Informatics in Education, 2012, Vol. 11, No. 1, 29–44

Teaching Web Application Development: A Case Study in a Computer Science Course

Marcos Didonet Del FABRO, Eduardo Cunha de ALMEIDA, Fabiano SLUZARSKI

C3SL labs, Departamento de Informatica, Universidade Federal do Paraná
Rua Cel. Francisco Heráclito dos Santos, 100, 81531-990, Curitiba, PR, Brazil
e-mail: {marcos.ddf,eduardo,fs09}@inf.ufpr.br

Received: March 2012

Abstract. Teaching web development in Computer Science undergraduate courses is a difficult task. Often, there is a gap between the students' experiences and the reality in the industry. As a consequence, the students are not always well-prepared once they get the degree. This gap is due to several reasons, such as the complexity of the assignments, the working environment, the frameworks used and the time-frame constraints. In this paper, we report on a case study on how we taught web application development using extreme tutoring and in an apprenticeship manner. The assumption was to take two real web applications as basis for practical teaching. We present the different issues that we faced: the setup of the development framework, the heterogeneity of human resources and the volatility of the environment. We describe how the process evolved positively. The students became independent, and implemented two applications. We conclude with the lessons learned.

Keywords: teaching software development, extreme tutoring, computer science undergraduate, web frameworks.

1. Introduction

Teaching web application development in undergraduate Computer Science (CS) courses is a difficult task. Often, there is a gap between the students' experiences and the reality in the industry. This is due to three major aspects: (1) the content of the courses, (2) the infrastructure and (3) the environment at the University.

The content is typically divided into theoretical modules and practical assignments, all within a tight time-frame, e.g., 60h at the Federal University of Paraná (UFPR). This is enough time to present the foundations of web development, however, large assignments may not be possible to present. In general, the assignments are chosen based on relative simple applications (e.g., a *Hello Worlds* for web development), which turn out to dissuade the students.

The physical infrastructure limits the number of students to the size of the laboratories. In addition, some labs may be shared all-day long with other lectures. In top-quality (and rich) CS courses, this is a minor issue. However, this is not always the case, specially in developing countries. The logical infrastructure is related to the framework and the

technologies used. Today there is a large variety of development frameworks, application servers, databases and process management tools. However, the curriculum can address only a few of them. In addition, the lecturers do not always have the time to follow the rapid evolution of all these technologies. This hardens the choice of the most appropriate for teaching and for the utilization in the industry.

Finally, the environment of the academia is quite relaxed with respect to companies in general: there is no product to ship, with quality control, interactions with the client, or a supervisor that may influence (in a good or bad way) in the career and wage. All these aspects have the negative consequence that the students are not always well-prepared once they get the degree.

In this paper, we present a case study on teaching web application development at the Department of Informatics of the UFPR. Our experiment had three core assumptions. First, we needed to choose two web applications based on real requirements, because we wanted to create a working and useful product as result. This should act as a motivator for the students. Second, we opted for offering practical apprenticeships instead of "classical" lectures. As such, we had less time constraints and more freedom about the content. Third, the students should participate in all the decisions.

We describe in detail our experiment that has been conducted during 6 months. Initially, we describe how we taught using a tutoring approach, since we drafted students from different levels. Our extreme tutoring approach couples programming/agile methods (Reifer, 2002) and the constraint-driven human resource scheduling method in software development (Xiao *et al.*, 2008). Second, we describe how we chose the development methodology and how we setup the teaching and development environment. We present as well the different project roles and how the teaching was organized. We describe how the process evolved on a positive way. The students acquired a good degree of independence, to implement as a result two working applications. Finally, we conclude with the overall lessons learned.

This paper is organized as follows. Section 2 presents the web applications chosen. Section 3 describe how we conducted the human resources allocation and divided the project roles. Section 4 shows how we choose the development architecture. Section 5 describe the main features of the resulting systems. Section 6 wrap up our experience by presenting the lessons learned and Section 7 concludes.

2. The Web Applications

We searched for two real requirements within the UFPR departments. At the end of our experience we should have two working applications satisfying our clients' needs. We (the tutors) had experience on industrial application development, but we needed to adapt the development process to this different environment. However, we wanted to do only minimal adaptations.

