A Framework-Based Approach to Teaching OOT: Aims, Implementation, and Experience

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 technology to undergraduate students. Before they can successfully tackle the projects they have to successfully shift to the OO paradigm, 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 provided 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 by themselves. We also believe that this policy closely resembles the way beginners are integrated into on-going projects in practice. We describe the Java framework we used, the preparations for and the organization of the project course, educators' and students' experience and some ideas for developing this approach further.

1. 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 OO technologies is the relatively long period of time that is required to get accustomed to "OO thinking".

In order to become a proficient OO software engineer, you have to climb several rungs of a very demanding ladder of technical knowledge:

- First, learn to solve problems by building small communities of interacting objects. People with a strong background in classical structured-procedural programming may have to unlearn their previous algorithm-centered approach before they can get used to the new paradigm.
- Second, adopt the habit of reusing existing classes instead of inventing new ones. This requires you to know where to look for reusable components, i.e. you must understand the scope and structure of the class libraries you are using. Getting to know a large class library in sufficient detail will take quite some time.
- 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. More experienced developers, however, will soon appreciate the benefits of incremental development and prototyping that are typical of OO methodology (see e.g. [7], [8] and [6]).

- The highest stage on the ladder corresponds to patterns and frameworks (cf. [5], [3]). *Patterns* describe concepts of proven solutions for recurring problems: when, where and how to apply them. Pattern names are carefully chosen for ease of communication between developers. *Frameworks* are application skeletons that can be turned into complete applications by providing parameters and/or subclasses of the framework's generic classes.

The higher you climb on this ladder, the more leverage you will gain for developing your own applications. However, learning to climb the ladder takes significant effort and time.

At Dresden University of Technology, we try to counteract this problem by teaching OO software engineering to computer science undergraduates very early in their studies: An introductory course in the third semester is followed by a project course in the fourth, where student teams are given individual tasks. This provides valuable experience for later programming assignments and advanced courses in the postgraduate phase of the studies.

Here, we describe our current approach to the introductory course and in particular the project course, 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 provide a rapid and dense education covering all stages of the ladder in the introductory course, up to the level of frameworks and patterns. The project course in the following semester gives the opportunity to extensive practical work based on an object-oriented application framework, i.e. on the highest stage of the knowledge ladder. Detailed education in several aspects of OO technology follows later in advanced courses.

A strong motivation for putting so much emphasis on quickly reaching the level of frameworks 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. A practical course based on one common framework therefore offers several 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. If competition is desirable you can assign identical tasks to different teams.
- As pointed out above, learning your way around a given framework corresponds to a characteristic professional activity. Experience with this activity is likely to be reusable to some extent also in a non-object-oriented context.
- 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 experience gained in extending application frameworks gives a good basis for future work.

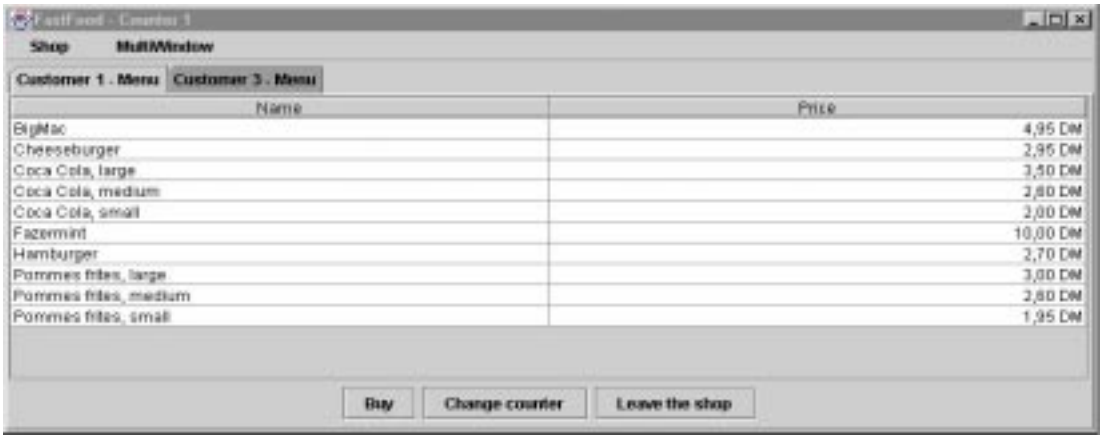
The rest of this paper is organized as follows: First we describe the domain, architecture, and the adaptation interface of the `SalesPoint` framework. The next Section reports how we prepared the students for their project work. Then we outline the project organization and briefly report on the results from a detailed student's questionnaire. The last section describes how our concept has evolved over time and indicates some future extensions.

