Software Engineering for Undergraduates

Nenad Stankovic
University of Aizu
Aizu-Wakamatsu-shi
Fukushima, Japan
+81 242 37 2559
nenado@msn.com

ABSTRACT
Software engineering has evolved, over a short period of time, into a dominant and omnipresent industry. In education we have recognized the importance of both managerial and technical aspects, but often failed to organize them in a coherent course with a relevant, if not realistic laboratory project. The problem is far-reaching, and should be dealt with accordingly. This paper presents our before and after findings, and elaborates on CPE 207, the new Software Engineering course that, in our opinion, helps in bridging the gap between university and industry.

Categories and Subject Descriptors
K.3.2 [**Computers and Education**]: Computers and Information Science Education – *Computer science education*

General Terms
Management, Documentation, Design, Human Factors

Keywords
Software engineering, education, project course, object oriented

1. INTRODUCTION
Over time much has been written on how to organize a course on software engineering (SWE), with arguments flowing for and against additional complexity (e.g. [2], [6]). We believe that SWE is a, if not the, umbrella course in computing. Unfortunately, this may still not be widely recognized, both in the industry and in academia. The modern course on SWE requires that the student is familiar with many aspects of computing (e.g. networking, Internet technologies, programming languages, databases, HCI, computer graphics, and project management). Depending on the year of academic study these expectations could be unrealistic, especially if that is a 2nd year course.

The theoretical component of a SWE course should be structured and presented such that the student eventually gets comprehensive understanding of the discipline, with technical and nontechnical topics logically and causally interwoven. The practical component (i.e. the SWE laboratory project) should refrain from unnatural simplifications of roles and work products. The student should experience that at times solutions are to be found, not prescribed by the book.

ICSE'06, May 20–28, 2006, Shanghai, China.

The uneven distribution of knowledge or even lack of it must not become an obstacle to the student. However, we should not scale down the curriculum but incorporate, if and when necessary, additional elements that will allow all students to make a uniform progress. SWE is a complex discipline that primarily aims at answering the question of how to build SW systems. But, we must also decide what tools to use, learn how to select tools and technologies, and understand their character and impact.

In this paper, we elaborate on these ideas. The past experience at Nanyang Technological University (NTU) in Singapore motivates our approach and decisions. Our background from IT industry also taught us about the difficulties that graduate students face when joining workforce. We inform on CPE 207, the new course on SWE at NTU in which we have put our findings and ideas into action. We also report on the outcome of the laboratory projects.

2. PAST EXPERIENCE
The past experience at NTU has shown that the expectations on student competence were not explicitly defined in the course material, or taken into account in the course structure. However, they came into light in the laboratory project. But that omission or unawareness thereof had far-reaching effects on the course itself. The lectures were focused on a small number of topics in project management, diagrams for SW design (both OO and structured), testing, QA, and HCI. They failed to provide a comprehensive yet approachable portrayal of SWE as a discipline, or any of those topics in particular.

As a result, the SWE curriculum became a collection of unrelated pieces of information on procedures and techniques. The outcome could also be attributed to a fact that the course was based on the lecture notes from various sources, not on a single textbook. For example, the COCOMO model was introduced, but it was not explained how to estimate KLOC. As a result, the best answer we could get from the student was *Sir, I did something similar before*. Frequently we were asked whether *to round the number of man-months up or down?*

Interestingly, many students demonstrated strong technical skills in their laboratory projects. They were competent in programming languages for online applications and services to support them (e.g. PHP, HTML, RDBMS, 3-tier, client-server), none of which had been addressed before or during that course. They would form teams of 6 to 8, and take on a role each (e.g. PM, customer manager, designer, developer, and tester). They would decide on and build a simple online shopping, ticketing, or matchmaking application supported by a RDBMS, and MySQL [3] used to be their preferred choice.

In most cases, however, the system had little in common with the design and planning. The other work products appeared sketchy, as well. The student became primarily focused on the artifact,

rather than the process that led her to that phase. Even more problematic was that laboratory supervisors could provide little quality input. They had little control over such projects, because they were not familiar with its true nature, the means, or intended outcome.

