Experiences with Teaching Design Patterns

Sylvia Stuurman
Open University
Heerlen
the Netherlands
+31 45 5762177
Sylvia.Stuurman@ou.nl

Gert Florijn
SERC
Utrecht
the Netherlands
+31 30 2308966
gflorijn@serc.nl

ABSTRACT

In this paper, we present an assignment for a course on Design patterns at the masters level, where students have to adapt an existing program to meet additional requirements. We describe the basic program, discuss the reasons why we decided for such an assignment, and show the results.

The assignment proved to be very effective both to train students to work with design patterns and to assess that students have reached the learning goals. This was true both for students with a professional background and for students with academic interests.

Categories and Subject Descriptors

K.3.1 [**Computers and education**]: Computer uses in education – *Distance learning.* K.3.2 [**Computers and education**]: Computer and information science education – *Computer science education.* D.3.3 [**Programming languages**]: Language constructs and features – *Patterns.* D.2.7 [**Software engineering**]: Distribution, Maintenance, and Enhancement – *Restructuring, reverse engineering, and reengineering.* D.1.5 [**Programming techniques**]: Object-oriented programming.

General Terms: Design, Experimentation.

Keywords: Design patterns, distance learning

1. INTRODUCTION

The courses of the Open University in the Netherlands are meant to be studied at home, without extensive help from teachers. Especially during the courses at the masters level, we expect students to be able to study independently.

One such a masters-level course focuses on Design patterns. The average student should be able to finish this course within 100 hours. Students are supposed to have a broad knowledge of and experience with Java, to have experience with object-oriented design using UML, and to be able to work with an IDE such as JBuilder and a UML tool such as Together.

Our goal with this course is twofold: on the one hand, we offer a course for students who follow the masters program of Technical Informatics. On the other hand, Open University students often have a job in the field of their study, so their interest is professional as well in many cases. Therefore, the course should be suitable for software designers who want to gain insight and become better designers: the course should offer professionals usable knowledge and experience.

The course is based on the book Design patterns explained [8]. Students read this textbook using a accompanying workbook [9] with exercises, background information and explanations. Furthermore, students get the "classic" book on Design patterns of Gamma et al [5] on cd.

In designing the course, we used several principles and had to meet several challenges.

First, while designing the course and writing the workbook, we held the principle that design patterns are not only learned by reading about them and drawing class diagrams, but by implementing them as well, as is argued in for instance [2].

Our claim is that this principle of learning by using design patterns helps the academic student to gain insight in the how and why of design patterns, and helps the professional to learn to use them instead of only knowing that they exist. We implemented this principle by providing design- and implementation exercises throughout the workbook.

In [6], Goldfedder and Rising observed that exposure to a variety of systems was more critical for being able to learn to use design patterns than the number of years of experience. Therefore, we tried to pick the examples and exercises from varying domains.

However, this is not enough for both academic students and professionals. Apart from learning individual patterns and the principle behind them, they should learn how to understand and apply patterns they have not seen before, how to integrate different patterns, and how to use this knowledge in real-life situations.

Furthermore, design patterns are not learned by reading and doing small exercises. In small exercises, the task of discerning which problems are present and which patterns could help to solve these problems is a trivial one because of the size of the exercise. Students should train such a competence by working with a fairly large computer program. The same applies to the ability to combine patterns: this is a trivial task in small exercises, and can only be trained truly using a larger program.

Another problem with designing the right exercises stems from the fact that teaching design patterns means teaching advanced object-oriented design. Design can only be learned by thinking about alternatives, and the advantages and disadvantages of these alternatives. This is a difficult task for a student studying at home: a classroom where one can discuss matters with teachers or other students is a much easier environment to train this competence.

In the following section, we describe the general idea of the assignment that we use, which forms a solution for the problems sketched above. In section 3, we describe some details of the program that we use as a base for the assignment. Details about the procedure of the assignment are given in section 4. In section 5, we summarize the observations we made. Section 6 contains some remarks about related work, while we give conclusions in section 7 .

2. THE ASSIGNMENTS

The key decision for our course was that the final assignment should consist of a design, an implementation and a report documenting the solution and especially the rationale as to why and how certain patterns were used. In this assignment, students have to change an existing program to meet new requirements, using design patterns. This means that students can work on a larger program than when they would have to work from scratch. Several sets of new requirements, called scenario's, were defined. The students do their assignments at home, like the rest of their study, but in order to force students to think and talk about different solutions, we require them to work in teams of two students.

