Original Call for Participation

DOING YOUR FIRST OO PROJECT
OO Education Issues in Industry and Academia

Overview

Doing object-oriented development is more than just object-oriented programming. Good object-oriented education should cover all aspects of object-oriented development. One way to do so effectively is to run an introductory object-oriented development project.

This workshop will address problem areas inherent in project-oriented approaches to object-oriented education.

Goals

To set up your first object-oriented project the availability of reliable experiences is very important. By knowing the key success factors your first object-oriented project will go faster and have a higher learning curve. To reach these goals this workshop will bring together people from industry and academia to

• share and discuss experiences from project oriented introductions/transitions to object technology;
• identify key success factors;
• establish an initial checklist with dos and don’ts;
• gather ideas for reference projects;

Focus

The workshop will focus on concrete experiences of successful and unsuccessful approaches and the issues involved in teaching and learning object technology.

Prospective topics include (but are not limited to):

1. the role of the development process (i.e. should all phases be done using object-oriented approaches);
2. the role of management support;
3. the role of previous experiences (oo and non-oo);
4. the role of methods, languages and tools;
5. the role of libraries and frameworks;
6. the role of patterns and design heuristics;
7. the role of formal methods;
8. the role of courseware;
9. the role of the type of the project;
10. differences in object-oriented education between industry and academia.

Requirements for Attendance

Attendance to the workshop is limited. Participation will be by invitation only, based on the organizing committee’s evaluation of the submission.

Prospective workshop participants are required to submit a position paper or experience report before August 4, 1997 to both organizers. Each submission should be accompanied by the authors’ “top ten” recommendations for successful introductory projects. Accepted submissions and “top ten” lists will be made available to all participants before workshop start.

We encourage both industry professionals and educators to submit a position paper on a topic relevant to the workshop. Experience reports describing actual projects are especially welcomed.

Organizers

Vladimir Bacvanski—InferData Corporation
2520 Quarry Road, Suite 101 Austin, Texas 78703, USA
vb@acm.org, +1 (512) 656-8360, +1 (512) 472-3062 (fax)

Jürgen Börstler—Umeå University
Department of Computing Science, SE-90187 Umeå, SWEDEN
jubo@cs.umu.se, +46 (90) 786-6735, +46 (90) 786-6126 (fax)
Project-Oriented OO Training—Experiences from Academia and Industry

Vladimir Bacvanski and Jürgen Börstler
InferData Corporation, USA and Umeå University, Sweden
vb@acm.org and jubo@cs.umu.se

Abstract
This paper discusses the basic problems with initial projects using object-oriented approaches. The authors present their experiences from doing such projects in academic and industrial environments. From these experiences some critical success factors are identified and guidelines for successful projects are derived.

Applying the learned object-oriented project is of crucial importance not only for the proficiency in the object-oriented approach but also for the acceptance of the approach in industry. The problems as well as positive experiences of first object-oriented projects in academia are presented. At the end the authors outline the path that leads to more successful pilot projects and bring the academic and industrial projects closer.

Introduction
Object-oriented development is becoming the “standard” way of software development. Industry and academia are reacting to this situation and offer an increasing number of courses in object technology.

Shifting from traditional software development approaches, based on functional decomposition, to object-oriented development is not just a matter of program languages. Serious object-oriented development requires a new way of thinking.

It is generally agreed that object-oriented programming courses alone are not sufficient to manage this paradigm shift. Grechenig and Biffle ([GrBi93]) show interesting examples of attempts to object-oriented design with the traditional paradigm in mind. They conclude that “[i]f you plan to introduce the OO paradigm, you will have to teach OOA and OOD to your developers thoroughly.” Even strong advocates of the “programming language first” approach (e.g. [Hoh96] and [Mey97]) admit that language classes must not ignore object-oriented analysis and design.

Object-oriented training should cover all aspects of object-oriented development. Effective training requires case studies and hands-on experiences. So, what
could be better (and more realistic) than running a project to gain first-hand experiences?

In this paper we discuss problems and experiences from running such introductory projects in industrial and academic environments. The next section contains a list of problems encountered when preparing or running introductory object-oriented projects. In section 3 the goals of a workshop on this topic are discussed. Section 4 presents the authors’ experiences with introductory object-oriented projects and a discussion of similarities and differences between industrial and academic projects. A conclusion and summary concludes the paper.

Problems in Project-Oriented OO Training

The problems associated with project-oriented training in object technology are the same as with any other training in general. But there are some more or less technical problems, which make things a bit more problematic with introductory projects in object technology.

Some of these problems seem trivial, but they can be of critical importance for the training success. Even trivial problems can become hard ones, when they multiply. When you are going to prepare study material for an introductory project, you will soon detect the following:

- **Lack of standard terminology.** Training material put together from different resources tends to use different or even conflicting terminology for similar things. This can be a very hard problem, especially when the audience has previous knowledge (from probably different resources).

- **Lack of well-defined processes and deliverables.** Many popular approaches provide only fuzzy processes and deliverables. This makes it very difficult for novices to get started. Where does the development process start; What is the input to the analysis phase; How do requirements engineering and OOA relate; Where is the borderline between OOA and OOD, are some typical beginners’ questions.

Fortunately things are becoming better with the so called “second generation object-oriented methods” (e.g. [WaNe95]).

- **Focus on languages/notations.** Many resources focus on specific programming languages or design notations. To explain all of their “bells and whistles” a special terminology is introduced, which makes it very difficult to grasp the essentials. Many of these resources add to the confusion between language specific issues and object-oriented concepts in general. Things are becoming better with design notations. Recent trends towards standardization of notations are welcomed. But from the educational point of view standard notations are not sufficient. To use a notation effectively a clear process and sound deliverables are necessary.
• **Too much importance attached to inheritance.** Inheritance is often presented as the most important contribution of the object-oriented approach. It is often neglected that the effective use of inheritance requires a thorough understanding of abstraction, information hiding, polymorphism, and typing issues. Novices do often think that inheritance has to be applied, when doing object-oriented development and wonder when we question its use.

• **Lack of pedagogical evidence.** There are no studies comparing approaches to object-oriented education. The literature does only provide anecdotal and subjective experience reports (as this one for example).

• **Lack of good and worked through examples.** All examples we know of are oversimplified, focus on specific development phases, and/or come with no or incomplete documentation. To provide complete examples is not easy, since there is no single right way to develop a non-trivial piece of software. But novices like examples, since it is much easier to discuss the pros and cons of an existing solution, than to develop one from scratch.

  Apart from this you can often find examples, which are not very suitable for object-oriented development. Suitable examples should exploit the benefits of object-oriented development and not exhaust themselves in yet another window hierarchy for a simple database application ([GrBi93], [Bör97]).

• **Availability of easy to use, but still useful tools.** A useful tool should reduce the complexity of tasks and not add to it. Furthermore it must not be based on an implicit development process. A simple drawing tool without any semantics associated to nodes and edges might be more useful than a full-fledged CASE tool.

• **Lack of good textbooks.** There are still very few textbooks covering the complete object-oriented development process. In an industrial environment it might be sufficient to focus on analysis, design, and implementation, since people are used to planning, teamwork, testing, version control, etc. In an academic environment these things definitely need to be covered in detail.

### Experiences

In this chapter we will discuss some of our experiences in doing project-oriented introductions to object technology.

In academia most team-project oriented courses are on graduate level. Since programming is already taught in most schools, the students do already have considerable programming experience in developing small to medium-size programs. But this experience usually covers only the traditional function-oriented paradigm. That means that education and industry are faced with the same paradigm shift problem, i.e. changing the developer’s mindset from a function-oriented to an object-oriented one.
But there are also some subtle differences between the academic and industrial environment. In industry most people are used to working in teams according to a more or less defined process and hard deadlines. This is not the case in academic environments. Undergraduate students as well as most graduate students are used to assignment-type practical experiences, where the work products are not (re-)used by other students. This oversimplifies software development and neglects some important aspects completely:

- problems due to inter-and intra group dynamics;
- ripple-effects caused by badly performed tasks;
- version and configuration management;
- testing and quality assurance.