2. The framework: user's view

The users of the `SalesPoint` framework are software developers, in our case undergraduate students. The 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, a post office offering stamps and a well-defined set of services, a drugstore, or a video shop. 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).

`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. As shown below 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;
- generic form and menu classes for user interaction (two separate layers: the *logical layer* which is part of the framework's adaptation interface, and which is based on the underlying hidden *physical layer*);
- a common organizational part comprising *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) and *time management* (for controlling the simulation time);
- base classes for catalogs, stocks, currencies, and money bags;
- transaction support including roll-back and logging mechanisms associated with data baskets;
- different algorithms for the standard problem of building a stock for a given value (needed e.g. for returning change money or at a post office for assembling a collection of stamps with a given total value).

Like other frameworks, `SalesPoint` is adapted to its users' needs in several ways:

- by supplying specialized subclasses; e.g. menu sheets are easily adapted using Java's *inner classes*;
- 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; similarly, editing and formatting of table entries can be controlled;
- 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.

In order to prevent the students from modifying the `SalesPoint` source files, they are given only the framework's Java class files. Otherwise, we would get slightly different versions of the framework for each application and thus a major maintenance problem. This is one of the reasons why commercial distributors supply only compiled versions of their products.

The `SalesPoint` framework comes with a comprehensive on-line documentation consisting of *three complementary descriptions*:

- a top-down *introduction* to concepts and notions on frameworks in general, and to the purpose, architecture and components of the `SalesPoint` framework,

- a *javadoc*-generated detailed documentation of all the framework's classes and methods including all the predefined adaptation interfaces (HotSpots and Hooks) (see screenshot above) and
- a *tutorial* describing in detail how the framework can be used for building a typical application: the simulation of a Fast Food Restaurant.

For documentation of adaptation interfaces, the methodology of hook descriptions is taken from [4]. The documentation covers all aspects of the framework's pplication interface for use as a *black box*, and also explains some of its internal structure for *white box* use.

3. The framework: implementor's view

As indicated above, the `SalesPoint` framework is implemented in Java (see e.g. [2]), currently using version 1.2.

Framework users expect their framework's software to be *robust* and *flexible*.

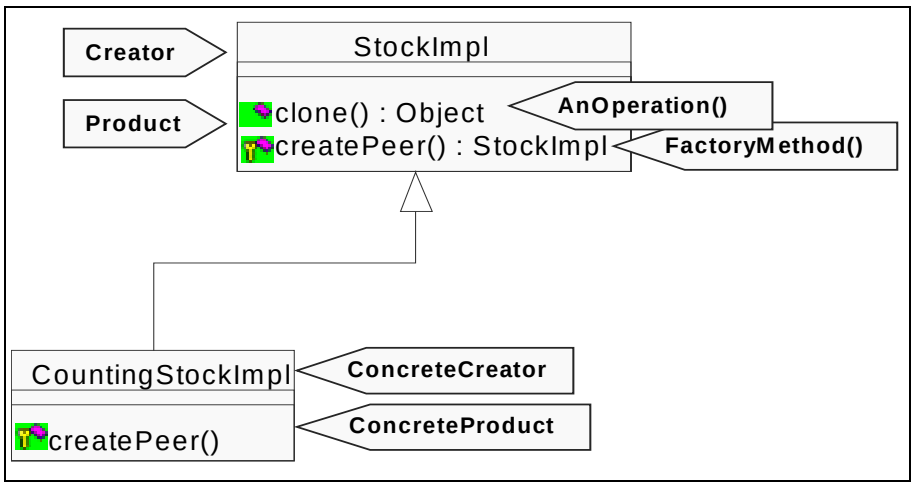
Among the robustness features of the `SalesPoint` framework that require no activity on the part of the application developers (and might even go unnoticed by them) are:

- `SalesPoint` data structures (including data baskets) are *threadsafe*, i.e. they can be accessed concurrently from different threads. Internally, access to data structures is guarded by lock objects.
- `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

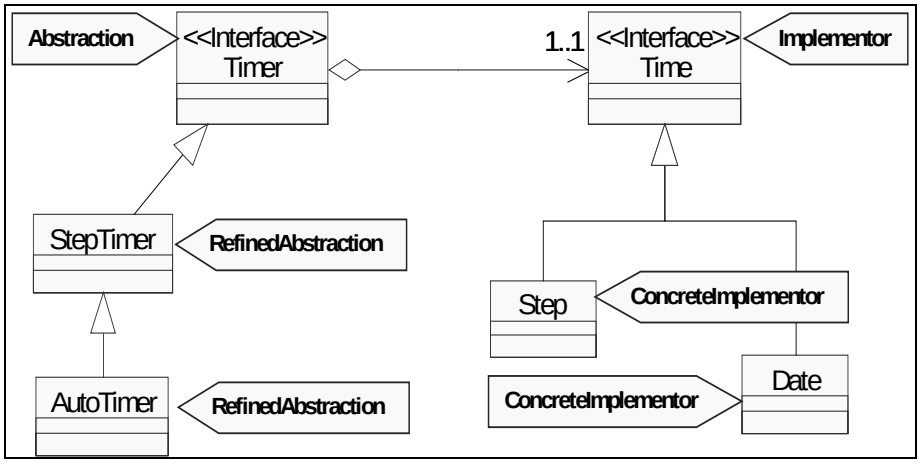
items from all stocks based on this catalog. These properties are implemented using suitable Java *Event* and *Listener* objects.

In order to make the `SalesPoint` framework flexible, some of the patterns from the [5] collection of design patterns were applied, e.g.

- the *Factory Method* pattern for creating stocks:



- and the *Bridge* pattern for choosing between different time management implementations:



4. Climbing the ladder

The basic knowledge for successfully using the framework is taught in the introductory course preceding the framework-based project. As a consequence of our general concept, the introductory course is a quite demanding effort for teachers and students. Within one semester, we aimed to provide the students with:

- an appreciation of OOA/OOD methodology and of requirements analysis, in particular;
- a thorough understanding of OO notions and terminology including familiarity with a core subset of UML notation;

- sufficient practical experience with the Java programming language as required for the intended projects along with a working knowledge of core parts of the Java class library;
- a good idea of patterns, frameworks, and how they are used;
- some basic knowledge about project organization, sufficient for starting a team project in the following semester.

We chose the programming language Java (instead of C++, which was used in former years) for well-known reasons: Java is a more pure OO language than the hybrid C++. It has useful new concepts like *interfaces*, includes automatic garbage collection and it avoids some of the error-prone features of C++ such as pointer arithmetic and multiple inheritance. Therefore, it seemed (and has proven) better suited as a first OO language. Moreover, Java implementations are available for almost all platforms at no or little cost. It should be stated explicitly that the current popularity of Java was only a very minor factor for the decision.

The lecture comprises nine major topics as listed below. The figures given in brackets indicate the approximate number of teaching hours in the lecture. Lectures are supplemented with lab exercises (in groups of approx. 12 students) and homework for hands-on experience. Due to its top-down structure, the lecture is relatively theoretical in the beginning (topics 1 to 3). The lab exercises during this period are used for a practical introduction into Java, based on the traditional programming course the students had to take in the preceding semester.

1. *Production of complex software systems (2):* Typical problems of large software projects, overview of the discipline 'Software Engineering', basic process models;

2. *Object-orientation (2):* Basic concepts of object-orientation, comparison with different paradigms for programming and modeling, historical overview of development methods;

3. *Object-oriented analysis (6):* CRC cards, use cases and scenarios, static and dynamic modeling with UML;

4. *Object-oriented design (8):* Software architectures, architecture description with UML, realization of UML designs with Java, design and selection of data structures (Java collection framework in the upcoming instance of the lecture);

5. *Architecture of interactive systems (4):* Event-controlled programs, Model-View-Controller architecture, principles of user interface construction using Java AWT (Swing in upcoming instance of the lecture);

6. *Reuse (4):* class libraries, application frameworks (using AWT/Swing as examples), a selection of classical design patterns;

7. *Parallel flow of control (2):* concurrency, Java threads, synchronization;

8. *Software quality assurance (1):* principles of process-integrated quality assurance, test and integration phases;
9. *Project management (1):* Basics of project planning, version control, and documentation.