We don't let the students plunge into the deep at once: before they start at their final assignment they have worked out two design assignments, both bigger than the exercises in the workbook. In the first assignment, they are told to try to use a fixed set of design patterns; in the second assignment, they have to decide for themselves which patterns are useful and why. The second assignment is done in teams; they tackle the scenario for the final assignment in the same teams.

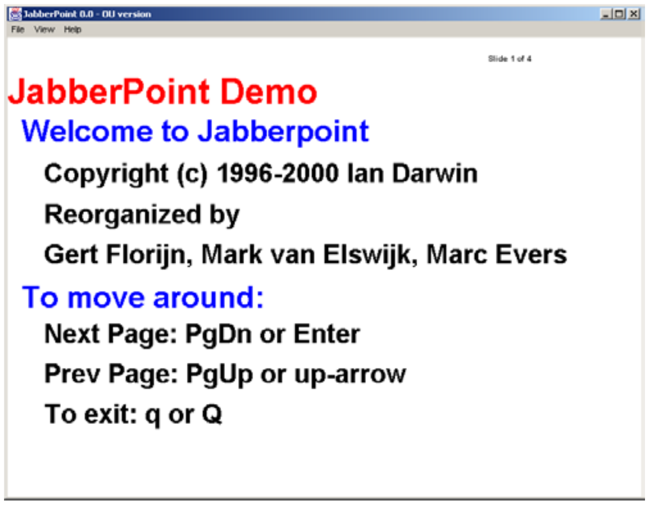
We kept to the principle of gradual exposure to complexity, as described in [3]: students start by doing small exercises from the workbook, thus working at the level of knowledge, comprehension and application. After having finished the first half of the text- and workbook, they do a design exercise using a limited set of patterns, at the level of application and analysis. The next big assignment is again a design problem, this time without a fixed set of patterns, at the level of analysis and synthesis. Finally, the students do their final assignment at the level of synthesis and evaluation, and showing that they are able to find and use patterns not treated during the course.

3. THE PROGRAM: JABBERPOINT

The starting point for the final assignment is an existing program, inspired by a program written by Ian Darwin: Jabberpoint [4]. Jabberpoint is a slides presentation program written in Java.

When we started thinking about a suitable application for the course, we had a few goals. Ideally, we wanted to use an existing, "real-world" application instead of a toy program. Furthermore, it would have to be an application that students recognize easily, without having to spend time learning domain details. Likewise, the application should make it easy to think of functional enhancements that could be implemented using patterns. Finally, the program should be small enough to understand it in a couple of hours and should be written in Java.

These requirements led us to the idea of using a presentation program for the course. A search on the Internet brought us to Jabberpoint. Jabberpoint was created by Ian Darwin, partially as a case study for the Java courses he gave. Basically, the program reads an input file (typically an XML file) containing the data about a presentation, and then displays the presentation in a window with keyboard and menu controls to nagivate the presentation.



JabberPoint Demo

Welcome to Jabberpoint

Copyright (c) 1996-2000 Ian Darwin

Reorganized by

Gert Florijn, Mark van Elswijk, Marc Evers

To move around:

Next Page: PgDn or Enter

Prev Page: PgUp or up-arrow

To exit: q or Q

Figure 1. Jabberpoint showing a slide

The core concept in Jabberpoint is a presentation. A presentation consists of a sequence of slides, each of which is built up of slideItems. Various types of slideitems were already available, ranging from text to images. Within each slide, the items have a level associated with them, indicating the indentation and (font) style properties that should be used in displaying the items.

The display of a presentation is handled in a user interface class that extends javax.swing.JComponent. It links to the presentation and keeps track of the current slide in the presentation using the Observer pattern. It retrieves items from the current slide, determines the position for each of them and delegates the actual drawing of the items to the objects of the concrete slide-item classes. The user-interface is setup fairly implicitly, using a few controllers (for keyboard and menu) to navigate across the presentation and load another one.

For the loading and saving of presentations, Jabberpoint contains several accessors. Formats supported include XML, HTML and plain text.

Jabberpoint met most of the goals we defined at the outset. It's both small and big enough, it's written in Java and it provides basic functionality that most of the students will understand. Furthermore, even occasional usage of a presentation program suggests several functional enhancements.

In order to make Jabberpoint usable as the basis for our assignment, several changes were needed. While considering exercises and (pattern-based) solutions for them, we refactored and simplified the code in some areas. For instance, some classes were renamed (such as Presentation instead of Model, Slide instead of M), interfaces were introduced, the accessors were simplified, and the Observer pattern that was already contained in the original code, was written out of it. The task of bringing the Observer pattern back into the program was turned into an introductory exercise for students as a way to get to know the ins and outs of the program.