**Experiences from Academia**

In academic environments the project schedules are traditionally bound to courses. For the last three years we ran our object-oriented project course in an eleven week schedule. The course covers all development tasks starting from project acquisition and planning to the development of operational prototypes.

To mimic industrial development students form company-like teams consisting of 4-6 students. Each team must contribute at least one project proposal to a list of available projects. This list is completed by proposals provided by our staff and even projects from local industry. Teams must not select their own proposals for further development. This restriction forces the teams to contact a “customer” for requirements gathering and have the document signed off by the customer. Before we introduced this restriction the teams delivered very bad requirements documents, since they were not sure about what to write down (“this is obvious, isn’t it?”).

The projects are guided by tight deliverables (nine in total, see http://www.cs.umu.se/tdb/kurser/TDBC18/CourseDescription.html for details). The deliverables are checked with short feedback-cycles to help keep the teams on track.

To make the project as “realistic” as possible we require each team to subcontract another team for a small part of its prototype implementation. This faces all participants with the importance of well-defined interfaces. Each team must also perform an evaluation of another team’s prototype. This motivates the production of a user manual and assists the lecturer in project evaluation. Furthermore do all teams have to present their results from time to time to facilitate lively discussions on different issues.

Our experiences so far with this kind of project course are very positive. Students like the course, even though they complain a lot about the heavy work-load and the tight schedules. Many students do even complain about the lack of tool support and that there still is too much programming involved. Although we do not agree with for example Hohman or Meyer ([Hoh96], [Mey97]) that the training success of an object-
oriented project mainly depends on the programming part we do admit that programming is necessary to validate the design.

Our experiences with this and similar courses are summarized below. Most of the experiences are similar to the findings reported in literature (see for example [Ada93], [Ber95], [GrBi93]):

- **Select the project carefully.** You should avoid typical business applications, like inventory and accounting systems. Their simple architecture consisting of a main menu, a layer of (functionally related) sub-menus and a database layer leads to function-oriented programs in object-oriented disguise.

  Such systems are often simply some kind of data entry and viewing system, without any object behavior. I.e. behavior modeling, which is a very important aspect of object-oriented analysis and design, cannot be taught adequately in such projects. The application of use case analysis, scenario modeling, state transition diagrams, etc. is usually not needed to model the systems behavior. The students therefore avoid the usage of these techniques, since they only add complexity to the analysis and design phase.

  We have successfully used games and (real-world) simulations. Such projects are very attractive for students and offer great opportunities for behavior modeling. Modeling the effects of the rules of a game on the behavior of the players or modeling the interactions of some (real world) objects in a simulation. Most games and simulations are easy to understand, since students are familiar with both domains. But they do also offer challenging opportunities. You can easily motivate changing requirements and play through scenarios that show their effects on the model. A very good example is to slightly change the rules of a game or change the number and types of dice. In doing so students can experience the importance of design, the planning for change, and reusability.

- **Do not use hybrid languages.** Hybrid languages like Turbo Pascal and C++ do not enforce encapsulation and allow for functionally decomposed solutions in object-oriented disguise. If students are not forced to apply the new paradigm, they might be reluctant to do so (see [GrBi93] for some interesting examples).

- **Produce a traditional requirements document first.** According to our experiences the biggest single obstacle in object-oriented development is to find the objects. Most resources treat this subject superficially. Virtually all available resources start with very general problem statements as input to OOA. Once an object or a class is brought up it is easy to convince students from its usefulness. But how they are discovered seems to be some kind of magic to them.

  We try to come around this problem by first developing a traditional requirements document in collaboration with the users. This document is then used as an input for OOA. Doing so, almost all analysis objects and classes are already present and can be discovered instead of “invented.”
• **Define a clear process.** Beginners need a well-defined process to check their progress. We follow a work product oriented process. For each work product we describe its intent and its contents. The methods and techniques to develop the work products are the main topics of the lectures. Students appreciate this guidance, since it helps them to concentrate on the basics.

• **Apply design heuristics and patterns.** We have developed an example to explore alternative designs. This example is used to explain the differences between (multiple) inheritance and aggregation and to motivate some design guidelines.

  In our last course we successfully introduced design heuristics ([Rie96]) and patterns ([Gam+95]). Students find this form of concrete advice much more useful than abstract guidelines.

• **Provide mentors.** Mentoring ([Laz95], [Lil96]) is a very effective way of providing the right information at the right time. We used older students as coaches for student teams. They should not actively participate in the development process, but keep themselves up-to-date on status and progress of a project.

One of the hardest problems in the transition to object technology is the paradigm shift from a function-oriented view to an object-oriented view. Studies show that people tend to create models with a centralized control. Unfortunately this is the opposite of what should be achieved in the object-oriented paradigm. Guzdial observed “that students were more comfortable writing decentralized models when they understand the components of the model” and referred to the problem as the **Centralized Mindset Problem** ([Guz95]). The usage of predefined and easy to understand components can probably help to overcome this problem. Some authors propose to completely organize object-oriented education on the (re-)use and development of library components (see for example [HüMe93] and [Mey97]).

**Experiences from Industry**

The critical issue in introducing object-oriented approach in the industry is the selection of the first project. In contrary to the academic environment, where a new project unrelated to the previous work can be started, development in industry is almost always bound to existing systems. The ideal case where an isolated problem can be selected for pedagogical reasons occurs very rarely.

In most organizations, the object-oriented part must fit in the overall system. In order to choose which part of the system to use as a test bed for object-oriented development, an evaluation of the existing system should be performed. Here, it is important to identify the part of the system which is going to benefit most from the object-oriented approach and which is of a suitable size for the pioneer project. Typically, in projects a large part of the code which works well can be identified, and these parts of the code often do not have a tendency to change.
In an organization which aims to develop a new generation of a system in an object-oriented fashion, it may appear that parts of the system that work well are low risk targets for the test of the new approach. A typical mistake there is to assume that the only thing to be done is to redesign those parts of the system in an object-oriented fashion. Such a mapping to a new design has a number of potential dangers. These are:

- the requirements for the part are not clearly documented;
- detailed knowledge about the subsystem does not exist in the company any more;
- developers end up mapping the old architecture to an object-oriented language;
- at some point, management realizes that this activity does not contribute to the bottom line and the project gets canceled, labeling the technology as unsuccessful in real world.

In general, the parts which work well and which are unlikely to change are not ones which are going to benefit the most from the transition to the object-oriented approach. The benefits can be qualified in terms of better encapsulation (thus providing better maintainability), reduced code effort, support for modifiability, enabled reuse, etc.

An additional obstacle can be the adherence to the used process (which often focus on waterfall based models and its deliverables). An unfortunate situation is when the required deliverables are "faked" while object-oriented development proceeds a different way than official documents propose.

For an organization moving towards the new technology, training has a crucial role. The fact that object-orientation introduces a new paradigm is much more significant in industrial environments than in academia, where students do not have problem solving framework fortified by many years of experience. Adopting a paradigm requires a great deal of learning along with deep changes in perceptions, attitudes and actions ([FoFr97]). Particularly for this reason, working with external mentors is of critical importance.

Education of developers for the transition cannot be accomplished merely by attending a course. In the transition process, internal strength must be built. For the first course, a group of top developers should be selected. After they have embraced the technology they will serve as promoters of the approach in the organization and become internal mentors.

When selecting an education program, organizations should look for programs that include mentoring. Mentoring is a very effective way of training ([Laz95]). At the same time, care should be taken with long term consultancy. Long term consultancy tends to prevent development teams from working independently. Make sure that the chosen mentors have proven implementation records.

At the end of the test project, reflect on the experiences gained applying the object-oriented paradigm. Discuss things that worked, things that did not work, benefits achieved and problems encountered. These issues are not necessarily going to be of technological nature. Preferably, the results should be quantified.
In transition to the new paradigm the following problems may occur:

- **Internal resistance.** Particularly the senior members of the team may resist changes. Those are well experienced in the traditional approach and comfortable with their positions in the organization. This situation may start a power shift in the organization.