There are different patterns of web applications, such as E-Commerce (e.g., Amazon), web searchers (e.g., Google), Wikis (e.g., Wikipedia), social media (e.g., Facebook)

or enterprise-like applications, based on create/read/update/delete (CRUD) functionalities (Martin, 1983). These patterns are relative to the user interaction. In our case, the goal was to teach MVC-based web development using CRUD applications. We describe the applications chosen below.

2.1. *Application 1 – Chemical Products Stock Management*

Application one aims to control the stock of chemical products. It was requested by the Chemistry Department, because the existing application was using legacy technologies and there was nobody able to maintain it.

The application contains two sets of components: maintenance and listing. The maintenance components implement CRUD operations. The listing components provide different kinds of listings over the existing data. The data model is formed by chemical products and by categories of users that own the products. Only the owners of a product can use it, or the user can cede them to other users.

The implementation request was made in mid-October 2010 and it should be implemented in four months. We negotiated with the Chemical department a four months scholarship for it.

2.2. *Application 2 – Management of the CS Graduate Course*

Application two aims to manage the graduate course of the Computer Science department, because all the management was done using paper forms. For instance, to obtain the transcript of a graduate student, it was necessary to search for all his data in different paper-files (the course has about 150 students).

The application is divided in three set of components: (1) CRUD applications, (2) listings and (3) general consistency checking. The consistency checking are various, for instance, students that fail the exams loose their scholarship. The data model should contain informations about students, professors, scholarships, courses, classes, grades and others.

The implementation request was made by the end of November 2010 and the basic functionalities should be implemented before March 2011, which is when the new school year starts (the school year in Brazil starts in March and it ends in November).

2.3. *Common Issues*

Both systems have a set of common requirements:

- CRUD applications and listings;
- user access control;
- database access.

The initial steps were:

- to define the team and project roles;
- to choose and to set up a common development framework;

- to define the application data model;
- to start implementing.

We present our methodology in the following section.

3. The Methodology

In this section, we describe the teaching format, the project roles and how we adapt agile programming techniques with tutoring time.

3.1. *The Teaching Format*

In Brazilian public academia, human resources (HR) are basically composed by undergraduate students and professors (no engineers were used). The teaching format was conceived based on the time constraints. The first constraint was the time to ship both systems, ranging from six to eight months. In addition, the students have a tight schedule to dedicate to the projects, generally split into lectures, exams, assignments and also the projects. Professors have an even tighter schedule.

We chose to offer non-obligatory and remunerated apprenticeships, instead of traditional lectures and assignments. This has three main implications: (1) the number of participating students is limited; (2) the working time is longer (3 hours/day); (3) the students are recruited and they do not have grades, thus they need to be motivated.

3.2. *Project Roles: The Rookie Challenge*

The roles were divided into project leaders, tutors and developers/architects. The lecturers were the project leaders and tutors. The students were the developers and the architects. After choosing the students, we allocated them in pairs according to their capabilities.

However, in CS courses, it is difficult to recruit experienced students (from 3rd or 4th years), because they prefer the computer science job market that delivers better wages (CNN, 2006). In addition, we had to "compete" for the best students with other academic projects. Research projects tend to attract students of the last year, since they are close to prepare their final project. Therefore, we were bounded mostly to 1st–2nd year students.

One may ask whether 1st–2nd year students have the skills to develop a complete web application upon a tight schedule. Some of these students did not achieve their basic formation yet, such as: some basic algorithms, non-linear data structures, or even object-oriented (OO) programming.

Due to the project requirements, we had to boost their formation with the following initiatives:

- to introduce Object-Oriented Design (e.g., inheritance, encapsulation, polymorphism);
- to teach the Model–View–Controller (MVC) software by handouts;

- to set up regular meetings when improving some feature or deciding complex assignments.

Teaching OO programming was the less complicated task. Sometimes students already reached OO classes and they helped each other along the development. Teaching MVC was harder since important concepts were still missing. For instance, complex data modeling is taught in the middle of the course. Teaching initial project management principles consisted on motivating the students to be organized and independent. Our approach was a tutorial-like and on-demand teaching, mostly with the concepts that the students did not know.

3.3. *Extreme Tutoring*

The extreme tutorial is an adaptation of extreme programming/agile methods (Reifer, 2002) and the constraint-driven human resource scheduling method in software development (Xiao *et al.*, 2008) with an amount of time for tutoring.