4. THE SCENARIO'S

Teams each receive a scenario. Examples of these scenarios are:

- The program should get the possibility to save a presentation in HTML.
- The program should be able to show two extra views on the slides, such as a slidesorter, slides with notes, an outline, or a next slide previewer (a small window showing the next slide).
- It should be possible to have hyperlinks on slides, combined with one or more actions, such as playing a sound, go to the next or previous slide, go to slide number x, open a new presentation, etc.
- While showing a slide, it must be possible to draw on a slide. These drawings should persist while the presentation is held, but should not be permanently saved.
- Instead of showing all elements of a slide at once, it should be possible to show only the elements of a certain level, or to show them one at a time.
- It should be possible to define more than one presentation using one set of slides. The order of the slides may change, slides may be left out, or may be used more than once.

It's easy to think of more scenarios, so it is possible to have students work at a fresh set of scenarios in the future.

The procedure for the assignment is that teams first develop a design and send it to their teacher, who comments on the design. Then the teams make their final design, implement it and make up a report, explaining what they changed, which design patterns they used, which alternatives they considered and why they decided to choose for the solution they present.

Teams are asked to describe which work was done by each of the students, and they have to agree on that description. The first 18 students held a presentation about their solution, but we dropped that requirement because of the amount of time it costs the students.

5. OBSERVATIONS

At this moment, 38 students have finished the course. The first 18 students who took the course, were asked to fill in evaluation forms. We also had a meeting with these students, where the teams held a presentation about their solution, and where they told us what they liked and disliked about the course.

5.1 Time spent

The first observation concerns the amount of time students needed. The idea was to use 28 of the 100 hours for the final assignment. One of the 18 students needed fewer hours (12); the rest needed far more than those 28 hours, with a maximum of 75 hours. The average time needed was 38 hours.

When asked about it, the common answer was that they deliberately spent more time than needed because they became hooked. They tried (and implemented) several solutions to compare, just for the fun of it. The general opinion was that the assignment could be done in about 28 hours, under the conditions that the program would be introduced earlier in the course and students would already be familiar with it, and that the presentation would be dropped.

5.2 Working in a team

A second observation is that students preferred working in a team for this course, as opposed to working individually. This is a remarkable fact, because Open University students in general dislike working in teams (or at least say they do), because of practical problems.

Two students worked individually for different reasons; all other students worked in a team. Students discussed several solutions with each other, and commented on having learned from having to give arguments, and being confronted with the ideas of someone else.

One of the students lived in California, one in Germany, eight in Belgium, and the rest of them in the Netherlands. Obstacles for coöperation were time-related (one member of the team studying during weekend, the other one at workdays; one member of the team studying at night; the other one during the day), and not related to the impossibility of personal contact.

The workload was equally divided in all teams, according to the students. In some cases, both students worked out a design, discussed it and chose a final one, then divided the classes to implement and the parts of the report to write; in some cases the work was divided by having one student make the design, having the other comment it, and working the other way around for the implementation.

5.3 Learning design patterns

All students showed that they had understood the meaning of using design patterns, that they could spot problems where a pattern might come in handy, that they could search for patterns that could provide a solution, and that they could argue why the solution they chose was the most flexible. All students were able to explain which future changes would be easy because of the solution they had decided upon. In other words: all students showed they had mastered the course.

In their comments, many students told us that the three assignments, and especially the final one, had helped them to grasp the concept of a certain design pattern: being confronted with a real problem was what they needed to "see the light". Many of the professionals told us they had begun to make use of their knowledge of design patterns in their work during the course: the relation to their work was made very easily.

5.4 Academic students and professionals

About 15 of the 38 students studied the course for their masters degree; the rest of the students had only a professional interest.

There was no difference between those two groups with respect to the degree of their appreciation of the course. Professionals commented on the usefulness of what they learned in their work; academic students commented on the thoroughness of the material and the depth of their understanding.

5.5 Support

We did not offer special support for working within a team, so students relied on e-mail and telephone to communicate. Providing a versioning server (e.g. CVS) for cooperation was considered, but declined because of the troubles getting clients to work on different Windows-versions, and because of the extra learning curve. The only support we gave students was a proposal on how to divide the work, and tips on how to find a suitable teammate.

This support proved to be sufficient: working together has worked out very well for all teams.

Another form of support consists of a website for the course, where we have collected links to more information on the web about the topics of the text- and workbook. The website was used by students during the course, but they almost exclusively used the cd containing [5] to look for useable patterns for the final assignment.