- **Inadequate education and mentoring.** Educators whose experience is based on object-oriented crash courses and without proven implementation record may not have the deep understanding needed for successful transfer of knowledge and technically leading the transition.

- **Unclear, changing requirements.** For the pilot project, decide on the set of requirements which is going to be covered. If confronted with a large set of changing requirements, prioritize them and concentrate on the most important subset.

- **Overgeneralization.** In the attempt to produce flexible, reusable components developers may create too broad frameworks to solve a large set of similar problems. Such overgeneralization is a common problem of too ambitious projects.

Having an external mentors helps to come around these problems. Having management support for the transition, together with an external mentor reduces internal resistance.

**Summary and Conclusions**

While the project-oriented introductions to object technology in academia and industry differ, there exist a common ground on which a better education and transition program can be built.

Especially in the corporate culture, it is important that the adopted technology provides immediate gratification. Therefore, the initial project must have clear goals and benefits. Software metrics provide quantified insight into the success of achieving the goals. The project itself should have a clear and reasonably scoped set of requirements.

In order for the pilot project to succeed, the core team of top developers should receive high quality education followed by involvement in the project with the help of an external mentor. The mentor ensures that the project stays on the track. In addition, developers learn by apprenticeship from the mentor’s experience and guidance.

This is even true for academic projects. Unlike industry academia has the freedom to choose a project based on pedagogical issues. This advantage should be exploited and care should be taken to select projects,
References


Experiences at ASIC Software Engineering

Dave Isaacs
ASIC Software Engineering
Texas Instruments Semiconductor, USA
isaacs@asic.sc.ti.com

Introduction

Our team’s latest software project began in the second quarter of 1997. One team member was wrapping up initial requirements gathering and two others, myself included, were about to join him and begin requirements analysis and design. The targeted language was C++. None of us had participated in a true object-oriented development life-cycle but agreed to consider OOA and OOD for this new project.

Some eighteen months had passed since we completed, together, OO analysis and OO design courses based on Catalysis (from ICON Computing [1]).

OO Analysis Phase

We approached the OO analysis phase as if we were handed a detailed list of requirements and needed to transform the details into object models. Customers were not involved in “speaking objects”.

System-Context Model

We began with a system context model, attempting to map external actors to the system through high-level actions. We struggled through this process. Much of what the team viewed as external actors involved data sources (such as design databases and electronic component libraries) and we quickly moved to defining use-cases. We then brainstormed domain objects. This proved useful in identifying high-level concepts in the domain and we produced a type-model.

Use Case Development

We completed use cases for a few of the more complicated system responsibilities and were happy to uncover a few major “holes” in the requirements. Motivation to complete more use cases was dampened by lack of tool support and a lack of emphasis that use cases were a primary mechanism for identifying objects in the system.
System-Level Transactions

The next step was to identify system-level transactions. In this way, we were to “exercise” transactions using the type model without assigning responsibility to objects. However, most of the team was uncomfortable with the use of system-level transactions, did not see the benefit, and very few system-level transactions were completed.

OO Analysis and Design Iterations

System Architecture

At this point, there was a progression to defining the high-level architecture, more or less layers, of the system.

The system-level transactions were abandoned and, although we were merely layering our application, we lost sight of any focus on modeling the domain. Instead, our efforts shifted to discussing how Command[2] objects would be created and passed to the Command Layer; how results of the Command objects would be stored, and how the user interface would report the results (and how an ‘undo’ feature would be implemented). I am not suggesting that these issues can’t be dealt with at this phase, but the team did not conceive of objects and their responsibilities separate from the Command objects that represented client requested transactions. We dove too deep, too fast, into detailed design and implementation details of the Command objects.

OO or Representational?

At this point, I had hoped for an object-oriented architecture, full of abstract classes and well-encapsulated behaviors.

The rest of the team wanted what may be called a “Representational”[3] design, where ‘objects’ represented the data to be operated upon but not the behavior. Behavior was to be completely encapsulated in the Command objects and the attributes and associations of the ‘data objects’ would be directly manipulated by the Command objects. There was significant consideration that, in order to allow complete flexibility in how Commands could implement behavior, none of the domain objects should encapsulate behavior (the emphasis was to change a Command object rather than change a domain object, and the team agreed that this approach hinged on the domain objects being very well-defined and rarely subject to changes). There was some controversy in debating whether many of the domain objects could be designed up front and there was disagreement on the benefits of encapsulation, in general.

Modeling: Considering Alternatives

Command objects were to encapsulate the responsibility for dispatching commands, tracking command history, providing for doing/undoing their composite Command
DOING YOUR FIRST OO PROJECT

objects and, ultimately, provide for remote debugging through execution of fine-grain Command objects. Command began to take on the appearance of a “god object”. We began to review key transactions, assigned to Command objects. This was an important undertaking, similar to use case refinement. This modeling effort made much more obvious the risks of abandoning encapsulation in domain objects. Alternative implementations of Command objects were considered, but in all the product functionality remained a responsibility of the Command objects. To the majority of the team, the Command objects provided for the ultimate flexibility, allowing for new operations to be encapsulated in Command objects without the need to modify domain objects to provide new behavior. This may be evidence that procedural habits die hard, that traditional programming leads to centralized control in software architectures and underlines just how difficult it may be for procedural programmers to make “the shift” to OO.

Complexity versus Flexibility

We struggled with concerns of trade-offs between complexity and flexibility. We often chose flexibility at a cost of increased complexity. I became concerned that the design was adding a lot of complexity to achieve run-time adaptability of the software (that is, to extend the system with new behavior at run-time) and advanced debugging capabilities and that we did not have the level of expertise to achieve these goals. We were still learning OO development and it may have been unrealistic to expect to complete such advanced features at the same time, especially without the help of an expert OO consultant.

Resolution

We spent a great deal of time debating design alternatives. Often, this consumed hours in meetings and involved a lot of misunderstandings.

I found that taking the debate out of the meeting room and onto paper went a long way to improving communication of design ideas and eliminating misunderstandings.

If you are debating design alternatives in team meetings and feel that you are treading water, consider simply documenting both design and, most importantly, the rationale for the design (the OO models alone won’t help you choose an alternative). Be sure to include references to industry OO literature and OO design principles, explaining how each design alternative may or may not embrace certain OO principles (that may reduce suspicion that the rationales are merely fabricated). Taking this approach was key to resolving many of the problems on our first OO project. After documenting some design rationale, communication on the entire project improved and we soon found ourselves able to discuss important design principles, such as: “program to an interface not an implementation” [3], “subclasses should not depend on the base
class methods that do not apply to them” [3], and “prefer composition over inheri-
tance” [2].

References

[2] Gamma, Erich, Richard Helm, Ralph Johnson, John Vlissides, Design Patterns: Ele-
ments of Reusable Object-Oriented Software, Addison-Wesley 1995.
Experience Report on a Project Approach to Initiate the Use of Object Technology

Peter Oyserman
Delaware Computing Group, Belgium
oyserman_peter@delaware-computing.com

Introduction
The ideas described in this experience report on making the transition to objects are the result of a set of projects, executed during 1996 and 1997 in a Belgium industrial company. The dos and don’ts and some problem areas described are as well the result of previous experiences in pilot projects with the objective to gain experience in object technology.

The company has its headquarters in Belgium with 4 subsidiaries around the world. It is operating in a design to order environment, software systems capture their core competence.

There was no history on using traditional methodologies at the time the set of projects started, most in-house software development was done ad-hoc. The company operates in a mixed Windows, VMS environment. At the beginning of the projects, no substantial experience in graphical client/server applications was available. Used programming languages were FORTRAN, a 4GL in combination with an RDBMS and some VB.

