Teaching OOT Using a Framework and Both Direct and Net-based Tutoring[1]

Birgit Demuth, Heinrich Hussmann, Steffen Zschaler
Department of Computer Science
Dresden University of Technology
01062 Dresden, Germany
{demuth, hussmann, zschaler}@inf.tu-dresden.de

Lothar Schmitz
Department of Computer Science
University of the Federal Armed Forces Munich
85577 Neubiberg, Germany
lothar@informatik.unibw-muenchen.de

Abstract: We report on experience from teaching OO project courses to undergraduate students. Before they can successfully tackle projects they have to climb a rather steep qualification ladder: pick up a working knowledge of some OO language, learn and practice OOA and OOD, and get used to advanced ideas like patterns and frameworks. In order to relieve this heavy burden somewhat, we provide an object-oriented application framework as a common base for the projects. That way, the students are given an architecture, which they have to adapt to their specific task instead of doing all the design on their own. We also believe that this policy closely resembles the way beginners are integrated into on-going projects in practice. We briefly describe the Java framework we use and then concentrate on the organizational issues involved.

Introduction

Education in object-oriented (OO) technologies has become a core part of any modern education in software engineering. A well-known problem with education in this field is the relatively long period of time that is required to get accustomed to "OO thinking":

- First, learn to solve problems by building small communities of interacting objects. People with a strong background in classical structured-procedural programming may first have to unlearn their previous algorithm-centered approach.
- Second, adopt the habit of reusing existing classes instead of inventing new ones. This requires you to know where to look for reusable components.
- Third, start to think flexibly about the organization of the software development process. Beginners have to be taught the importance of a formal software life cycle and of a proper requirements analysis (see e.g. Wilkinson 1995, Wirfs-Brock, Wilkerson & Wiener 1990 and Rumbaugh, Booch & Jacobson 1997).
- The most advanced concepts we teach to our novices are patterns and frameworks (see Gamma, Helm, Johnson & Vlissides 1994, Froehlich, Hoover, Liu & Sorenson 1998). *Patterns* describe concepts of proven solutions for recurring problems: when, where and how to apply them. *Frameworks* are application skeletons that can be turned into complete applications by providing parameters and/or subclasses of the framework's generic classes.

Here, we describe our current approach, which we have found very useful and which might be applied similarly by other organizations as well. This approach was developed jointly between Dresden University of Technology and the University of the Federal Armed Forces in Munich, and is applied at both universities. The central idea of our

approach is to first provide a rapid and dense education covering all topics mentioned above in an introductory course, which immediately precedes the project course. The project course where student teams are assigned individual tasks requires extensive practical work based on a given object-oriented application framework. This provides valuable experience for programming assignments and advanced courses in the postgraduate phase of their studies.

A strong motivation for putting so much emphasis on using a framework stems from the following observation: In academic as well as industrial settings beginners will often join projects that are already well in progress. Finding out enough about the project's structure to be able to do your job is similar to learning how to apply a framework. Experience with this activity is likely to be reusable to some extent also in a non-object-oriented context. A practical course based on one common framework even offers some more advantages:

- For the organizers it is easy to define a number of similar projects and to scale the projects' complexity from moderate to reasonably hard. Since they are based on the same framework, all the different tasks are still comparable.
- For the students it is simpler to extend the application architecture predefined by the framework than to design it for each application from scratch. Also, the framework provides many domain-specific components that can be used "off the shelf".
- Beginners get a chance to learn good design by example: Frameworks by definition are designed for change. Therefore, they typically exhibit patterns that increase flexibility.

The rest of this paper is organized as follows. First we briefly describe the domain, components, adaptation interface and documentation of our `SalesPoint` framework. The next section outlines the project organization: our aims, the persons involved, the timetable and, most important, our tutoring concept. The final section gives some statistics and recent experience.

The Framework

Our `SalesPoint` framework supports the development of point of sale simulations ranging from simple vending machines to big department stores. Typical applications include an exchange office where you can obtain foreign currency as well as a post office offering stamps and a well-defined set of services. The simulations comprise both business with customers (selling, buying, or renting goods) and administrative tasks (like accounting, refilling the stores, removing slow-moving articles, and putting new kinds of items on sale). All applications from the `SalesPoint` domain share the following characteristics: There is one single *shop* where *customers* are served at a number of *counters* (or *SalesPoints*, in our terminology). At each counter there is a queue of customers. Every counter offers articles from some fixed *catalog*. For each article, the catalog has an entry giving its name, price, and other relevant properties. A *stock* is a bag of articles from the catalog. Examples of stocks are: the goods on an order form, the articles contained in a vending machine, in the shelves of a store, or in a customer's shopping basket. *Money* fits into this terminology as a special case: Here the catalog is called a *currency*. It describes the set of valid bank notes and coins and their values. The contents of a personal purse or those of a cash register are called *money bags*. `SalesPoint` customers have *data baskets*, which may contain a number of goods chosen by the customer. Thus data baskets closely resemble the shopping baskets carried around by real customers in real shops. The contents of a data basket represent the state of a customer's current shopping *transaction*. Like other transactions, data baskets can be *committed* (e.g. when the goods are paid for and taken out of the shop) or *rolled back* (i.e. the goods are restored to the shelves they came from).