Students did not use the discussion group that we provide for the course to discuss problems with their assignments with other students. They did use the discussion group during the rest of the course, so the explanation is probably that they believe in an implicit rule that assignments should not be discussed. We will consider the consequences of telling students explicitly that assignments may be discussed in this discussion group.

Teams were also supported by allowing them to send in a draft design to the examinator, who commented on it. We chose for this procedure to help students avoid spending much time in dead alleys. This form of support was highly valued, especially the fact that the comments were given within a day or two after sending in the draft.

6. RELATED WORK

Design patterns are often used in introductory courses. In [10] for instance, Wallingford shows how to use both procedural and object-oriented design patterns in an introductory course. In [1], material for introductory courses in design patterns is presented. In [7], a set of patterns is described for guiding students through the topics of an introductory computer science course.

A course at the master's level imposes different requirements, especially when not only academic students, but also professionals are addressed. We haven't been able to find other courses on design patterns at this level.

7. CONCLUSIONS

Design patterns represent a subject where the academic and the professional world almost meet by definition: patterns are a form of distilled professional knowledge, and by studying them at an academic level, the mechanisms of object-oriented design principles become clear. Design patterns have been "discovered" in the academic world, and are based on thorough professional experience. Teaching design patterns at the master's level, in a way that the interests of both professionals and academic students are met, requires hands-on experience on a project of sufficient size and complexity.

Our approach of giving change scenario's for a given program has proven to be a solution for the problem of having students work on a fairly big program to be able to learn from practical experience, but at the same time making it possible to finish the course within a limited amount of time. Students were able to finish the course within 100 hours (after some adaptations to get the assignment fitting within 28 hours), and they showed to have learned what the course teaches, for instance by applying patterns not taught within the course.

The cooperation within teams had the form of contact by e-mail and telephone, without personal contact, in all cases. The limited support for cooperation proved to be sufficient for all students.

Some considerations when using this approach are:

- Working out a scenario often both consists of refactoring the original program (restructuring without changing the functionality), and adding functionality. In principle, it would be better to separate these two activities. We are afraid that the assignment would cost students more than 28 hours if we would ask them to explicitly separate these activities, but it would be worthwhile to experience with it.
- Students would experience the benefits of design patterns even more than in the current assignment if they would have to integrate two solutions for two different scenarios. That would show how patterns really enhance flexibility. We cannot take this approach because of the time constraint, but we are considering these type of assignments in courses building on the knowledge learned in this course.
- The two students working alone sent in poorer results than all students working in a team. Of course, the numbers are not high enough to make statistically sound conclusions, but the prediction that working in a team, and therefore being forced to discuss solutions with each other benefits the learning process at least seams to hold. It is strongly recommended to have students work in teams.

8. ACKNOWLEDGMENTS

We would like to thank Frank Wester of the Open University of the Netherlands for being an excellent and stimulating team leader for this course. And we would like to thank Johan Jeuring and Herman Koppelman, both of the Open University of the Netherlands, for their comments on this paper.

9. REFERENCES

[1] Astrachan, O., Berry, G., Cox, L. and Mitchener, G..Design patterns: an essential component of CS curricula, Proceedings of the 29th SIGCSE technical symposium on Computer science education, 2000.

[2] Beck, K., Crocker, R., Meszaros, G., Vlissides, J, Coplien, J.O., Dominick. L. and Paulisch, F. Industrial experience with design patterns, Proceedings of the 18th international conference on Software engineering, 1996.

[3] Buck, D. and Stucki, D.J. Design early considered harmful: graduated exposure to complexity and structure based on levels of cognitive development, Proceedings of the 31rst SIGCSE technical symposium on Computer science education, 2000.

[4] Darwin, I. Jabberpoint, sourcecode in Java, available online: http://www.darwinsys.com/, 1996.

[5] Gamma, E., Helm, R., Johnson, R. and Vlissides, J. Design patterns elements of reusable object-oriented software, Addison-Wesley, 1994.

[6] Goldfedder, B., and Rising, R. A training experience with patterns, Communications of the ACM, vol 39 (10), 60-64, 1996.

[7] Proulx, V.K. Programming patterns and design patterns in the introductory computer science course, Proceedings of the 31st SIGCSE technical symposium on Computer science education, 2000.

[8] Shalloway, A., and Trott, J.R. Design patterns explained, a new perspective on object-oriented design, Addison-Wesley, 2001.

[9] Stuurman, S., Wester, F.J., and Witsiers-Voglet, M. Design patterns, Open Universiteit Nederland, 2002.

[10] Wallingford, M. Towards a first course based on object-oriented patterns, Proceedings of the 27th SIGCSE technical symposium on Computer science education, 27-31, 1996.