An important observation is that the use of object technology fitted in an overall attempt to drastically improve the “Quality of Service” of the whole IT staff of this company, making the objectives of this transition project very different from previous ones. On time delivery and a high user satisfaction on offered functionality were THE Senior Management expectations for the different projects, as such the adoption of object technology heavily dependent on their success. The CIO perceived objects as the mean to fulfil expectations of senior management, he communicated this strategic choice in a consequent way throughout his IT staff.

The chosen software development environment is Smalltalk (VisualWorks) extended with the team tool Envy, no particular case tool is used. The infrastructure is composed of Windows 95/NT/Alfa, Oracle and RDB.

Three projects have been setup in both commercial and technical areas, each of them encompassing the education and guidance of several IT people from the customer.
The Role of Management Support and the Overall Project Approach

A first key element in the successful adoption of object technology are “successful” first projects. As described the key concern of general management is “Results”. An “It works!” feeling is crucial to keep inexperienced people motivated, it provides them with the necessary confidence. Other projects with “learning” and “gaining experience” as the main objective can leave people in a state of uncertainty because to many issues have to be tackled within the scope of the pilot project (the “It looks very powerful, but it looks to tuff for me” feeling).

We (the consultants) took a “commitment” approach to ensure successful completion of the three projects. As such the role of our object mentors evolves from a “Consultants do the work, customer’s people watch” to a “Consultants and your people work together” and “Customer’s people work and the consultants watch” approach, to quote Kenneth Rubin and Adele Goldberg

Approach to Education

The classroom training was composed of:

- A one week training on object basics, Smalltalk and the VisualWorks programming environment (The PPD “Intro to Smalltalk using VisualWorks” course).
- A two day training on a SCRUM-like project process model and team development using Envy, discussing good attitudes for team development using Smalltalk.
- A three day course on Coad/Yourdon OOA.
- A three day course on the application framework that is used within the projects, covering all aspects of a typical client/server architecture based on shipping SQL between client and server.

A first set of training was setup before projects started, 7 people were trained, another set of training sessions was organized after first deliveries of projects, 5 people were trained.

The key element on good training is “enthusiasm”, students must go home motivated to proceed. Confidence is not necessarily an objective (i.e. it must be gained from the project itself).

One cross-project mentor was available all time during the projects to reinforce basis skills on object modelling, Smalltalk, how to explore the extensive class library, the VisualWorks and Envy toolset. When appropriate “one person” projects, related to the defined projects, were guided to gain more insight in programming and design. The Smalltalk environment contains quite some examples of good design, providing a lot of “educational opportunities”. The interactive nature of the programming environment makes it an excellent educational tool.
A first project concerns the design of products based on unique customer specifications, executed by a mixed team of four developers, and one architect. The architect combines mentoring for two of the developers (customer IT people), the other two developers are experienced Smalltalk developers (Delaware Computing).

Since the trained IT people combine domain expertise with the assignment to construct supporting design software, we let them evolve from domain expert towards developer, one could speak of a gradual training.

They received a training and workshop on CRC/RDD to explore the envisioned solution.

During the first increment, their prime responsibility was requirements gathering. The architect constructed the overall object models, presenting them regularly to the whole project team, at that time, design patterns were introduced as a mean to communicate design decisions. The trainees verified domain rules by browsing the ongoing implementation and tested the first increment, learning how their domain model has been expressed in an object model.

After the delivery of the first increment, major development proceeded for the next increment. Trainees were assigned to implement intermediate enhancements on the first delivery, gaining more in depth knowledge of the design doing so. As their experience and skills grow, they get more involved in the development activities.

A second project setup simultaneously with the first one, concerns a quotation system adapted to the design to order environment of the company. The project is performed using a SCRUM-like development process, iterating over changing business requirements.

A mixed team consisted initially of six developers, one architect and a project manager. We failed to elaborate the necessary skills of one of developers in order to participate in the project. After extensive individual training by the cross-project mentor it was decided to take the person off the project. We failed to interpret indications during the classroom training, making things worse during the first phase of the project. Two trainees remained in the team.

The existence of an application framework, guiding the implementation of the constructed domain model during analysis, and tackling common topics in client/server applications is the key element of this project. It reduced risk because trainees were implementing domain logic, tasks and interfaces in a structured, organized way. The available functionality assured the necessary productivity to attain early results.

The used iterative process model is very appropriate to show the power of language, design and toolset to the trainees, encouraging them to threat programming as a learning experience. The used project process model was crucial to attain senior management expectations.

A third project involving four developers and one architect concerning jobfloor automation was setup, one trainee was part of the team, following the gradual approach from domain expert to developer.
All three projects have been delivered on time, with an excellent feedback on the proposed functionality.

**The Role of Languages and Tools in the Education Process**

The VisualWorks Smalltalk programming environment is composed of a large amount of classes and numerous frameworks addressing different issues like building GUIs, RDBMS integration, etc. ... The availability of literally all implementation details of each of these components is considered to be a powerful feature of this Smalltalk programming environment. This “If you don’t like it, change it!” approach may permit one to adapt certain parts of the library to new demands in an elegant way.

However, this explicit presence of all kind of implementation details within the class library and the way the programming environment, i.e. the debugger and browsing tools, deals with it, can be confusing for a novice programmer learning how to use available frameworks to build his own applications.

One of the major advantages of the Smalltalk environment is the presence of an extensive class library. Getting familiar with all the components in this library is considered to be the steep part of the Smalltalk learning curve. Very often the only way to grasp offered functionality is to browse the library, with a special second nature to separate the essential from the superficial. Better support to learn about existing functionality should be offered by the development toolset.

In order to grasp the VisualWorks application framework the following simplified view of the “Valuemodel paradigm” is presented to introductory students learning about Smalltalk and VisualWorks:

- a 10000 feet overview of MVC is presented;
- the dependency mechanism is introduced;
- the “value” protocol is described as the way a “generalized” widget communicates with it’s model, embracing the dependency mechanism;
- different kinds of valuemodels are discussed;
- widgets are introduced as view-controller pairs, holding a reference to their model and sending them value or value: messages.
• at last the process of connecting valuemodels to widgets is introduced as one of the major responsibilities of the programmer using the application framework. Students learn how to reimplement stub methods provided by the development tools to return an appropriate valuemodel. Different approaches, i.e. succeeding versions of the Intro. Course, on teaching students this application framework have shown that this simplified approach is necessary in order not to overload 90% of the students.

A typical error made by students is to omit the valuemodel around the domain object when reimplementing the stub methods. The erroneous method return will make the Smalltalk message stack pop up. Even though the error message indicates the type of error made, the message stack only consists of methods the student isn’t aware of yet, as illustrated by the dialog presented in Figure 1.

![Figure 1](image)

Another example of this problem can be found in the formatting support of widgets. As simple as a choice (Number, String, etc. ...) in a pop-up menu during GUI painting, the programmer might get a message stack like the one illustrated in Figure 2, which is actually quite logic, if you know the complete implementation on how print policies are implemented in VisualWorks.

The only help the trainee can get today is an expert sitting next to him, explaining what is essential and what is superficial, or, very often rare, documents providing more abstract descriptions of the used library components.
Figure 2

References

Experiences from OO Mentoring

Jutta Eckstein
Integral Development GmbH, Germany
jeckstein@acm.org

Introduction
In this position paper I want to address the different roles which have to be considered when training teams for their first OO project, specifically in comparison with educating a team for a structured project.

The Call for Participation for the workshops lists 10 topics of interest. In the sequel of this paper I will discuss these topics one by one.

Discussion

The role of the development process
There are a lot of discussions going on considering the development process in connection with the OO curriculum. What has to be taught first, OOA/D or OOP? The answer is: it depends: If you teach on the project you should teach the stuff in time, that means start with the topic which is the first thing to do, which is normally OOA. If the trainees are not involved in this phase it is often better to start with OOP, which is followed by an OOA/D education. Why this order? Because the trainees do not have an understanding why the Analysis and Design has to be different than it was in their old world. If they know something about an OOP, they should be able to reflect for themselves, that their old way of doing analysis and design does not fit anymore to the solution space. This means they know about the problem OOA/D helps solving them. On the other hand if you have trainees without legacy experience it is much better to start with OOA/D, because it is easier to grasp the concepts behind it, when using their domain vocabulary. Whereas if you start with OOP with those people there is the risk that they will loose themselves in syntactical problems of the programming language, without ever having the chance to pick up the underlying idea.