All `SalesPoint` applications have similar organizational and GUI requirements. Accordingly, the `SalesPoint` framework supports the development of point of sale simulations by providing (among other things) similar GUIs built from the same components including generic form and menu classes for user interaction. A shop is represented by a main window, which contains a set of subwindows, one for each customer at the currently visible counter. Tabular form components are available for presenting catalogs and stocks. The common organizational part comprises *base classes* for catalogs, stocks, currencies, and money bags, *persistency management* (on request the state of the simulation can be stored in a file to be restored again later), *user management* (allowing users with possibly different capabilities to be created), *time management* (for controlling the simulation time), *transaction support* including rollback, and a *logging mechanism.*

On a more technical level, framework users expect their framework's software to be *robust* and *flexible*. Among the robustness features of the `SalesPoint` Java implementation that require no activity on the part of the applica-

tion developers (and thus might even go unnoticed by them) are: `SalesPoint` data structures are *thread safe*, i.e. they can be accessed concurrently from different threads. `SalesPoint` guarantees the *referential integrity* of catalogs and stocks, i.e. you cannot add an item to a stock if there is no matching catalog item in the corresponding catalog. And you cannot simply remove a catalog item without first removing all corresponding items from all stocks based on this catalog. These properties are implemented using suitable Java *Event* and *Listener* objects. In order to make the `SalesPoint` framework flexible suitable design patterns were applied, e.g. the *Factory Method* pattern for creating stocks and the *Bridge* pattern for choosing between different time management implementations.

Like other frameworks, `SalesPoint` is adapted to its users' needs in several ways: first, by supplying specialized subclasses; e.g. menu sheets are easily adapted using Java's *inner classes*; second, by providing *hook methods*; e.g. the presentation of tables is adapted with hook methods: redefining the method *compare (x,y)* that compares table rows one can impose any sorting order one wishes; third, by providing *parameters*; e.g. when creating a new stock object one of the constructor's parameters describes which catalog to use, another chooses one of the algorithms for building stocks with a given value.

The `SalesPoint` on-line documentation consists of *three complementary descriptions*: a top-down *introduction* to the purpose, architecture and components of the framework, a *javadoc*-generated detailed documentation of all the framework's classes and methods including all the predefined adaptation interfaces, and a *tutorial* describing in a line-by-line fashion how the framework can be used for building a typical application: the simulation of a Fast Food Restaurant.

For more details on the introductory course and the framework see (Demuth, Hussmann, Schmitz, Zschaler 1998) and (Demuth, Hussmann, Schmitz, Zschaler 2000a). On the (English) `SalesPoint` homepage (Demuth, Hussmann, Schmitz, Zschaler 2000b) all the material for using the framework (code and on-line documentation) is accessible.

Organization

The project course organization, which we describe in the following, is very similar at both universities. A few differences come from varying conditions: the large number of students that participate in the project course every year in Dresden; two semesters (Dresden) versus three terms (Munich) per year; and students living on campus in Munich.

Aims

For the majority of students, the project course is the first hands-on experience in software engineering. Prior to that, they were taught (among other things) structured programming and verification techniques, sorting algorithms and data structures including hash tables and AVL trees. Programming tasks were defined precisely, small to moderate in size and typically could be solved by one person in a few days. In order to succeed with their projects students now have to develop a number of new ("soft" and auxiliary) skills:

- They have to learn to work in teams: assume different roles (team leader, developer, etc), communicate in a professional way within the team and with customers, and work systematically for a period of three months following some given method and a plan made by themselves.
- They have to find out what their task is: In contrast to the programming tasks they were given before the project definition is rather sketchy. It is the students' job to find out what to do and to negotiate with their "customers".
- They have to present their work: Both oral and WWW presentations are required. The presentations are aimed at either customers or consultants and thus differ very much in the level of abstraction and amount of technical detail.
- They have to acquire a working knowledge of the tools they need: In addition to some Java IDE they need a CASE tool and some HTML and FTP background as required to set up their own small WWW site.

During the project course a written process outline, guidance by the tutors and feedback from the presentations are available. With this background, the students are expected to develop the soft skills mainly by themselves.