3. CPE 207

For those with industrial background these findings might appear familiar. SWE has been evolving heavily dependent on practical experience, systematic reflection, and constant improvement. Even in a university environment, it requires strong familiarity with above mentioned prerequisites. Nowadays, to build a SW system requires teams of engineers, good collaboration, and understanding by all involved, together with timely exchange of information that are detailed, correct, and precise. The student should learn that to estimate effort requires understanding of what has to be done, how it is going to be done, and what challenges can be foretold based on current knowledge.

It became our belief that the skills and motivation of our students should be challenged and employed on a project that is open, yet formal and controllable. It should reflect what modern SWE really is, given that university laboratory is not a professional institution it mimics and students are not employees. After all, SWE is a natural process that is logical and understandable, albeit proven difficult to apply in practice [3]. As a result, CPE 207 follows these ideas and object-oriented approach. CPE 207 is a mandatory 2nd year course, attended by almost 450 students over a period of 1 semester.

Many authors dealing with this topic qualify the attributes *real world*, *large scale*, and *complex* as desirable, when describing the laboratory project. We believe, however, that those should be understood and approached such that the project simply servers and supports the modern SWE curriculum. Thus, in our approach *large* means rich enough in features to engage a laboratory session of about 80 students, *real world* means requirements that are familiar and easy to understand, and *complex* means capable of incorporating many if not all (important) elements from the SWE theory. Anything less than that is inadequate, anything more than that is unnecessary. The problem then becomes how to find a balance between what is readily available and what should be applied or discovered by the student. If the project is too simple its importance might be missed and if too hard it may demotivate or result in errors that cannot be rectified in time.

This kind of course is anything but mechanical. Lecturers and laboratory supervisors must keep an open mind and eyes for the whole duration. Having established that laboratory should serve lectures, we have also learned the opposite is true. Although our past experience demonstrated what our then students were capable of, it became apparent that this time they needed additional direct help to overcome the difficulties related to implementation and technology. This laboratory must not turn into a programming exercise, but without it the whole enterprise would lack purpose and adequate verification. We organized, therefore, two laboratory briefings (not planned for) on RMI, HTML, Servlets, JSP, Tomcat, and MySQL. The programming work then became simply to modify and incorporate those examples into code. Further, those briefings helped the student in concentrating on other relevant issues of planning, analysis, design, and testing, etc. The final mark took into consideration all those work products, including meeting minutes, release notes, and status reports as important.

3.1 Course Textbook

We believe that having a textbook benefit our students, not only to those who would like to learn more, by providing more details on each topic and pointers to other references. It also helps the assisting staff in preparation and better understanding of course objectives.

The course text book has been Object-Oriented Software Engineering Using UML, Patterns, and Java by Bruegge and Dutoit [1].

The strengths of the textbook can be summarized as:

- Broad, even innovative range of topics that deal equally well with technical and nontechnical aspects of SW development, while not overloading the reader with details of a lesser importance.
- Consistent graphical presentation based on UML. The student can easily understand that different concepts can be presented in the same (UML) notation.
- Encouragement to undertake our own real world, large project at University. The book provides many examples that can be used for self study, when not possible to address the problem in a lecture.

We also encountered problems, some of which were unexpected:

- This edition is riddled with (not only) typographical errors that must be constantly point at, and explained.
- Some topics are described rather sketchy that leaves many questions unanswered (e.g. Chapter 14, and 15 in particular). We find Chapter 12 valuable, but vague.
- Sometimes, the narrative does not quite match with the figures (e.g. MVC), and some figures are incomplete.
- Although related, MVC solves a different problem than Client-Server, or Repository. They are better not described together, even when clearly distinguished. This became a source of confusion that we failed to recognize until after the examination time.
- The consistency in presentation yielded unwanted effects, too. For example, PERT charts are neither UML activity nor statechart diagrams, as many students came to believe.
- Information is sometimes scattered around, which makes it easy to miss an important concept.
- It would be useful to summarize and describe all the UML symbols in one place.

Finally, the textbook elaborates extensively on Arena system, but we often found the presentation as a distraction. It could be better presented as a case study in one piece, in an appendix.