This scheme turned out as successful on the whole. Of those students who actively participated in the exercises, about 80% were able to pass a written test. However, for one group of students the results are quite unsatisfactory: These are students with extensive previous programming expertise in traditional programming languages. These students tend to over-estimate their ability to adapt to an object-oriented programming language, and

therefore do not participate actively in the Java exercises. The end result is that they often pass only those parts of the test which refer to theory of object-orientation (e.g. UML modeling) but fail the Java programming part.

A particularly good experience can be reported from the CRC card sessions we organized for student teams of about five persons each. Most students overcame their inhibitions and began to discuss their analysis and design problems freely and vividly. Also in an academic setting, CRC cards turn out as a very helpful tool to learn "thinking in objects".

5. Experience with framework-based project courses

After the introductory course, we perform a one-semester project course. Since winter 1997/98 when we gained our first experience with the described framework-based teaching approach we could improve both the project organization and the used framework every year [1]. Currently, version 2.0 of the `SalesPoint` framework is introduced the first time at the University of the Federal Armed Forces Munich.

The project organization which we describe in the following is very similar at both universities. A few differences come from varying curricula and the large number of students that participate in the project course every year in Dresden. Therefore a rather formal mode of organization is needed. All information is distributed via WWW: the framework, its documentation, the tutorial, and the project specifications. The students are encouraged to present their solutions on HTML pages in the same way.

The students are asked to form teams of about five 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 the student who had developed the framework.

A rather rigid time table is prescribed for project work. At the end of 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 time table is as follows (The number of weeks per phase is given for Dresden/Munich.):

Getting Started	... 1/3 weeks
OO Analysis	... 2/2 weeks
OO Design and Prototyping	... 2/2 weeks
Implementation and Test	... 4/2 weeks
Maintenance	... 4/2 weeks

Alternatively, incremental development with several development cycles is allowed. This approach is adopted by most teams. The students start with activities such as finding the specific project organization, and lectures on teamwork organization, the `SalesPoint` framework and related problems. During this phase respective OO analysis the students also have to study the framework and its tutorial. We experiment with different approaches in 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. OO modeling has to be done in UML notation [6] using static, use case, state and sequence diagrams. Maintenance includes removal of bugs and satisfying some minor client's wishes.

The overall results of the previous project courses are very encouraging. The table below specifies teams and students that took part in the project courses. For summer 2000, we expect about 250 students to take part in the project course in Dresden.

	Dresden Winter 97/98	Dresden Winter 98/98	Munich Winter 98	Dresden Summer 99
success rate	90%	91%	100%	70%
number of teams	22	25	6	2
number of students	116	120	32	7

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 8 to 10 hours per week on their projects; there also seemed to be some backlog in the form of missing OO and Java knowledge from the introductory course;
- students liked the tasks they were given; some teams even tried to find out *real* clients' requirements by doing field studies;
- the assignment of identical tasks to different teams resulted in astonishingly different solutions in spite of their using the same framework;
- 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.

At [10] respective [11] all the material for using the framework is assembled, most of it in English. If you can read German, you can also have a look at the tutorial. Project courses, their organization and some of the students' results are recorded at [9] (mostly in German).

6. An evolving process

Before arriving at the current course organization, we had tried out different approaches at different places as described below. Some hints on how to further evolve it are given in the end.

Learning a new programming paradigm without some practical experience seems impossible. Therefore, conventional written exams were replaced in Munich by individual homework projects: students were allowed to work in teams. After two weeks they had to present their design, after two more weeks to demonstrate a working (Smalltalk) prototype in class. In those courses, the primary focus was on OO programming techniques and on reuse, in particular. Students enjoyed the projects and, therefore, worked hard to achieve these goals.