The role of management support
It is important to integrate the management in the education plan, because a lot will change for them, too. The management has completely to understand that the process is incremental and iterative which implies, that there isn’t something like the end of the analysis or the end of the design, because we allow ourselves always to step back
and do another iteration. That is the reason, why the estimation for an OO project has
to be done in a different manner; too, or at least the definition of the milestones. If
management does not understand this you can expect many conflicts with the custom-
er.

The role of previous experiences
The educator will have a hard time if neither domain nor IT background is available.
Concerning teaching OO it turned out that it is really true, that it is easier for people
with almost no programming knowledge to pick up the OO concepts, than it is for
people with a programming background. Although this depends on the OOPL which
is taught, e.g. it is very hard for C programmers to catch the idea of OO when learning
C++, because the languages are so similar. And the normal process is to rely on the
things you know than on the difference, which is here the underlying paradigm.

The role of methods, languages and tools
The better the tools are, the more fun is it for the trainees to learn the new stuff and
vice versa. And besides if the tools really provide a lot, the trainees are able to develop
a whole system even in the educational time frame. Good tools (methods and or lan-
guages) also help to concentrate on the topics, like making the trainees to think OO
rather than loosing themselves in the details, like how to deal with the tool etc.

The role of libraries and frameworks
Also that one is important, because if there is already a lot available it is possible for
the trainees to complete something during the training, because they are able to use
predefined components. Also they feel by themselves what is meant with reuse. And
if the libraries and frameworks are really good, they are acting also as a good example.
Although this has also several risks inside: (1) Libraries as well as frameworks could
also be overwhelming; (2) if they do not use a good style, they are also acting as an
example, this means the trainees will also adapt this bad style.

The role of patterns and design heuristics
This more an advanced concept, or rather a tips and tricks thing. If these topics are
covered too early, they may overwhelm the trainees. The people really have to have
a full understanding about the concepts before you can start talking about this topic.
Teach the stuff just in time: If there are patterns used in (educational) project, men-
tion those concepts as patterns, but do not explain them in detail. This way the train-
ees are getting also an idea about reuse or reliance on a working concept.
The role of courseware

Courseware is a kind of a metric tool. On the one hand it is a frame which could be used to measure the progress of the trainees, also in comparison with different teams. On the other hand it is more or less a controlling instrument for the educator to figure out what works and in which order, and what does not. Without courseware it is hard to remember what, where and how to improve. And besides, this is also another kind of reuse.

The role of the type of the project

Always try to keep the first project small. If that is not possible—what is rather normal—then it is absolutely necessary to bring in more mentors. If OO is new to the team members than at least the domain has to be familiar, specifically if the project deals with a rather complex domain, like telecommunication, insurance or finance.

Differences in object-oriented education between industry and academia

In industry:
• Time is money, most often has nothing a higher priority than money.
• Team members often still have to maintain some legacy stuff, so they often can only work part time on the project.
• And they have to live in “two worlds.”
• The management almost always is not really interested in an elegant and easy-to-reuse solution, but in one which comes as soon as possible to the market.
• Social difference: Team members often have a family so they have additional other obligations.

In academia:
• The objective is often to research, to explore and learn and to find an elegant solution.

References

Introducing Object-Oriented Technology in Industry

Wei Li and John Talburt
University of Alabama in Huntsville, USA
and
Acxiom Corporation, USA
wli@cs.uah.edu and jtalbu@acxiom.com

Abstract

Software industries have embraced the object-oriented (OO) technology for years. We would like to share with you the good and bad experiences of pushing for the OO technology in industry, as well as our experience of industry-academia collaboration in Software Engineering education and application.

Making the Transition to Object-Oriented Technology

Our first attempt to introduce the object-oriented (OO) technology was a redesign of one subsystem in a legacy system. Our interpretation of OO technology is to use OO analysis, design, and implementation techniques to develop software. The project team has 11 programmers and one customer-service manager. Our plan is to introduce OO technology to the team and supervise the team during the software development.

The team is responsible for maintaining the operation of the legacy system. Redesigning part of the legacy system is added on to their responsibility list. We devised a plan to accommodate the team’s daily responsibility and their plan to use OO technology in the new project. The choice of OO analysis and design technique for our first OO project is Booch Method [Booch 1994]. Our plan was to provide an on-site training for Booch method for the team engineers for two weeks. We would then work with the engineers to solve the problems as they encounter them during the application of the method. Therefore, our plan to the technology transition emphasizes: (1) management commitment to the OO transition; (2) on-site OO training using the Booch OOA/D method; and (3) working in the trench with the engineers (12-member team).

We had the management commitments from all levels: CEO, department, and team administrations before we started the project. We were pleased to have those commitments. The CEO of the company and the CEO of the company, the department

1. This work is supported in part by Acxiom Corporation through Arkansas Science and Technology Authority Award #96-A-03.
manager, and the team leader attended our training seminar with the rest of the project team.

We carried out a 16-hour training seminar for the project team. The training seminar lasted two weeks. The OO analysis and design using the Booch Method [Booch 1994] was introduced in the seminar. In addition to the Booch Method, we also used the examples presented by Whites [Whites 1996] in the seminar.

We worked with the team in the trench after the training. At least for two full days per week, we worked with the programmers in the team to answer their questions and helped them solve any problem they had regarding object-oriented analysis and design (OOAD). A weekly design meeting was held for the team to discuss the OOAD issues relating to the project.

The project failed after four months. We summarize our experience in the failed project as follows:

- Engineers did not have the basics in OO programming which made it very difficult, if not impossible, to understand the OOAD concepts, regardless of the intensity and methods of the training.
- Engineers were preoccupied with the maintenance of the legacy system and never fully committed in time to the OO project.
- Management changed in the middle of the project; the new management doubted the OO technology.

**The Second and the Successful Object-Oriented Project**

Learned from the first and failed OO project, we reassessed our strategy and started our second OO project. Our adjusted plan includes: (1) leading by example, not supervision; (2) embarking on a smaller project; and (3) using engineers who are completely dedicated to the OO project.

The second OO project was successfully. The experience that we learned from the success can be summarized as follows:

- The engineers have the basics in OO programming, thus more receptive to OOAD concepts.
- The engineers were fully committed in time to the OO project with no interference.
- The management was deterministically supportive of the OO project.

**Conclusion**

We would like to summarize our three-year experience in OO education in industry as follows:
• Individual programmers’ OO programming experience is essential in a successful transition to OO technology in a team.
• The OO programming experience cannot be obtained in a short period of time, say a couple of months.
• Half-hearted commitment from the management is doomed to fail the project.
• OOA/D should be introduced with a lead by example method. The people who have OO experience should do the analysis, design, and prototyping to show the team how to do it, rather than relying on the team to do OOA/D and prototyping with the OO experts.
Experiences Gained Teaching Sophomore Students Applied to Advanced Courses

Ana Zanella and Alejandro Fernández

Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS), Brazil and
Universidad Nacional de La Plata (UNLP), Argentina
azanella@inf.pucrs.br and casco@lifia.info.unlp.edu.ar

Abstract

The interest in Object-Oriented Technology has grown a lot in the last years, and many Universities have included in their curricula disciplines covering this paradigm. OOT courses that, some years ago, where taken just by advanced students are now introduced in the sophomore curricula. Nowadays universities have courses covering the entire Object-Oriented Software Development Process.

While some universities, like UNLP in Argentina have been teaching OOT for many years, other universities, like PUCRS in Brazil, are just starting.

Faculties from both entities are interested in using experience gained during the development of courses in UNLP develop new courses in PUCRS. UNLP has courses in OOT for sophomore and courses for advance students and courses at PUCRS are taken just by advance students. There is an important issue to consider: experience with non-oo technology.