3.2 Laboratory Project

Due to their size, complexity, and time constraints, many SW projects employ teams of professionals who must also have, or develop organizational and communication skills, and become team players. Technical personnel must know how to make estimates, document their work, manage their time, and work

under little or no supervision. When organizing a SWE course, it is expected that the student can apply acquired theoretical knowledge in a laboratory environment and through interaction with fellow students who work on the same project. This expectation does not depend on the project size, being either a large or a toy project. Our goals can be summarized as follows:

- We educate future SW engineers with a broad set of skills and knowledge, not experts with particular or highly specialized skills.
- SWE concepts and decisions are often interdependent. For better understanding, they require nonfragmented application. Practical examples represent elements of a comprehensive solution to a problem.
- Organizational and project management concepts can be practiced within a model that is realistic yet easy to understand, and effective.
- In SW projects it is equally important to produce standardized documents, as it is to interact with and serve other participants.
- While working on a project that is relevant our students acquire experience that will be of value past graduation day.

It has been impossible to predict the exact number of students in the course, and even more so per laboratory session. Therefore, the project must be technically and organizationally sound and flexible to accommodate any number of students greater than 62 (i.e. the typical allocation for the past SWE laboratory). We need a project framework in which students are encouraged to employ their creativity, and come up with original elements. The 450 students were organized in 7 laboratory sessions, 6 projects, and each project employing 65 to 110. The 110 project was divided in 2 sessions, scheduled on 2 different days. Having less than 60 students per project would make it too demanding to complete.

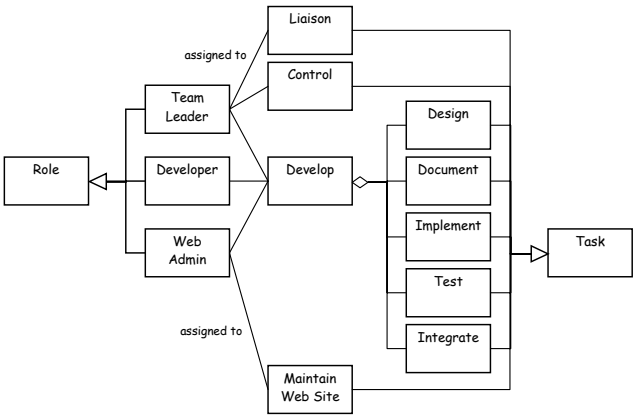

Figure 1 Student Roles and Tasks

Organizationally, we opted for a simple, flat structure (Figure 1) and a nonlinear information flow, in which each student gets involved in all aspects of the project. This serves our educational goals, and is easily accessible to the student and her level of experience. Each of the three roles engages in development work. A provision was made for project managers (i.e. laboratory supervisors) to control the team indirectly via a team leader. Team leaders are responsible for liaising with other teams. However, students are encouraged to interact across team boundaries, rather than concentrate only on their own subsystem.

The laboratory project name is Public Bus Simulator Systems (PBSS). PBSS appeared as the project that can meet our goals because everyone is familiar with buses, public transportation, and traffic regulation. That should make the basic requirements easy to understand and refine. We expected that each student would produce one or a couple of related classes, accompanied with documents. That was the work package. We mandated what documents must be produced by the team and by the student, but not the content. Not all diagrams are applicable in all situations, and not all work packages require coding and testing. We have also provided the templates for each document type (e.g. meeting minutes, release note, problem note, status report, system design, object design, test plan). The major steps in the process are:

- Form teams – students got a briefing in advance on the laboratory project and its important aspects.
- Understand the system architecture (Figure 2) and your subsystem. Refine the initial requirements.
- Apportion the work and select a work package – based on individual preferences and team consensus.
- Apply SWE activities including estimation and planning – as explained in lectures, textbook, and laboratory manual.

Students first familiarize themselves with the requirements and the system as presented, and then pick their work packages they would carry throughout the project. They should make their own estimates, and combine those until the project schedule gets completed. The whole process, including all phases, was as described in the course textbook. Further, to help with estimation and planning, the laboratory manual provided concrete examples that complement the lecture material. An attempt was made to implement a RUP [4] model with 3 iterations, mainly to generate a momentum initially as to speed up the process. Also, we wanted to avoid the situation when most work is done and knowledge acquired towards the end of the semester. If first iteration can be completed in 8 weeks, then the remaining 4 weeks can be used to polish up the work products and prepare for the exam.