In the initial phases of the project, the tasks were simple tutorials, lasting from 1 to 2 weeks. The result should always be a new functionality. For instance, a task could be the generation of a PDF report from some database result set. The tutoring time followed agile programming principles: they were short and periodical. The students felt confident to reproduce the solution step-by-step afterwards. An example of assignment was to study the MVC API and to reproduce a simple application found on the Internet. The objective was to show them how to search for a solution. Once they acquaint the procedure, all the tasks were done following the same rules. However, tutors should not DO the tasks, only HELP students with the their questions and motivate them to search for solutions for themselves.

Tasks were appointed regarding some rules: (i) to assign simple tasks. Moreover, the business rules required the participation of the end users. Students were encouraged to search for the end users and to talk with them; (ii) to avoid or diminish the work-load during the exams week. If some feature was urgent to the system, then it was done by a pair professor/student; (iii) to reallocate pairs of students. The objective was to let all the students to execute any task and to share their experiences to accelerate the development (Henninger, 1996).

Following the sample principle of the tasks assignments, the concepts were taught on-demand. For instance, if students already knew linear data structures, we might teach them collections, such as: lists, queues and sets.

The goal was to motivate the student by quickly seeing a result. Yet, simple tasks were easier to debug by more experienced students or by a tutor. This approach has proven to be worthwhile: in average, after 3 months of work any member of the team was capable to execute any task (even more complex ones, such as a business rule). Furthermore, more complex assignments were always discussed in periodical meetings to make sure the students understood the underlying concepts.

4. Choosing the Infrastructure

We used the Model–View–Controller (MVC) architecture (Krasner and Pope, 1988) to develop the web application. The MVC architecture has three layers, with a good separation between the data model, the visual interface and the interaction between both. The model layer contains the data definitions and the business logic. The view layer contains the user interface, which may be web-based or not. The controller layer contains the code responsible for handling the application flow, i.e., the glue between the model and the view.

The first design choice was to choose an MVC development framework. However, there are dozens of MVC frameworks available, with different set of features. The most common features are:

- usage of patterns for storing the files, i.e., the same kind of files are stored in similar structures and places;
- availability of initial standard components for developing views and models;
- generation of the CRUD code.

Using such frameworks, the CRUD components are developed in very short time. However, the existence of so many MVC frameworks for almost every platform and language revealed to be a problem, because we needed to choose amongst several of them. In addition, we were starting from scratch, without any constraint or preference on the platform, so we could choose between any existing MVC. We cite below just a few of them[1]:

- *Java*: Struts, Struts2, Spring, Spring Roo;
- *Ruby*: Ruby on Rails, Nitro, Ramaze;
- *.Net*: ASP .Net MVC, MonoRail, Spring Framework.Net;
- *PHP*: Agavi, Drupal, Joomla;
- *Python*: Django, Pylons, TurboGears.

To choose the most appropriate framework, we set up 3 meetings with the students and tutors. The meetings also followed the extreme tutoring principle: to assign/do simple tasks, which should be entirely completed in a short time. The tasks here were (1) to do a research on the existing frameworks, (2) to experiment them and (3) to choose the most appropriate one. They had different backgrounds, with experience in distinct development architectures. We did initial assumptions that restrict the choices: everything must be open source, so the .Net branch was not a choice. PHP neither, because of some security issues. The framework must support an open source database, such as PostgreSQL or MySQL derivations, which led to Java, Python and Ruby-based frameworks.

We reviewed the remaining systems platforms. We needed to choose between a rich-enough platform for rapidly developing the applications and that was suitable for teaching as well. However, it was not feasible to test all of them in detail. Based on the size of the community, the documentation found and on the apparent maturity, we narrowed the choice between Spring Roo (Roo, 2011), Ruby on Rails and Django (Django, 2011).

[1] Another list can be found at `http://en.wikipedia.org/wiki/Model-view-controller`.