While at first many different and unrelated tasks were given to the students, a few years later a more ambitious approach was chosen: Different tasks from one common domain were handed out to the students. After some teams had completed their prototypes, other teams tried to distill their experiences and solutions into a framework. Then a third group of teams were to do reimplementations of the original prototypes using this framework.

Both times we followed this new approach we were only partially successful: Still, the students with much enthusiasm finished their prototypes. But the task of abstracting the common parts into a framework and proving its usefulness by doing reimplementations turned out to be too hard. Within the short time available and with their limited experience, students produced frameworks that were too immature to be applied successfully. Another problem with the new approach was that projects had to be started before all the relevant techniques had been taught.

In the current organization, the above problems are avoided: the latter one by decoupling the introduction to OO technology and the project course, the former one by providing the students with a well-prepared framework in advance. Remaining minor problems appear to be organizational ones that we shall try to remedy as indicated.

At Dresden, previous courses had been based on OMT, the Booch method, and C++ programming language. In the lectures, emphasis was on OO analysis and design. An elementary introduction to C++ was given during lab exercises. Beyond that, students were on their own to find out about the C++ constructs and libraries they needed to do their projects. By switching to UML and Java we not only modernized our curriculum but also had the chance to teach OO programming in sufficient detail.

The existing framework can (and will) be improved in many ways. Possible major extensions include:

- using the Java JDBC interface to make catalogs and stocks persistent by storing them in a relational data base;
- using the Java RMI or CORBA interface to obtain families of cooperating `SalesPoint` applications distributed over the net; e.g. competing shops might buy their goods from different wholesalers who in turn obtain them from different factories; here, shops, wholesalers and factories all share the `SalesPoint` characteristics.
- providing a `SalesPoint` Application Builder (SAB) program: Given a suitable high-level description of a `SalesPoint` application SAB would generate those parts of the `SalesPoint` simulation program that are simple yet tedious to write. In other words:

SAB would be similar to a GUI builder. Such a tool can be implemented using the Java Bean technology.

Also, 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.

Acknowledgements

We thank our students Axel Grossmann, Jens Stuendel, Brit Engel, Martin Hamsch, Sandro Herpich and Nane Kratzke who, through their efforts and contributions as tutors, made the organization and execution of the described project courses successful. For `SalesPoint` version 2.0, Stephan Gambke and Sven Matznick provided domain specific Swing components and a new tutorial.

References

[1] Demuth, B., Hussmann, H., Schmitz, L., Zschaler, St. *Erfahrungen mit einem frameworkbasierten Softwarepraktikum.* in: Tagungsband des 6. Workshops Software-Engineering im Unterricht der Hochschulen, Teubner-Verlag, 1999

[2] Flanagan, D. *Java in a Nutshell.* O'Reilly & Associates, Sebastopol, 1996. Also extended second edition 1997.

[3] Froehlich, G., Hoover, J., Liu, L., Sorenson, P. *Designing object-oriented frameworks.*, in CRC Handbook of Object Technology, CRC Press, 1998.

[4] Froehlich, G., Hoover, J., Liu, L., Sorenson, P. *Hooking into Object-Oriented Application Frameworks.*, in Proceedings of the 1997 International Conference on Software Engineering, Boston, Mass., May 17-23, 1997.

[5] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. *Design Patterns - Microarchitectures for Reusable Object-Oriented Software.* Addison-Wesley, Reading, 1994.

[6] Rumbaugh. J., Booch, G., Jacobson, I. *Unified Modeling Language Reference Guide.* Addison-Wesley 1997.

[7] Wilkinson, N. *Using CRC Cards. An Informal Approach to Object-Oriented Development.* SIGS Publications, New York, 1995.

[8] Wirfs-Brock, R., Wilkerson, B., and Wiener, L. *Designing Object-Oriented Software.* Prentice-Hall, Englewood Cliffs, 1990.

[9] Demuth, B. Software Engineering Project Course. http://www-st.inf.tu-dresden.de/ProjectCourse/

[10] Schmitz, L. *The SalesPoint Framework V 2.0 Homepage.* http://inf2-www.informatik.unibw-muenchen.de/Lectures/SalesPoint

[11] Zschaler, St. *The SalesPoint Framework V 1.0 Homepage.* http://www.inf.tu-dresden.de/~sz9/SWTProject/ver05