One of the problems in teaching Object-Oriented Paradigm is to deal with students experience with non-oo methods (usually the structured paradigm), because it is difficult for them to “forget” their previous knowledge and experience and start thinking in terms of objects.

The intent of this paper is to identify some problems in the teaching students experienced in non-oo methods and to find some aspects that can make these difficulties smaller in the context of the course.

Introduction

The interest in Object Oriented Technology has grown a lot in the last years, and many Universities have included in their curricula disciplines covering this paradigm. OOT courses that, some years ago, where taken just by advanced students are now
introduced in the sophomore curricula. Nowadays universities have courses covering the entire Object Oriented Software Development Process.

While some universities, like UNLP, in Argentina have been teaching OOT for many years, other universities, like PUCRS in Brazil, are just starting. Faculties from both entities are interested in using experience gained during the development of courses in UNLP develop new courses in PUCRS. UNLP has courses in OOT for sophomore and courses for advance students and courses at PUCRS are taken just by advance students. There is an important issue to consider: experience with non-oo technology.

One of the problems in teaching Object Oriented Paradigm (OOP) is to deal with students experience with non-oo methods (usually the structured paradigm), because it’s difficult for them to “forget” their previous knowledge and experience and start thinking in terms of objects.

The intent of this paper is to identify some problems in the teaching students experienced in non-oo methods and to find some aspects that can make these difficulties smaller in the context of the course.

Experiences from PUCRS and UNLP

Experiences from PUCRS

At PUCRS, the development of OO projects as been covered in a third year course on OO design, based in Responsibility-Driven Design methodology (RDD). This course is an extension course, not included in the curriculum of PUCRS Computer Science Graduate Course, but offered to all interested students. There are no courses on OOP in the regular curriculum yet.

The course is attended basically by students from the 3rd year, that have a lot of experience in structured programming and some experience in structured design methodologies. They all have developed some structured projects as final assignments of regular courses in the university.

At the beginning of the course, examples are used to present the main OOP concepts to students, such as objects, message sending, polymorphism, classes and inheritance. The examples are based on real life objects.

After that, students are introduced to guidelines of RDD methodology, in order to give them some tips to start designing small projects. They don’t use any programming language, and we try to base our course in design guidelines to find and define classes, responsibilities and collaborations between objects of those classes.

The problem we face with this course is a consequence of the strong experience in procedural paradigm students have. Students fell comfortable with procedural programming and refuse to think in a new and different way. They fell confused when instead of coding they must start defining the classes, hierarchies and distributing re-
sponsibilities among objects. They would prefer to look for operations system executes instead of partitioning the system in different entities or objects that perform those actions.

**Experiences from UNLP**

At UNLP, in the other hand, the process of teaching OOP is already consolidated, and a significant number of students learn the paradigm in the second year of the Computer Science Undergraduate Course.

There is a full year course on OO programming and design. The first lectures introduce the students to the basic concepts of OO technology using Smalltalk as the programming language. In the first laboratory assignments students learn to program in Smalltalk and to use its environment.

Although there is a strong emphasis in design and that students don’t start coding until the are used to think in terms of objects, Smalltalk is useful to show how objects work.

When students have advanced with the basic concepts of OO, they start with OO Design by exploring design examples of the Smalltalk system hierarchy.

They don’t use any particular methodology, because methodologies are considered too ample to apply to the first design problems. The students are given a set of small patterns, and rules, for them to apply in their first problems. By using these rules the students develop good design skills, a design and programming style, and a taste for design quality.

As we can see, these courses have different contexts and different kinds of students. However the authors believe that the structure of the UNLP courses, that brought real interesting results, could be used as a model for courses at PUCRS, with some little modifications.

**Suggested Curricula**

The most important point in the learning process of the Object Oriented Paradigm for students experienced with non-oo methods is to change the way they see the design of the problem. We must be sure they think in terms of classes, objects and message sending among objects and that they are not only using an OO programming language while programming in the same old style or paradigm.

One approach, to provide an effective learning, could be the following: At the beginning, center the students work in design of OO projects and use Smalltalk as programming language when they acquire knowledge in the Object Oriented Paradigm. Before start coding, the students must be able to think in terms of objects. We must encourage them to work always in groups and to discuss as much as they can during the development of the task.
The increase in project complexity must not be given by an increase in the size but by introducing more complex relations among objects.

The first exercises ask students to assign responsibilities to individual objects they are given, such as Person, Account, and Car. The students have to find the responsibilities each object is able to perform, based in situations of real world. It's important to give the students objects they see and manipulate often in real life and try to use exercises they have never done in non-oo methods before. In this way they won't be influenced by previous non-oo experience.

Then, students start assigning responsibilities to a group of related objects so they can make up a system. They can model a Bank, a Hospital and a University. Once again realities they deal with a lot. At this point, they are asked to find the objects and responsibilities from a description of a problem (or reality), and we can present them some RDD guidelines to help them in this task. It's important to keep the methodology hidden and use only its interesting parts.

In order to avoid bad experiences as a result of early programming, students try out their models following the message passing with their mind or using role playing, to simulate the program. At this point they could use CRC cards to help in this process, with each student being responsible for a set of objects modeled.

During this first stage (the most important) students must be monitored a lot. Success in learning OOP depends on how well they acquire the basic concepts.

Until this point in the suggested curricula students haven’t programmed. After creating some complete object models they are ready to start coding.

It is important to use a pure object oriented programming language, such as Smalltalk. The process of coding must be organized just like the phase they passed by before: start with very simple exercises and increase the complexity when they feel comfortable.

We use the Smalltalk class hierarchy to teach students about topics such as inheritance and software reuse. They can be faced with some existing objects, like Smalltalk collections or magnitudes and we can assign them exercises to deal with those objects.

We believe students experienced in non-oo methods can have an effective learning this way, and after passing those phases, they can be able to work on OO projects of medium size and complexity. They understand the OO concepts, and can use them correctly and have acquired some design and programming style.
Report on FBC II Development

Ping Ning
Hughes Network Systems, Inc, USA
pning@hns.com

Introduction

This is the report on my involvement in the development of the PES second generation file broadcast (FBC II) project. The software has two major components, the hub server and the comm-server. The hub-server runs in a center site to broadcast files. The comm-server runs in a remote site to receive the broadcast files.

Previously, the hub server was running on VAX platform, and had some problems and limitations inherited from the design. The goal of this development is to move the hub server from VAX to Sun Solaris, to redesign the file broadcast protocol to overcome the problems and limitations from the previous software, and support IP multicast.

Object-oriented technology was chosen to improve the development cycle. My goal was to help the FBC II team apply object-oriented technology in the development, help the team members to master object-oriented technology and the related CASE tool, get involved in the whole development cycle to help the product to be successful, and at the same time, get familiar with the products of the company.

The project was a success. It met all the requirements, and was completed and delivered to the customer on time.

Development process

The basic steps in the development cycle include:

- Code review of existing software
- FBC II protocol design
- Subsystem and unit design
- Coding and unit test
- Integration test

I will describe these basic steps based on the hub server development. The comm-server development was a parallel effort and very similar. The hub server is much more complicated than the comm-server. The hub server was developed for Sun Solaris platform. The comm-server was developed for SCO Unix platform.
Code Review of Existing Software

The development team is new to the file broadcast product. We started by reviewing existing code, to get familiar with the functionalities of the existing system, and to identify potential reusable code.

In this step, I was able to use Rational reverse engineering tool to generate class diagrams, specifications, to facilitate understanding of the code.

I also gave demo of the Rational Rose CASE tool to the team members. Mary gave the team members a brief object-oriented training. These activities help the team members get familiar with the object-oriented development cycle and CASE tool. Due to the tight schedule, a five-day object-oriented training course could not be scheduled. Instead, Mary gave a three hour training. It's difficult for the team members to be trained on object-oriented technology in such a short period of time.

FBC II Protocol Design

We redesigned the file broadcast protocol to overcome some of the problems in the old system. The communication between hub server and comm-server, and the message formats were considered.