Table 1
Comparison of web development frameworks

–	Spring Roo	Ruby on Rails	Django
Implementation	Java	Ruby	Python
CRUD generation	graphical and command line (++)	graphical and command line (++)	integrated with code (+)
CRUD appearance	very good (++)	good (+)	user defined (+−)
Database integration	multi database (++)	multi database (++)	multi database (++)
Data model	generated (++)	generated (+)	manual (+−)
Security support	multiple support (++)	multiple support (++)	manual support (+)
Application server	easy (++)	easy (++)	easy (++)
Reporting support	yes (++)	yes (+−)	? (−)
API richness	very rich (++)	limited (+−)	rich (+)
IDE integration	very good (++)	very good (++)	good (+)
Framework	complex (−)	complex (+−)	complex (−)
Documentation	good (+)	good (+)	fair (−)
Testing	generated (++)	generated (++)	manual (+)
HR: current skills	1/4 (+)	1/4 (+)	1/5 (+−)
HR: maintenance	1/4 (permanent)	1/4 (students)	1/5 (permanent)
HR: jobs	several (++)	few (+−)	some (+)

Spring Roo is based on Spring, which is one of the most known and used MVC in Java. Ruby on Rails has increased popularity, it is being used in different projects, and it has enough maturity. Django is Python-based, which has a very rich API and resources. These three frameworks are also chosen based on the experiences of the involved people: we have 2 experts in Java, 1 in Python and 3 in Ruby. The familiarity with the languages and frameworks are important aspects on the choice of the framework, to be able to provide valuable tutoring. Otherwise, it is difficult to technically help the students. For that reason, we may say the choices were not only based on technological issues, but also on practical aspects for teaching.

We defined a set of requirements to compare these three frameworks in more detail. At this point, we went further and created sample applications in all the three frameworks, to simulate the development process. We separated the students and professors in three groups. After that, we set up a competition to create a simple web application and to present in detail their features. The competition was a motivator for the teams to dig into the details of each tool. Table 1 shows these requirements and how it is supported by each platform.

It is important to note that this table is based on our experiments and vision. We do not say that one framework is better than the other, but only which framework is more adequate for our requirements: teaching and rapid development.

We were able to do the same tasks with all the frameworks. However, Spring Roo and Ruby on Rails have a better code generation support. Since we wanted to teach as well how we can benefit from code generation facilities, we restricted our choice to these two frameworks. In addition, another reason for avoiding Python is the lack of experts in our staff.

Spring Roo and Ruby on Rails have a similar set of capabilities. The CRUD generations of both systems are very complete. We can use both tools either using command line, or integrated within Eclipse. Spring Roo produced better-looking layouts. Both have support to different database systems and application servers. The data model (i.e., the database tables) generated by Spring Roo are simpler than the one from Rails. This is because the tables and columns in Roo have exactly the same name as the object model. Ruby make assumptions about plurals, which are not evident at the beginning. In addition, the plurals are English-based, so it is necessary to modify the Object-Relational mapping manually when using non-English concepts. The closer mapping between the objects and the data model of Spring Roo are more easily understood by students.

Both systems have good support for security. The API of Spring is richer than Ruby, because we can use any Java-based API. This is also valid for the report generation system. As first conclusion, the technical aspects of both frameworks are rather equivalent, with a small advantage of Spring Roo due to the richness of API and available components. In contrast, Ruby is simpler to use for simple projects. We think frameworks would be good enough considering only the technical aspects.

The human resources aspect is also important, because the tutoring could be compromised if we started from scratch with a new framework and language. In our case, we had more permanent people with Java skills than with Ruby. This is important for the application maintenance at the long term, since students turnover is quite high. The number of available jobs is also important, since we wanted the students to learn something that they can use in the market. This point advantages Java. For instance, a search on `http://monster.com` returned 1000+ job offers for Java, 488 for Ruby and 789 for Python.

Finally, we chose Spring Roo as our development framework. While we were inclined to choose Ruby on Rails, because it is relatively new, with a good set of features and simple, we chose for "safety" and we took a Java-based framework. The complexity of Spring Roo turned into an advantage to show students that real world projects are difficult and involve several technologies.

5. The Resulting Systems

We started the implementation with the chemistry department stock control. This was the simplest application, thus we used it for the initial tutoring lessons. The application had only 8 model classes, thus the model and the first user interface was generated rapidly. The additional functionalities concentrated the implementation efforts[2].

The MVC framework has several different components. We noticed that the students did not understand everything they were doing in the beginning, because the numerous number of components that interact and that are generated. There were several concepts and technologies that were involved and that required some experience to understand:

[2]The chemistry product control system source code is freely available for download in our public repository: `http://git.c3sl.ufpr.br/gitweb?p=c3sl/cpquimica.git;a=summary`.

