
South
Done
My Computer

You are in the burial chamber of Tutankhamon. The sarcophagus fills most of the room barely leaving room for you.
You can see exits from the room going: SOUTH WEST

Figure 1: The presenter framework instantiated to present Tutankhamen's tomb.

defined by the presenter framework, it is simply impossible for the students to mix UI and model code except through the well-defined hotspots provided by the framework.

3.1 Design

The presenter framework provides the application programmer with a simple interface[1]:

```
public abstract class Presenter
{
    public void showImage(String filename) { ... }
    public void showText(String text) {...}

    public abstract void northButtonPressed();
    public abstract void eastButtonPressed();
    public abstract void southButtonPressed();
    public abstract void westButtonPressed();
}
```

Presenter provides the backbone functionality: a large area for displaying images, a smaller one for displaying text, and four buttons labelled with the direction buttons. The buttons respond to user clicks and invokes the four abstract methods—these form the hot-spots that students must override to provide tailored functionality.

[1]A simplified version is provided in this paper for the sake of clarity. A more detailed treatment can be found in (will be filled in if accepted).

Thus, to instantiate the tomb presentation is a matter of overriding the `..ButtonPressed()` methods as e.g. in:

```
public void northButtonPressed() {
    visitor.move(NORTH);
}
```

where the move method of the visitor object must test an exit leading north and invoke the `showImage` and `showText` methods with appropriate parameters.

Though very simple, the students must cope with two new concepts, that are fundamental for frameworks: *inversion of control* and *hot-spots*.

3.2 Inversion of control

In the students' previous programming experience from example code and exercises, there are always a number of interacting objects and a single 'driver' that does the setup and defines the main control flow. Now the control flow is dictated and controlled by the presenter framework instead. The application code comes into play only when the overridden `...ButtonPressed()` methods are called. This is a simple variant of event-driven programming and illustrates the inversion of control principle.

3.3 Hotspots

Frameworks define core functionality, control flow and object collaboration patterns. Application programmers refine frameworks to specific domains by adding code at well-defined points: the hotspots (also called hooks or variability points). Hotspots can be defined using a number of different techniques: callback methods, delegating to objects that implement interfaces defined by the framework, subclassing, etc. We have adopted the subclassing technique as we find it the simplest and as it also demonstrates yet another use of polymorphism and specialization.

3.4 Discussion

Several interesting, yet simple, instantiations can be made from the Presenter framework.

The first exercise is to make a virtual tour of a museum or gallery; a layout of a number of locations in a gallery is defined and a painting is associated with each location. The buttons can be used to move around the gallery and see the various paintings. This exercise is deliberately similar to the tomb instantiation. In another exercise only the "north" and "south" buttons are used to run through a list of images, essentially making the presenter a slide-show application.

The basic directional navigation metaphor also lends itself naturally to "classic" adventure games. We have an extension of the framework with the ability to show and move items between the visited room and the visitor's inventory.

After having introduced the students to Java's graphical toolkit we have an exercise to make a 4x4 slide-puzzle by defining a grid of buttons marked with the numbers 1–15 and an empty button denoting the “hole”[2]. The “hole” is then moved by pressing the compass buttons so the user can try and solve the puzzle by arranging the numbers in the right pattern in the grid.

In summary we find that though the provided functionality of the framework is limited and simple, there are a number of intriguing exercises to be made based upon the framework that forces the students to negotiate the basic principles of inversion of control and refining hotspots.

4 JHotDraw: A CS2 Killer Example

We also teach a course in “Programming in the Large” with emphasis on concepts, techniques, and tools to develop large-scale object-oriented programs. In this course, we go into detail with design patterns and frameworks.

We have adopted JHotDraw [3] as an example of a relatively large, complex, and very powerful framework. JHotDraw was originally developed by Thomas Eggenschwiler and Erich Gamma but is now part of the open source community. JHotDraw is a framework for developing 2-D semantic drawing editors.

JHotDraw serves a number of purposes in the course:

- *Backbone for compulsory project.* The 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 here serves the same purpose as in our introductory course: to make the result of the students’ effort look impressive.

- *Design patterns in action.* JHotDraw consists of more than 180 interfaces and classes. This makes it completely incomprehensible unless some “road map” is given. And—design patterns provide this road map. The central architecture of the framework can be explained in terms of three patterns; and sub-architectures yet again as combinations of patterns.

  Another important point: If not careful, teaching design patterns easily becomes just as boring and numbing as lecturing over a cook-book: “Today we will talk about “observer”, tomorrow about “strategy”, and next week about “state”. This easily makes the students miss the real point: that a given class in the UML class diagrams showing a particular pattern is *not* a class—it is a *role* that some class must play in the resulting design. In JHotDraw the central abstractions each play a role in three or four design patterns and the role aspect is evident.

- *Understanding collaboration patterns.* Adapting JHotDraw to a particular need, like moving graphical checkers on a backgammon board, demand that the students understand and follow the interaction patterns defined by the framework

[2] A general version of the Presenter framework is used that can display any Java graphical component, not just images.

architecture. If not, they end up on big trouble where they “fight against” the framework instead of letting it care about the low level interaction details. Thus it focuses and reinforces the students’ attention to the dynamic aspects of patterns which is the main point of the behavioral patterns.

5 Conclusion

We have in this paper argued that object-oriented frameworks is an important topic to teach. Frameworks are examples of reusing design as well as code—whereas design patterns is 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 has already made a major impact on how industrial software is developed as well as the cost-efficiency and reliability of industrial software.

We still lack to see the same impact in teaching—it is definitely an “under-utilized technology” in our opinion. We have argued that frameworks indeed have the potential to have a large impact in the form of more interesting, “impressing”, and thus motivating exercises for the students and as a vehicle for teaching and understanding both basic as well as advanced principles in object-oriented programming and design patterns. We have presented two examples of frameworks that we have found profitable to use in our own teaching.

We hope that our participation in the Killer Examples workshop will provide us with feedback on our ideas as well as make framework teaching appealing to a wider audience of teachers.

References

[1] H. Carter, A. Mace, and J. M. White. *The Discovery of the Tomb of Tutankhamen*. Dover Publications, 1985.

[2] M. Fayad, D. Schmidt, and R. Johnson. Building application frameworks. In *Building Application Frameworks*, chapter 1. Wiley and Sons (?), 2000 (?).

[3] Jhotdraw as open-source project. http://www.jhotdraw.org/.