Persons Involved

The students are asked to form teams of five or six persons each and to adopt a *chief programmer team* organization, i.e. to assign chief, assistant, secretary and developers' roles to the team members. The resulting teams are coached by senior students who in turn are supervised by the project course leader. The senior students work as tutors: They are both consultants for the younger students and clients for the software project. Technical questions and requests for framework correction or extension are handled by senior students who have participated in the framework development.

The situation resembles that of a start-up software company specialized in some domain (as represented by the framework). The company's technical staff consists of a technical director, four or five experienced developers (one of whom is responsible for the framework) and a bunch of 50 to 150 beginners who are to be "trained on the job". To avoid disaster, some strict discipline and a lot of communication is required:

- Teams have to adhere to a predefined timetable (see below). Otherwise, management cannot guarantee that projects will be completed in time.
- Development has to be founded on the framework. Beginners tend to do everything from scratch and "reinvent the wheel" many times. Only by using the framework they will contribute to the company's know-how and produce solutions that can be maintained easily.
- Guidance by management (i.e. the tutors) is impossible unless progress is documented extensively all the time. The company's work process description therefore lists what kind of documents have to be produced in which phase.

Teams that will not stick to these rules are excluded from the project course.

Timetable

A rather rigid timetable is prescribed for project work. Once or twice during each phase, results (documents, programs) have to be presented to the tutors. Final delivery includes a formal oral presentation per team where the main results including the working program have to be shown and questions to be answered. The timetable is as follows (The number of weeks per phase is given for Dresden/Munich. A Dresden semester has approximately 13 weeks, a Munich term only 11 weeks.):

Getting Started	... 3/3 weeks
OO Analysis	... 2/2 weeks
OO Design and Prototyping	... 2/2 weeks
Implementation and Test	... 3/2 weeks
Maintenance	... 3/2 weeks

Table 1: Project timetables

Alternatively, incremental development with several development cycles is allowed. This approach is adopted by most teams. In the *Start* phase students establish their teamwork organization, obtain and install the development and documentation tool sets as well as the `SalesPoint` framework software and documentation. During this phase the students also have to study the framework and its tutorial. We experiment with different approaches to learning the framework. For example, each Munich student has to implement the same and rather simple `SalesPoint` application at the beginning of the project course. This takes time (3 weeks), but it helps the students to develop their main application much faster than when learning the framework in parallel with the software development. In the *Analysis* phase, students first identify the use cases of the system to be built. CRC card sessions are then used to elaborate corresponding scenarios in order to better understand the dynamic behavior and the classes of the target system with their responsibilities. OO modeling has to be documented in UML notation (Rumbaugh, Booch & Jacobson 1997) using static, use case, state and sequence diagrams. In the *Design* phase, suitable framework components are identified for implementing this model. The model is adapted accordingly and a first prototype can be built. The *Implementation and Test* phase is devoted to growing and testing the prototype. The *Maintenance* phase includes the removal of bugs and satisfying some minor clients' wishes. The latter serves as a test whether the design is clean enough to allow for easy modification.

Net-Based and Direct Tutoring

All information is distributed via *WWW*: the framework, its documentation, the tutorial, and the project specifications. Using *email* help can be obtained all the time from the tutors, the frameworks specialists, from system administrators, and from the course manager. Framework bugs and requests for new framework features can be entered into a *web-based list*. A public *electronic calendar* is kept for booking presentation dates and rooms. Some teams even install their own private *electronic chat-rooms* and *bulletin boards* for communication within teams. All these communication channels are used rather heavily indicating that without net technology this kind of course might not be possible - or at least take much more time for all persons involved.

The students are required to present their solutions on HTML pages. For this purpose, each team is given an account on one of the institute's web servers**.** *Student team web sites* are the most important prerequisite for net-based tutoring**:** therefore, the students are required to publish their results as early as possible and to update them on a daily basis. That way, the tutors and the course manager can asynchronously observe student progress all the time. This allows for instant (email) feedback and thus helps to avoid mistakes and too much time spent taking wrong turns. For beginners this kind of continuous guidance is very important.

However, net-based tutoring alone does not suffice: In difficult situations, inexperienced developers cannot be expected to even state their problems precisely. Some find it difficult to overcome the psychological barrier of asking an anonymous consultant for help. To get them started, tools and ways to work with the documentation have to be shown them directly. Although student web sites are really helpful, not all mistakes show up there. And sometimes it is easier to give feedback when meeting in person.