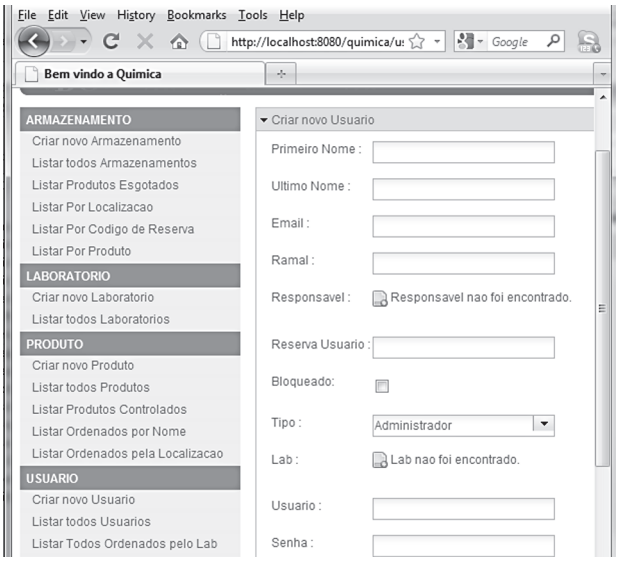

Fig. 1. Chemistry product control system.

JSP, object oriented programming, Java, HTML, Javascript, internationalization, ORM mapping, SQL, relational databases, the MVC framework, configuration files, application servers, version control, etc. This was overwhelming for students that did not have any real development experience. This means that the initial tutoring time was crucial.

The first application was finished in 4 months (1 and a half months more than expected). The students needed approximately two months to be relatively comfortable with the framework. This means that initially the applications were developed slowly. The framework was even contra-productive, because new functionalities could involve new concepts/technologies that needed to be taught. After this stage, there was a remarkable increase in the productivity. A simple new view could be developed in less than one day.

A screen-shot of one application view is illustrated in Figure 1. It has one left menu for each model component. The selected view creates new users (first name, last name, email, phone number, login, password, etc.).

Once the first application was finished, we started the implementation of the second application: management of the graduate course. We defined a first version of the model with 26 classes[3]. This system is still under development. We used a as starting point a data model available in the database book of Navathe(Elmasri andd Navathe, 2000). We adapted this model to our needs.

We followed the same development and extreme tutoring process. Figure 2 shows the listing of courses (this information is public).

[3]The management of the graduate course is also available for download in our repository: *http://git.c3sl.ufpr.br/gitweb?p=c3sl/sapos.git;a=summary*.

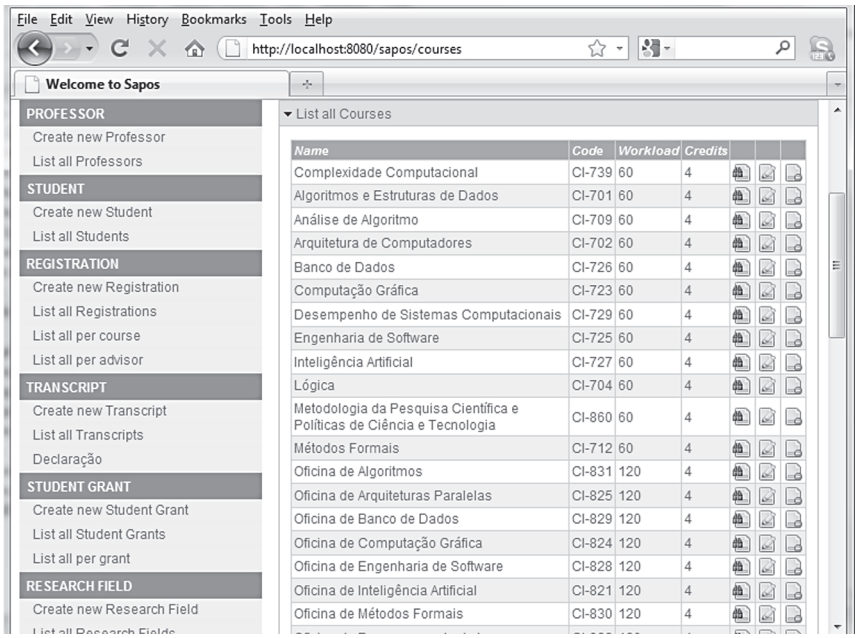

Fig. 2. Courses listing.

The applications had database access, user access control and test interfaces. We found little bugs related to the standard generation, thus it was possible to concentrate on the new functionalities. The choice of a MVC code generation framework was really helpful to develop both applications.