Subsystem and Unit Design

We identified the major classes and created major event traces. In this step, using the CASE tool was very helpful and should be enforced. We had three members in the hub server team, including myself. I was able to help one team member to use the CASE tool in the design. Another team member seemed to have difficulty in using the tool, and avoid using it as much as possible. We agreed that the design should finally be put in the model using the tool.

This was the most uncomfortable step. In this stage, we struggled between functional decomposition approach and object-oriented approach. There was an urge to decompose the hub server in to functional units, processes, etc., from the very beginning. Also some team members were anxious to start coding right away. When people did not feel comfortable with the design, they tended to blame it on the tool and the approach.

I worked closely with the team members to apply the object-oriented approach. Gradually, we identified most of the classes, and created the major event traces. Once the design gradually took shape, the team became more focus on the design, and we were able to follow the object-oriented approach.

The comm-server design progressed quite smooth. But when I worked for the comm-server design with the comm-server team, the hub server design dragged on without much result, which cause some concerns. Constant close support is required to help a team adopt new technology.
Coding and Unit Test

We divided the coding and unit testing into two phases. The first phase implemented the major functionalities for broadcasting a file, leaving the reconciliation and other functionalities to the second phase. After each phase, we had integration test.

Due to the extremely tight schedule, we did not spend much time in unit testing. Design review and code review were bypassed. All these factors only pushed back the problems and bugs to the integration test.

Integration Test

As mentioned above, we did integration test after each phase of the coding and unit test. However, since we did not spend enough time in code review and unit testing, many problems were found in this step.

Project metrics and analysis

I have collected some project metrics in the following tables. The first one, Project Metrics, gives an idea about the size of the project. The development team were under tremendous schedule pressure, and performed very well, considering most of us were new to the project, the methodology and even the development environment.

One interesting observation is that, per capital productivity in terms of number of lines of source code (SLOC) are almost the same for both hub server team and comm-server team.

Looking at the Hub Server Effort Distribution and Comm-server Effort Distribution, one can see the proportions of effort spent on each step. The hub server is more complex, so the design took much longer time (34%) than the comm-server. One can easily see the rushing to meet schedule by looking at the effort spent on the hub server coding and unit testing (32%), compared to that on the comm-server (44%).

The integration test took more time than it should because a substantial amount of time in integration test was actually used to fixed unit level bugs in the hub server. The time spent in integration test for the comm-server is thus somewhat illusive. Since we tested hub server and comm-server together in the integration test, we assumed the same amount of time was used for comm-server integration test as for hub server. However, when we were fixing the bugs for hub server, we were not doing anything to the comm-server, so the integration time spent on the comm-server should be much less than on the hub server.

<table>
<thead>
<tr>
<th>Project Metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Component</strong></td>
</tr>
<tr>
<td>-----------------------</td>
</tr>
<tr>
<td>Umeå University, Sweden</td>
</tr>
</tbody>
</table>
### Hub-Server Effort Distribution

<table>
<thead>
<tr>
<th>Activity</th>
<th>Days spent</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Old Code review</td>
<td>14</td>
<td>21%</td>
</tr>
<tr>
<td>Protocol design</td>
<td>8</td>
<td>12%</td>
</tr>
<tr>
<td>Subsystem/unit design</td>
<td>18</td>
<td>27%</td>
</tr>
<tr>
<td>Coding and unit test</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Phase I</td>
<td>10</td>
<td>15%</td>
</tr>
<tr>
<td>Phase II</td>
<td>7</td>
<td>10%</td>
</tr>
<tr>
<td>Integration test</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Phase I</td>
<td>10</td>
<td>15%</td>
</tr>
<tr>
<td>Phase II</td>
<td>7</td>
<td>10%</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>5%</td>
</tr>
</tbody>
</table>

### Comm-Server Effort Distribution

<table>
<thead>
<tr>
<th>Activity</th>
<th>Days spent</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Old Code review</td>
<td>14</td>
<td>22%</td>
</tr>
<tr>
<td>Protocol design</td>
<td>8</td>
<td>12%</td>
</tr>
<tr>
<td>Subsystem/unit design</td>
<td>10</td>
<td>16%</td>
</tr>
<tr>
<td>Coding and unit test</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Phase I</td>
<td>15</td>
<td>23%</td>
</tr>
<tr>
<td>Phase II</td>
<td>7</td>
<td>11%</td>
</tr>
<tr>
<td>Integration test</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Phase I</td>
<td>10</td>
<td>16%</td>
</tr>
<tr>
<td>Phase II</td>
<td>7</td>
<td>11%</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>5%</td>
</tr>
</tbody>
</table>

### Summary

The project is a success. We have designed and implemented the the next generation file broadcast software, met the requirements, and delivered the software to the customer on time.

I have achieved my objectives for joining the project. I have helped the file broadcast project to apply object-oriented technology to improve their software development cycle, build better software system, and meet customer requirements and schedule. In terms of transferring object-oriented technology to the product line, I have
helped the team members to follow object-oriented approach. They are familiar with the object-oriented development process. Most of them can use the Rational CASE tool.

**Lessons learned**

Some of the problems I experienced in working with the file broadcast project are discussed below. Although some problems may be specific to this project, we may want to be aware of these issues and try to avoid them in future technology transfer effort.

While working on this product line, the first thing I had to overcome was to gain the trust of the team. I was able to gain trust from both the hub server team and the comm-server team and had good working relationship with them.

There are several reasons that the team or project manager do not trust the help from Software Tech. One is that we are “outside consultant”, meaning we can not be held responsible, thus should not get involved in decision making. We have to work very hard and consciously assume responsibilities in order to gain their trust.

Secondly, whenever there are problems, people tend to blame first on the technologies or tools we try to introduce. On the one hand, we need to be flexible, and be aware of the conflict between the existing environment and culture and the new way we are advancing. On the other hand, the project management should also make clear commitment to the technologies we introduce.

Third, some managers may think that we have our own agenda, and use their project as a test bed, thus see a conflict of interest at times. This demands clear and constant communication between the managers to make sure we are achieving the certain goals of common interest.

The most important thing, in my opinion, is to focus on the product, and try to make the development successful.

From hindsight, I feel we did not have a good project plan, which could identify small iterations. Taking one small iteration at a time and working all way down to coding and testing might help ease some concerns. The first iteration was always the most difficult one, and we seemed to pick up a little too much in one iteration.

More in-depth training on object-oriented technology may also be very helpful. The team members would understand the object-oriented approach better, and thus improve their readiness of accepting the methodologies. There were cases where some team members had hard time following the object-oriented approach and tried to avoid applying the technology or using the CASE tool. If they were better trained, they might be more willing to follow the methodologies and use the tool.
Top 10 Lists Summary

The following is a summary of the original Top 10 lists. These were submitted together with the position papers in advance of the workshop. They appear in no special order and are only slightly edited. This summary contains 37 items only, since some participants did not submit explicit Top 10 lists or lists with fewer items.

1. There is no substitute for experience.

2. In general, developers do not do well to teach themselves OO. Making “the shift” from traditional, procedural design to object-oriented design is difficult. From what I have seen, people tend to take OO concepts and try to fit them into their current, comfortable working model of software development.

3. Understand the role of use cases.
   Objects in use cases are actors as well as specific instances of objects (scenarios). A use case represents a transaction on a particular “instance” of a system (that is, a use case is more concrete than abstract and does not capture alternatives to the specific instance). In our first OO attempt, there was a strong push to treat use cases more like pseudocode or flow-charts thus hindering the ability to find domain abstractions.

4. There should be a focus on modeling the domain during OO analysis. This seems to be a simple concept but you may be surprised at how quickly a team begins discussing implementation and system details such as databases, etc. Consider modeling “system-level” transactions such that assignment of responsibility to objects is deferred. Members of the team may be uncomfortable with the concept of beginning with system-level transactions, followed by successive refinements that eventually assign behaviors to specific objects. This is a key concept of the Catalysis methodology but we largely ignored ensuring responsibilities were assigned such that we designed good domain abstractions.