At the end of every phase (Dresden) or every week (Munich) students have to present their progress orally. In a short *presentation* of 15-30 minutes at most the teams demonstrate the latest version of their prototypes; each developer explains his or her contribution showing documentation and code fragments, and answering questions from the tutors or course managers, respectively. This is a way of ensuring that all team members actively take part in the development. Also, frequent presentations result in a more regular style of working. In the final presentation, we also collect student feedback for improving the course organization.

Evolving the Framework-Based Project Course

Since winter 1997/98 when we first taught the framework-based project course we have improved both the used framework and the project organization every year (Demuth, Hussmann, Schmitz & Zschaler 1998). A major step was taken in winter 1999 (Munich) and summer 2000 (Dresden), when we upgraded successfully to version 2.0 of the framework.

The overall results of the previous project courses are very encouraging. In Munich, there were no dropouts so far. This is probably due to the fact that in Munich almost the same number of tutors are employed as in Dresden while the student numbers are much smaller. Another advantage is that Munich students are paid by the armed forces and live on campus. The table below specifies the number of teams and the total number of students for each project course up to now.

	Dresden Winter 97/98	Munich Winter 98	Dresden Winter 98/99	Dresden Summer 99	Munich Winter 99	Dresden Summer 2000	Munich Winter 2000
Success rate	90 %	100 %	91 %	70 %	100 %	88 %	100 %
Teams	22	6	25	2	7	21	9
Students	116	32	120	7	42	115	54

Table 2: Project success rates

During the whole process we had a lot of feedback: from the tutors, some students' questions, many intermediate documents, final presentations with discussions and the detailed questionnaires we requested from the students. We learned that:

- studying the framework took more time than we had expected (about 25 % of the whole effort); in retrospect we feel this justified since it covers a good deal of what would otherwise have been part of the design phase; students rated the tutorial and the on-line support rather high;
- the time table was realistic, given the students' tendency to postpone work towards the end; on an average, the students spent about 10 to 12 hours per week on their projects; some additional time was needed to catch up on missing OO and Java knowledge from the introductory course;
- students generally liked the tasks they were given; some teams even tried to find out *real* clients' requirements by doing field studies; a growing number of students, however, would prefer to develop *real* applications instead of just simulations; students rated their own achievements rather high; for them, team work experience was novel and important;
- in the tutors' opinion, most students performed rather well, but there still seemed to be some who had hacked their way without a true appreciation of OO technology; on the positive side, the framework proved practicable and accommodated all kinds of students' approaches: everyone felt they had learnt a lot.

In the summer 2000 (Dresden) project course another 20 students were sent to a software company to do "real" projects there. Typically, more successful students applied for the external projects. The four teams were coached by a former university colleague. On an average, they spent as much time on their projects as the framework-based teams, but produced twice as much code. Also, they liked their projects very much.

So why not send all the students to industry instead of doing in-house projects? There are several problems: First of all it is difficult to find enough software companies where beginners are systematically trained as opposed to being misused as cheap code hackers. Today, short-term profit appears to rank higher than long-sighted investments in education. Because of the diversity of projects and companies it is difficult to offer equal opportunities to students; also, the organization and necessary quality control would probably bind at least as many university resources as the framework-based course.

On the other hand, our approach should carry over easily to frameworks in other domains and there provide the same advantages for beginners: *realistic professional activity* is simulated and *guidance in the form of a framework* is offered.

References

Demuth, B., Hussmann, H., Schmitz, L., Zschaler, St. (1998). OOPSLA'98, Educators' Symposium. *Using a Framework to Teach OOT to Beginners.*

Demuth, B., Hussmann, H., Schmitz, L., Zschaler, St. (2000a). 13th CSEET, Austin, Texas. *A Framework-Based Approach to Teaching OOT: Aims, Implementation, and Experience.*

Demuth, B., Hussmann, H., Schmitz, L., Zschaler, St. (2000b). *The SalesPoint Framework v2.0 Homepage.* http://ist.unibw-muenchen.de/Lectures/SalesPoint (Also for v3.0: http://www-st.inf.tu-dresden.de/SalesPoint/v3.0, German only)

Froehlich, G., Hoover, J., Liu, L., Sorenson, P. (1998). Designing object-oriented frameworks. In CRC Press (1998) *CRC Handbook of Object Technology*.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design Patterns - Microarchitectures for Reusable Object-Oriented Software.* Reading: Addison-Wesley.

Rumbaugh, J., Booch, G., Jacobson, I. (1997). *Unified Modeling Language Reference Guide.* Reading: Addison-Wesley.

Wilkinson, N. (1995). *Using CRC Cards. An Informal Approach to Object-Oriented Development.* New York, NY: SIGS Publications.

Wirfs-Brock, R., Wilkerson, B., & Wiener, L. (1990). *Designing Object-Oriented Software.* Englewood Cliffs, NJ: Prentice-Hall.