There was a flagrant increase on the productivity from the first system to the second one, because the students already knew the framework and the core concepts when they started the development. We implemented more functionalities in a shorter time. We also diminished the number of periodical meetings and close tutoring, since the tasks were more easily understood. This means the students acquired a good degree of independence. The quality of the developed applications was also very satisfying.

6. Lessons Learned

The teaching and development of both systems using the extreme tutoring approach was a valuable experience that enabled us to learn different lessons about web application teaching. We present these lessons below.

6.1. *Methodology*

The extreme tutoring style coupled with agile programming provides a good balance between programming and teaching. The students should work in pairs, at least initially.

This is because they are at different levels, so they can exchange experiences and learn from each other. This also implies that the learning curve is slightly different. We cannot assume that the same task is developed similarly by two different developers. The tasks must be very clear and concise, with small and independent functionalities developed in short time. The development of large tasks does not work, specially when there are several concepts involved.

It is very important to have periodical code inspection by one experienced developer. This is because students first want to develop something that works, without paying enough attention to coding patterns. For instance, in the initial coding inspections we found components with the method signatures shown in the listing below. The first *createForm* is the standard method to create a form; the second one creates a form and initializes some parameters (this was not acceptable). After some inspections, the students get used to reasonably good development patterns. We need to get increasingly exigent with students. We could see that the first application code could be improved a lot, but it was a good training.

Listing 1. Example of bad method naming

```
public String createForm(
  @PathVariable("id") Long id,
  Model model,
  HttpServletRequest request) {
public String createForm2(
  @PathVariable("id") Long id,
  Model model,
  HttpServletRequest request) {
```

6.2. *Extreme Tutoring*

It is necessary to set up frequent meetings and follow up tasks. The tutoring tasks need to be a complete functionality (even if simple) and preferably with a new concept or technology involved. The concepts were taught on-demand. The tutoring is specially important in the initial phases, when the students do not have autonomy. However, it requires a lot of time from the tutors.

The positive part is that the tutoring time decreases with time. We started working about 3 hours/week, which is a lot considering the tight schedules of the tutors. This time decreased to almost zero at the end of the project, since it becomes a typical development project. Figure 3 shows the number of tasks implemented by a professor and a student per week (**Effort** – Y axis) and its evolution over the weeks (**Week** – X axis). This means that initially, one professor and one student developed 3 tasks per week. The number of tasks implemented by the students increases with time. The turning point is about after 12 weeks: the students get autonomy and "learned how to learn": they continue implementing with little tutoring time. After 6 months the students perform an average of 5 tasks per week.

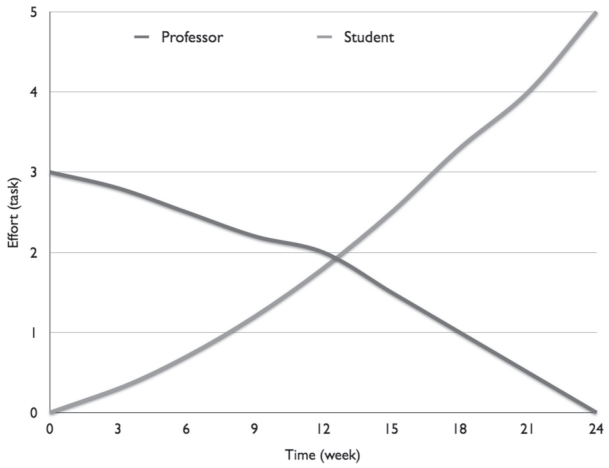

Fig. 3. Professor/students tasks x week.

We give an overview of the developed tasks below. A particular task could be implemented either by a student/couple of students, by a professor. There is no distinction among "who does what".

- implementation of a simple CRUD application (e.g., a "HelloWorld");
- inclusion of database access;
- inclusion of security capabilities;
- generation of the CRUD interfaces;
- implementation of one "finder", i.e., a method that returns a list of elements according to some criteria;
- implementation of a new listing, i.e., a view based on a previously implemented finder;
- implementation of a PDF report.

The weekly meetings must be kept to synchronize and to help with any pending issues. This means the initial part of the apprenticeship should be concentrated to tutoring and learning. The second part is more related to practicing and development. The process evolved from extreme tutoring to extreme programming.

