Introducing Objects to Novices in Six Days
A use case centered approach for teaching object-oriented development

Till Schümmer† and Daniel Tietze‡

GMD – German National Research Institute for Information Technology
IPSI - Integrated Publication and Information Systems Institute
Dolivostraße 15, D-64293 Darmstadt
† also Darmstadt University of Technology – ‡ also GroupTech GmbH
{schuemme | tietze}@darmstadt.gmd.de

Abstract. This paper outlines a six-day course on object-oriented analysis, design and implementation. The course was designed as part of a retraining course, which lasted seven weeks. Within the course, we emphasized the importance of analysis. The participants analyzed a hotel management system on the basis of a short movie outlining the job of a hotel employee. Real world use cases were derived from this movie. In the following steps, the participants used a case tool to perform design and implementation. Our experiences show that it is possible to get a good understanding on analysis and design after three days. The success for the implementation phase was satisfying.

1 Introduction

German industry is currently spending big efforts on recruiting new employees for their software development departments. Still, the demand for qualified IT professionals is growing faster than university or schools can follow with training. In 2001, an expected number of 5500 computer scientists will finish their studies in Germany. In contrast to this, 75000 are needed by German industry and the number grows by 25000 each year [Dambeck 2001].

For that reason, many companies have started to re-educate their employees in the area of software development. One such approach was made by the httc [httc 2001] in co-operation with the Darmstadt University of Technology to train employees of a large German telecommunication company. Within seven weeks, they should gain an insight into different areas of computer science to see which area is the most interesting for their future work [Feuck 2000]. Six days of these courses were devoted to object-oriented analysis, design, and implementation.

Within this paper, we will first give a brief overview over the curriculum of the whole course, followed by a more detailed description of the part on object orientation. After this, we will present our experiences gained from the last three runs of the course, followed by some concluding remarks.

2 Curriculum

The courses were structured in six building blocks. All modules expected that the participants did not have previous knowledge in the block's area. Participants could have worked in a totally different area before. These areas range from administrative or accounts departments over electrical production up to work in the field.
The first six weeks of the courses contained introductions to operating systems, structured programming with C, computer networks, project management, object orientation, and databases. In the seventh week, the participants had time for one final exercise, which covered all taught areas.
When the participants entered the block on object orientation, they often had only as little programming experience as they could gain in the four days of the structured programming block.

2.1 The Curriculum for object orientation

Topic	Duration (minutes) Theory & Practice
Process models	90
System theory (*)	60 60
Objects and messages	60 60
Use cases	180 240
Classes	60 90
UML	60 90
Transition: Use case to model	60 60
Inheritance	90
Case tools	60 90
Object-oriented languages (*)	90
Java	180 180
Patterns (*)	60
Testing and refactoring (*)	90 90
Additional exercise	60 180
Summarization	90

Table 1: The curriculum for object orientation.

The object-oriented analysis, design, and implementation were taught in six days. Within the first three to four days (depending on the group's performance), we covered the areas of analysis and design. The last two to three days were dedicated to the introduction and use of the programming language Java (this was requested by the customer). The whole structure is shown in table 1. The vertical arrangement of the bars shows the sequence of the topics, the length of the bars indicates the time spent on the topic. Asterisks indicate topics omitted or shortened for time reasons.
Process models. The first topic of the course (for the remaining part of the paper, course refers to the part on object orientation) was an introduction to the object-oriented process. Within this block, the participants were introduced to two currently

relevant process models (the Rational Unified Process [Jacobson, Booch, Rumbaugh 1999] and eXtreme Programming [Beck 1999]). We included this block in order to give the participants an overview over the upcoming topics. Since they were not familiar with the activities taking place in the different steps of the process, they got only a vague impression of the whole process.

System theory. The goal of the system theory block was to make participants sensitive for seeing their environment as a system. This seemed obvious, since OO system development implies the understanding of the system. Several didactical approaches argued for the emphasis on system theory, when teaching object-oriented analysis and design (e.g. [Baumann 1990], or [Foegen 1996]). We skipped this block in the third run of the course for reasons that will be explained in the last chapter.

Objects and messages. We started this block by telling the participants what an object is, illustrated by real objects that were around in the room (pens, books, etc.).

The next step in teaching objects was to figure out what an object is made of. We ended up with three concepts that are important for objects: Identity, state, and behavior. Again, all the concepts were explained using real world examples. After the concepts were clear, we explained encapsulation. Participants should understand that the only way to get any information from objects or to change an object's state was to ask it to do so. This led directly to the concept of messages.