The problem here, that we were aware of at the time of planning for the course, is not only in the complexity of the project, but also in the nature of the educational process itself. When we teach we follow a waterfall model and lectures run for a whole semester. The question then becomes is there enough information presented and knowledge readily available to the student when needed. The answer is no; however, it is beneficial if the student, at times, explore and forward learn on his own. These made them more proactive in seeking help and answers from the staff. For example, estimating effort and planning on a SW project still represents a major problem in IT industry, and even more so here because they do not understand the lifecycle yet. The textbook addresses these topics in final chapters, while they have to be addressed early in a SW lifecycle. Laboratory manual, as a concise and targeted document, can assist rather effectively in resolving these shortcomings, and we took advantage of that.

We have also noticed that our students have spent much time in negotiating and contemplating requirements, without paying

attention to how much would all those fancy features *cost* and, sometimes, even their feasibility. But this is not all bad by itself, because it also reflects the excitement about the project. Also, some requirements may appear harder to understand or document. It takes time to gain confidence in applying a methodology and start communicating freely. A departure from the process model or project plan does not mean that that time is wasted. A simple prototype for better understanding is helpful and stimulating. For example, one team started their work by reusing code from another course laboratory. This propelled their efforts far ahead because they understood the task ahead. Instead of struggling with a bare minimum they contemplated advance graphics and features. The momentum extended to their documents that were complete and among the best we examined.

To build the PBSS, we decided on the following technologies:

- Java 1.4 – we chose Java because it provides all the elements necessary to build this system within a single programming environment.
- JSP – still Java, but adapted to Web pages.
- HTML – we only need the most basic features to present database tables and implement access control.
- RMI – does not break the notion of object and locality in interprocess communication.
- Java 2D – simple yet provides fancy features for those more ambitious.
- Tomcat 4.x – supports servlets, and in public domain.
- MySQL 4.1, Administrator 1.1, Query Browser, Connector/J3.1 – in public domain, with good online help and automation.
- Visual Paradigm Standard Edition – to create UML diagrams.
- MS Word for documentation.

4. Public Bus Simulator

For this laboratory project we defined the software architecture, and proposed the initial requirements that each team and session has elaborated upon. This way, they were given a realistic, fair, and targeted framework to work within, but each project has defined and developed a unique PBSS.

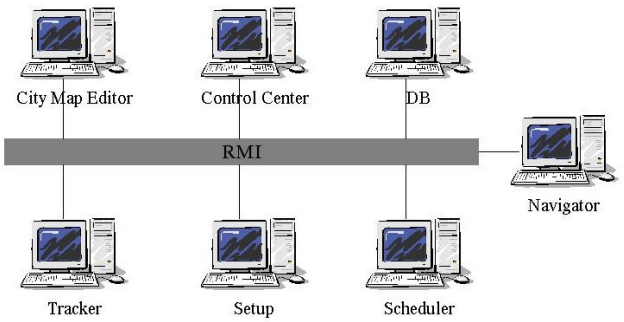

Figure 2. PBSS

The PBSS was conceived as a distributed system that consists of 7 subsystems (Figure 2). Each subsystem runs on a dedicated PC. They exchange information and issue requests by making RMI calls. The subsystems in Figure 2 and their roles are summarized as follows:

- City Map Editor – enables interactive definition of a 2D city map with streets, traffic signals, buses, bus stops and routes, passengers, roadblocks, bridges, etc.
- Control Center – is the central registry for subsystems and acts as forward request facilitator.
- Database – stores city maps and other persistent information. It consists of a Java RMI frontend and a MySQL database that communicate via SQL/JDBC.
- Navigator – directs each buses along its routes.
- Scheduler – is the simulator engine that schedules each bus to its timetable, manages passengers, and operates traffic signals.
- Setup – provides web pages to initialize and configure the PBSS at runtime.
- Tracker – displays the city map with dynamic updates as buses and passengers move in time, and traffic signals change state.