6.3. *Teaching Format*

The use of remunerated apprenticeships instead of lecture-based courses enables to teach in much more details the core aspects of web application development. However, it has two main drawbacks. First, the number of students is limited to the number of scholarships available. Second, it is hard to find students for non-obligatory apprenticeships. This is because the "competition" with CS companies, which offer better wages. We also compete with research projects that generally attract last-year students. It is necessary to be prepared for high turnover of people. There is no guarantee that one student stays

along the entire development. We started the project with 5 students and we finished the initial experiment with 3 students, who were different from the initial group. For this reason, it is important that the students master the full process, from the design until the testing phase. It is difficult to keep a consistent development pace, since students have exams and handouts. Thi s turnover is positive in the sense that "senior" students help to tutor the newly arrived.

The choice of real web applications motivates the students and it helps to finish on schedule. However, the application should not be overly complex, in the sense it could not be delivered in 4 to 6 months, because the result would not be visible at the end of the apprenticeship.

6.4. *Framework*

The choice of the framework is a difficult task, because there are dozens of frameworks available. One MVC frame-work is not better than the other, thus the choice is based on a compromise of technical capabilities and also practical issues. The expertise of the tutor is very important. We have chosen a quite complex framework. While initially it is more difficult, it prepares the students to go more easily from a complex framework to a simple one. The goal is to demonstrate that the technology is just a facilitator and that the related concepts are the most important to know (OO, MVC, agile programming, ORM, etc.). The complexity of the framework gives a vision that application development is complex, with dozens of API's, databases, providers, functionalities. This difficulty reproduces a non-familiar environment that is often found in companies when they will start working.

Setting up a competition to choose the framework is a positive activity because the students want to win. Thus they are motivated to create sample applications and to explore the frameworks. Some go farther than expected by showing hard coding, so they can better "sell" their framework.

The evaluation and choice of frameworks is not only driven by the technical details. The existing knowledge of the permanent team about the language and related frameworks plays an important role. Even if this is a key issue for teaching the concepts, choosing the framework is important when the systems need to be maintained on the long term. This also gives more responsibility on the choice of the technology.

We spent a reasonable amount of time to learn the frameworks and to feel relatively comfortable with them. This is mainly due to the lack of experience of students with complex application development. In addition, many of the concepts were not yet learned in the course. To address this issue, the initiatives we took to boost their formation were worthwhile. However, we should avoid ad-hoc teaching, i.e., teaching particular concepts without giving a global view.

6.5. *Follow-Up*

The students follow-up is important to evaluate if the method was worthwhile from them. Unfortunately, it is difficult to gather information from students after the end of

the project. We needed to informally find out what they were doing. From the 7 students that participated, 1 student is now doing an internship in application development in a company; 2 students are doing apprenticeships in application development in other projects; 2 students continue in the CS course without no particular internship or task; we lost contact with 1 student; finally, 1 student is still working at the project, improving the application of the CS graduate course. This means 4 out of 7 continue in the application development domain. However, we cannot conclude if the experience influenced on their choices or not.

To summarize, teaching web application development using apprenticeships was a worthwhile experience. On one hand, we had to spend considerable time in the beginning with tutoring. On the other hand, we clearly saw the improvement of students, which gradually learned (and are still learning) how to develop simple to complex applications. We used up-to-date development practices and technologies, with hard deadlines. This enables to better prepare students to application development.

7. Conclusions

In this paper, we reported on a case study of teaching web application development on a CS undergraduate course. We described the main issues we had and how the process evolved to ship two working applications. The main success factor was to teach in an apprenticeship manner, coupled with extreme tutoring at the initial phase of the project, instead of typical theoretical/practical lectures. During this experience, we faced different issues. First, the choice of the development framework is not straightforward, because there is a very large set of available implementations. Second, the turnover of students is high compared with typical lectures, so we needed to handle project changes frequently. Third, our extreme tutoring approach that couples programming/agile methods (Reifer, 2002) and the constraint-driven human resource scheduling method in software development (Xiao *et al.*, 2008) has proven to be effective.

We initially spent a large amount of tutoring time, but this time decreased considerably along the months towards a weekly follow-up meeting. The result could be seen in the good level of autonomy acquired by the students. Fourth, we could not simulate an industrial environment, due to the differences in the physical/logical environment and due to the educational character of the project. However, we could see the positive evolution of the students in terms of maturity, which probably would not be acquired with typical lectures. The implementation of real applications was a major factor of motivation: they are both running today with their code available for download in a public repository.