The interactive presentation was followed by a hands-on example: The participants used Squeak Smalltalk [Ingalls et al. 1997] and the included bunny character [Pierce 2001], which moved as reaction to messages sent to it. The only thing they knew was that the bunny's visible components were objects and how to send messages to the objects. They were asked to let the bunny hit his drum, but very soon they discovered a wide range of different movements of the bunny, including dances and saltos.

Use cases. We introduced use case analysis following the book of Cockburn [Cockburn 2000]. The participants were first taught how to create precise stories of real world experiences (their last arrival in a hotel). These stories were then used to create real use cases, find right abstraction levels, and think about pre- and post-conditions.

This systematic procedure for use case analysis was then applied to a larger real world example. We would have liked to have had real world customers in order to start the analysis exercise with real interviews, but this approach was not feasible. Therefore, we used a video tape, which explained the job profile of a hotel employee ([Bundesanstalt 1991]). The video was produced by the German employment office to help middle school students to figure out what job they want to carry out after school. Here, one could see all aspects of hotel administration, in which the hotel employee was involved. Figure 1 shows examples from the video. It shows three scenes, which illustrate three different areas of the hotel management system: task assignment for service staff, stock-keeping, and information about leisure possibilities.

The participants were asked to extract all use cases from the video that were important for a hotel management system. Each scene revealed a different use case appearing in the hotel. After each small group (2-3 participants) came up with a full list of use case titles, the use cases were reviewed and sorted thematically in the plenum. Then, the plenum split again and each small group refined two to four use cases.



Figure 1: Three different scenes of the movie, which was used to perform use case analysis

Classes – UML – Transition: Use case to model. Classes were introduced using the metaphor of a blueprint for similar objects. Again, we took real world objects as examples and let the participants group them into classes. Having found the classes, the participants had to identify attributes and methods. Here, we also introduced the UML notation for class diagrams. We also explored use case diagrams.

The text-based use cases then served as a basis for designing the hotel management system. We introduced some basic syntactic rules to identify classes, attributes, relations and methods from the use cases. The participants applied these rules and created UML diagrams describing their design.

Inheritance. The concept of inheritance was introduced as soon as the participants started to create related classes. As an exercise, they were asked to create an inheritance tree. They were given about 100 different classes (these classes were taken from the index of a mail-order catalogue), which were written down on small cards. The groups could then rearrange the small cards on the table, until they had a satisfying hierarchy. The hierarchies were then presented and discussed in the plenum. An important lesson from this exercise was that there was no *one* correct answer to creating the class hierarchy. Rather, the result depended on the emphasis placed on different qualities of the given classes.

Case tools. We used the case tool Together [TogetherSoft 2000] to let the participants model the hotel management system. After a short introduction into the tool, each small group was asked to draw the class diagrams for the design of their use cases. The CASE tool guided them by ensuring that the design was UML-compliant.

Object-oriented languages. Before we started to teach the participants a specific OO language (in our case Java), we planned to give an overview on the history of object-oriented programming languages. This block was to include some words on each language's specific characteristic and serve as a mean for deciding which OO language fits the problem best. In all three runs, we omitted this block for time reasons.

Java. Implementation was taught using the JAVA language. In this block, we first taught the basic syntax and language constructs of Java. Since the participants had learned C before, they already had some basic knowledge.

After the basic language features were covered, we tackled some parts of the class library (mainly container classes), to show how such a library can be used. Because of the use of the case tool, the load of programming was minimized. The tool produced a skeleton and the participants only had to fill the empty methods.

We decided not to teach any GUI aspects like Swing, since they would have been too complicated for novices. As an example, we built a system for a video store. The design was given and we considered only the activity of implementing the design.

Patterns – Testing and refactoring. The blocks on patterns, testing, and refactoring were intended to be additional blocks. We planned to give the participants additional means for improving their design and their implementation. In all three runs, we omitted these blocks for time reasons.

Additional exercise. In the additional exercise, we asked the participants to finish the implementation of a use case for the hotel management system. As an option, we also offered that the participants could refine the video store system. This exercise combined all aspects of the previous blocks and was therefore well suited for recapturing and improving the learned knowledge.