5. Avoid digging too deep, too quickly. We had difficulty working with abstractions early in design without jumping into long discussions regarding design details (i.e. “... but how is that going to do this...?”). Such details are best left for later design iterations, otherwise this may lead to sacrificing the ability to have good, generalized abstractions in a design.

6. Procedural habits die hard (an OO designer is not built in a day). Understand that a team of experienced procedural programmers on its first OO project has a long road ahead in learning to embrace strong OO principles.

7. Avoid “god objects”.
   Fledgling OO developers may tend to gravitate toward a centralized flow of control.

8. Do not underestimate the power of good encapsulation. First-time OO programmers may have difficulty understanding the concept of “program to an interface, not an implementation”. Don’t be surprised if your team appears to find this con-
cept both unimportant and constraining their ability to add features to the proj-
ject.

9. Beware of grand plans. Be careful committing to providing advanced features like remote debugging and run-time configuration of objects in a C++ program. These present admirable goals but on all but the most liberal development sched-
ules, it is unrealistic to expect to learn OO and, at the same time, produce the “ulti-
mate app.”

10. Get an injection of knowledge. We did not have an experienced OO mentor to help us through the transition away from procedural programming. Despite all having completed well-taught courses in OOA and OOD, classroom training is likely not enough to broaden individual understanding of OO principles and design tech-
niques. The presence of an expert OO consultant could go a long way toward re-
solving the conflicts, uncertainties and misunderstandings that consume long hours in conference rooms. Whether you have a mentor or not, consider seriously documenting design rationales to assist learning and the process of making de-
design decisions.

11. When educating in a class style manner, involve the students as much as possible. Avoid the frontal lecture style whenever possible. This kind of lecture style is also known as structural lecture style, but we want to teach OO ideas, so we have to teach in an OO manner. That means, treat the trainees as self responsible objects.

12. Use analogies and role plays. This belongs also to the OO-teaching style. The more levels of communication you could integrate, the better it is. Remember the chines proverb? Tell me and I’ll forget, show me and I may remember; involve me and I’ll understand. But this is most often ignored, the most educational session use a frontal lecture style manner.

13. Teach the stuff just in time (before starting OOA/D, teach it—not earlier, not lat-
er). Often the trainees run through a curriculum but it takes a year till than can make use of it.

14. Familiarize yourself with the domain, speak the same vocabulary as your tra-
nees.

15. Use mentors, you bigger the project, the closer the time frame, the more mentors.

16. Evaluate the education plan (both from the educator’s and the students’ side) for improving it. You won’t remember it the next time if you do not capture that. And you can only capture it and improve it if you use courseware.

17. Use a behavior driven approach, this prevents the students from thinking about an objects like a database table. This avoids also using associations out of the old world. (Hint: a natural foreign language is learnt the best if you are completely in this foreign environment and you are only allowed/able to speak this foreign language).
18. Factorize the stuff to be taught, for not overwhelming the students. E.g. If you have a complex architecture to explain, break it into pieces and again involve the trainees as much as possible, let them act as part of the architecture.

19. Have a good mix of skills in your team. It does not help if you have a team, which consists only out of UI-experts, or Analysis experts.

20. Use a small project. This has the lowest priority, because it is seldom implementable.

21. Each programmer in the team should be well versed in one of the OO programming languages.

22. Introduction of OOA/D to a team without any OO programming experience is a waste of time.

23. The team should be fully committed to the OO project in their time and effort.

24. The management should be fully committed to the OO project.

25. Change of management and management philosophy in the middle of the project is destined to fail the project.

26. The “let me supervise you to do the OO design” philosophy does not work.

27. OO experts should do the analysis and design and show the team by example.

28. Don’t start coding.

29. Don’t increase project complexity by increasing the size of the assignment but by introducing more advanced concepts.

30. Assignments must be solved in groups.

31. Groups share their experience discussing in a collaborative manner.

32. Be sure students understand how objects work before giving them a complete programming assignment.

33. Use methodologies just as a tool to communicate design solutions. Sometimes it is useful to use an amalgam of methodologies.

34. At the beginning use role playing instead of a programming language. Acting like an object could help understand concepts such as like method lookup, dynamic binding and message passing.

35. When dealing with complete development project (analysis, design, programming) students must submit their design and if it is correct they can go on coding.

36. Use peer reviews to discuss correctness of designs. Students will learn from their peers and when trying to communicate their design.

37. Favor design over programming.
Worškshop Poster

During the workshop sessions we collected more than 40 different recommendations. Discussion time was not sufficient to investigate all of the recommendations in detail and burn them down to one “top ten list” as suggested. However, we selected 20 recommendations for our workshop poster and let OOPSLA’97 participants vote on those. The results of the poster session are summarized in the tables below.

We prepared lists for the trainers’ as well as the trainees’ opinions. Votes were collected separately for industry (column Ind) and academia (column Acad).

During the workshop we did not talk very much about the trainees’ point of view. Our original list of trainee-related topics was therefore very short. To compensate for “bad preparation” we asked OOPSLA participants to add topics to this list. The added topics are marked in the tables below.

Please note that the results are by no means representative!

<table>
<thead>
<tr>
<th>Topic</th>
<th>Votes (total)</th>
<th>Voter from</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Ind</td>
</tr>
<tr>
<td>Get mentoring not only (traditional) training</td>
<td>29</td>
<td>24</td>
</tr>
<tr>
<td>Small teams (about 4–7 persons)</td>
<td>22</td>
<td>19</td>
</tr>
<tr>
<td>Domain experts are important for analysis</td>
<td>21</td>
<td>19</td>
</tr>
<tr>
<td>Use Cases need to be understood</td>
<td>19</td>
<td>15</td>
</tr>
<tr>
<td>Focus on domain analysis during OOA</td>
<td>18</td>
<td>16</td>
</tr>
<tr>
<td>Functional decomposition habits die hard</td>
<td>18</td>
<td>14</td>
</tr>
<tr>
<td>Example, examples, and examples</td>
<td>17</td>
<td>10</td>
</tr>
<tr>
<td>Avoid “god” (not good ;-) objects</td>
<td>16</td>
<td>12</td>
</tr>
<tr>
<td>Favour BFS over DFS</td>
<td>15</td>
<td>12</td>
</tr>
<tr>
<td>Requirements Engineering and OOA are not the same</td>
<td>13</td>
<td>10</td>
</tr>
<tr>
<td>Document (intermediate) results</td>
<td>13</td>
<td>8</td>
</tr>
<tr>
<td>Do not use hybrid languages</td>
<td>12</td>
<td>10</td>
</tr>
<tr>
<td>Favour analysis over design over programming</td>
<td>12</td>
<td>9</td>
</tr>
<tr>
<td>Courseware is important</td>
<td>12</td>
<td>7</td>
</tr>
<tr>
<td>Beware of grand plans</td>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td>Do not use tools too early</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Do just-in-time teaching</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Familiarize with the domain</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>Use a behavior–driven approach</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>“Play” objects</td>
<td>7</td>
<td>7</td>
</tr>
</tbody>
</table>
As a trainee the following topics have been hard to grasp

<table>
<thead>
<tr>
<th>Topic</th>
<th>Votes (total)</th>
<th>Voter from</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Ind</td>
</tr>
<tr>
<td>Polymorphism</td>
<td>13</td>
<td>10</td>
</tr>
<tr>
<td>Subclass versus subtype</td>
<td>13</td>
<td>8</td>
</tr>
<tr>
<td>What is an object anyway? (2)</td>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td>Class versus instance (4)</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>Finding the objects</td>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>Has-a versus is-a (3)</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>Dynamic binding</td>
<td>9</td>
<td>4</td>
</tr>
<tr>
<td>Doing class diagrams (1)</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>Confusing terminology, differs from source to source (5)</td>
<td>7</td>
<td>5</td>
</tr>
</tbody>
</table>

(i) Added by voters in order i.