Acknowledgments. We would like to thank the students that participated in the implementation, the team from the C3SL (`http://www.c3sl.ufpr.br/`) labs that helped to choose the framework and the CS graduate course and the Chemistry course for providing the case studies.

References

CNN (2006). *What some fastest-growing jobs pay*. CNN Report.
`http://edition.cnn.com/2006/US/Careers/01/26/cb.top.jobs.pay/index.html`.

Django (2011). *Django project*.
`http://www.djangoproject.com/. Django Community`.

Elmasri, R., Navathe, S.B. (2000). *Fundamentals of Database Systems*, 3rd edn. Addison-Wesley-Longman.

Henninger, S. (1996). Accelerating the successful reuse of problem solving knowledge through the domain lifecycle. In: *Fourth International Conference on Software Reuse*, 124–133.

Krasner, G., Pope, S. (1988). A cookbook for using the model-view controller user interface paradigm in smalltalk-80. *J. Object Oriented Program.*, 1, 26–49.

Martin, J. (1983). *Managing the Data Base Environment*, 1st edn. Prentice Hall PTR, Upper Saddle River, NJ, USA.

Reifer, D.J. (2002). How to get the most out of extreme programming/agile methods. In: *XP/Agile Universe*, 185–196.

Roo (2011). *Spring Roo*. Springsource community.
`http://www.springsource.org/roo`.

Xiao, J., Wang, Q., Li, M., Yang, Y., Zhang, F., Xie, L. (2008). A constraint-driven human resource scheduling method in software development and maintenance process. *ICSM*, 17–26.

M.D.D. Fabro is assistant professor at Federal University of Paraná. He worked as researcher at IBM Software Group (France), on the integration of business rules and model driven engineering. He did a post-doc at ILOG. He received his PhD in computer science from the University of Nantes in 2007. His current research is about model driven engineering applied to business rules management systems, software development and data integration. He has been responsible for the Eclipse/GMT component AMW (ATLAS Model Weaver), and also a contributor to AM3 and ATL components. He worked for 7 years as a software developer in Brazil.

E.C. de Almeida received his computer science PhD, with high honors (félicitations du jury), from the University of Nantes (2009), France, supervised by Patrick Valduriez and Gerson Sunyé (LINA ATLAS-GDDTeam). He received his computer science MSc from the Federal University of Paraná (2004), Brazil, and his computer science BS from the UP (1999), Brazil. From 1998 to 2005 he worked as an engineer in database technology at HSBC Bank, GVT Telecom, and UFPR Foundation. He is currently an assistant professor at the Federal University of Paraná where he have also been deputy head of the graduate program in computer science since 2010.

F. Sluzarski is an undergraduate student of computer science at the Federal University of Paraná (UFPR). He is specialist on developing web applications and he is currently the responsible for maintaining the web application of the CS graduate course at UFPR.

Internetinių svetainių kūrimo mokymas: atvejo analizė informatikos kurse

Marcos Didonet Del FABRO, Eduardo Cunha de ALMEIDA, Fabiano SLUZARSKI

Internetinių svetainių kūrimo mokymas informatikos paskutinio kurso studentams yra sudėtingas uždavinys. Dažnai egzistuoja atotrūkis tarp studentų patirties ir tikrovės pramonėje. Dėl to, studentai yra ne visada gerai pasirengę, kai jie įgyja bakalauro laipsnį. Šis skirtumas yra dėl kelių priežasčių, pavyzdžiui, dėl užduočių sudėtingumo, darbo aplinkos, naudojamų sistemų ir laikotarpių apribojimų. Straipsnyje autoriai aprašo atvejo analizę, kaip mokė internetinių svetainių kūrimo, naudodami ypač didelį kuravimą ir gamybinės praktikos metodą. Buvo paimtos dvi realios interneto programos kaip pagrindas praktiniam mokymui. Autoriai pateikia skirtingas svarstomas problemas, su kuriomis jie susidūrė: tobulinimo struktūros sąranka, žmogaus išteklių įvairiarūšiškumas ir aplinkos kintamumas. Aprašomas visas procesas kaip jis vystėsi teigiama linkme. Studentai tapo nepriklausomi ir įgyvendino dvi programas. Autoriai pasidalija sukaupta patirtimi.