3 Experiences

In contrast to other approaches that claim to teach object-oriented methods and languages in a short period of time (e.g. [Balzert 2000, Cadenhead 2000]), we put a large emphasis on analysis of object-oriented systems. This was reflected in the results: The participants produced very good use cases, and acceptable design diagrams, but they had major problems to transform the design into a Java program. Even though we used a CASE tool that automated some parts of the implementation work, the participants struggled with the syntax of the programming language.

Overall, the results exceeded all expectations. The goal of the course was to give the participants first insights into the topics of object-oriented analysis, design, and implementation. Already in the first hours, they managed to communicate with objects (in the Squeak example). Almost no syntax was needed here and the way in which one can use objects appeared very natural.

All groups produced very good use cases. Following Cockburn's method helped them to concentrate on the right level of abstraction and detail. The students made a very detailed analysis of the hotel management system, extracting all the facts that were in the video. In the first days, the video served as a very good motivation.

When the participants transformed the use cases to design, they always tended to include fictional parts in the system. But at the end of the design exercises, they began to concentrate on the requirements and fictional parts appeared less often.

The longer we used the example of the hotel management system, the less motivated the participants were. Although they produced good designs, they did not really get to the point where they believed that they could implement the whole system in the remaining three days. Thus, they decided unanimously to implement the video store example.

The participants were disappointed that we did not cover graphical user interfaces. But the disappointment vanished when they started to implement on their own: Learning Java was the biggest problem for most of the participants. They were not used to expressing themselves in a programming language. Thus, simple syntactic constructs meant a large barrier for them. More practice would be needed here, to allow the students to become more familiar with programming languages.

In order to provide more time for practice, we had to omit the blocks on system theory, programming languages, patterns, testing, and refactoring. These blocks were all quite theoretical and did not suit all group members, who were no longer used to taking courses. Nevertheless, we are still confident that these blocks are very important in order to train software engineers. From the students' reactions, though, we learned that too much theory confused them and reduced their level of motivation. Thus, these blocks should be taught after the students have had more experience.

4 Conclusions

In this paper we presented a course, which taught object-oriented techniques in six days to students with no or only little previous knowledge. We observed that it is possible to get a good understanding of analysis and design within the first days. Real-world examples, especially the use of video for learning use case analysis, helped the students to understand the topic of analysis. In the phase of design, we had to carefully make the students concentrate on the results of the previous analysis phase, but when they took this into account, they were able to produce good designs.
Throughout the course, the participants performed best when there was an emphasis on practical experiences. The use of a programming language will need more practice, than it was possible in the restricted time frame of six days. Taking into account that the participants should be trained on the project after the course, this practice could be granted afterwards by experienced team members (if the project teams consisted of developers with a mixed level of experience). We have already received informal feedback that the participants of the first installment of the course are performing well in such an environment.

References

Baumann, R.: "Didaktik der Informatik", Stuttgart: Klett-Schulbuchverlag, 1990.
Beck, K.: "eXtreme Programming Explained"", Reading, MA: Addison Wessley, 1999.
Bundesanstalt für Arbeit: "Filme zur Berufsfindung: Hotelfachmann/frau", movie, available at the Landesbildstelle, Hessen: 1991.
Cockburn, A.: "Writing Effective Use Cases", Boston: Addison-Wesley, 2000.
Dambeck, H.: "Schweinezyklus - Wie viel Informatiker braucht das Land?", in: *c't*, 5/2001, 66.
Feuck, J.: "Technische Uni leistet Hilfe", in: *Frankfurter Rundschau*, 25.08.2000.
Foegen, M.: "Entwurf eines didaktischen Konzepts der Informatik", diploma thesis, Darmstadt University of Technology, 1996.
Ingalls, D., Kaehler, T., Maloney, J., Wallace, S. and Kay, A.: "Back to the future: the story of Squeak, a practical Smalltalk written in itself", *OOPSLA 1997*, Atlanta, 1997, 318-326.
Jacobson, I., Booch, G. and Rumbaugh, J.: "Unified Software Development Process", Addison-Wesley, 1999.
Pierce, J.: "Alice in a Squeak Wonderland", in Guzdial, M. and Rose, K., Ed., *Squeak, Open Personal Computing for Multimedia*, Prentice-Hall: 2001 (to be published).
TogetherSoft: Company Homepage, http://www.togethersoft.com/, 2000.
httc e. V.: Company Homepage, http://www.httc.de/, 2001.