In addition, we also defined the:

- Web Site – is not part of the PBSS. It provides a repository for work products (only soft copies are accepted), and a project forum. Stored work products can be accessed online, by project members only. We defined the features that must be supported for us to mark work products

A fully distributed solution like this one creates technical problems that must be assessed against a monolithic solution. The additional complexity comes from the RMI interfaces. Here, we are not concerned about performance due to networking. On the other hand, good design decisions lead to a simple, scholastic solution and fast implementation, benefits of which we want to exploit. For example:

- Clean and firm subsystem boundaries that promote understanding of and applying the important concepts of strong cohesion and weak coupling. We aim at a simple and elegant solution, with code sharing and no redundancy.
- More freedom to explore and lesser dependency or impact on other subsystems, and firm understanding of roles (i.e. who is my client and who am I client to).
- The student can learn that subsystem may act as client or server, or both, by requesting and providing services. This became important technically when implementing the RMI interface for each role within one process.
- The student can learn that the impact of architectural styles is not local, as opposed to design patterns that are concerned with low(er) level elements and details.
- How are events generated, and propagated throughout the PBSS? Who is interested in those events?
- The notion of what represents the common system wide knowledge and concepts, as opposed to specialized (i.e. subsystem specific) representations and assumptions. For example, what does Bus represent to Database, or to

Scheduler, etc.? Who should define the system wide class Bus? What are benefits?

- The database frontend enables uniformity in inter-subsystem communication. It directly implements the example of reducing the coupling of subsystems from the textbook (Figure 6-5 [1]).
- The usage of a central repository not only at system level, but also in the subsystem detailed design.
- Where the concurrency fans out, how it propagates, through the system, and where it fans in? Does it get affected by the distributed architecture?

In a monolithic solution, the concepts presented in lectures may not be easily recognizable or enforced, because everything that is known and functionally supported is readily available.

The laboratory manual has 34 single spaced A4 pages, including figures. In addition, 63 slides accompanied the two outstanding laboratory briefings. As mentioned before, the manual not only describes the PBSS, but it also provides information on project organization, roles, planning, estimating, and on controlled and incremental development process. It complements the textbook by providing missing information and specifically targeted examples. For example, it provides hints and suggests a generic approach to specific design problems in graphical editors. It explains the important sequences of interaction between the subsystems, but that information is deliberately scattered around. The student is expected to follow those leads and work on the missing elements.

The laboratory manual is written such that each student, irrespective of the team he or she joined, must read through the whole document to learn and understand not only about the subsystem that the team is responsible for but about whole PBSS. Whether this appeared unexpected or not, it produced the result we hoped for. For example, roadblocks are mentioned in the Scheduler section, even though Scheduler is not concerned about roadblocks. The idea here is to make them understand what strong cohesion means in practice and about inconsistent requirements. They understood that Scheduler does not make use of a city map (in which roadblocks are defined) for its task. However, Navigator teams on all 6 projects demanded that the Scheduler team should be responsible for avoiding roadblocks. (Or could this be a case of less work over sound engineering?) In each case, laboratory supervisors had to explain what design decision to make.

Table 1 Team Size Based on 80 Students per Session

Team	Size	Team	Size
City Map Editor	18	Scheduler	6
Control Center	7	Setup	9
Database	9	Tracker	10
Navigator	15	Web	6

In our planning, we assumed 80 students per laboratory session (i.e. project), which is our future target. Thus, we estimated the size of each team as presented in Table 1. As a rule, it is easier to make a large team even larger, than making a small team smaller in size. Therefore, when cutting down the team size due to a lesser number of students, we cut more aggressively the large teams. No team had fewer than 6 members. In particular, the Scheduler team could be smaller, because there are no additional features that can be contemplated or worked on or, for that matter, features that can be removed. However, smaller teams run these risks:

- One student can do all the work. Rather, those with the skills should help in moving the project forward.
- We do not know, and cannot predict how many students may not be initially up to their task.
- We cannot predict if one or more students will leave, which can overload the remaining team members.
- Small teams do not necessarily reflect the team work aspects that we expect.
- Small teams do not necessarily perform better, because they might not be formed purely on the basis of interest in the subsystem.

Still, the Web team was deliberately small in size, even though their requirements could be easily made more complex. We hoped that a small team would be more efficient, and responsible. This was an independent but nevertheless a critical component, as explained above. Technically, the work they are doing is similar to the DB and Setup work combined together (i.e. Tomcat, JSP, MySQL). In addition, their requirements are only marginally affected by the other teams. This is to the extent that they consult other teams as to establish what their preferences would be. On the other hand, the Scheduler team realized how limited the scope was so they were not open to taking more members. Large teams were encouraged to form subteams based on the closeness of their work packages, and pick a subteam leader. This recommendation was followed easily and without exception.

5. Experience

Students are a motley group of would be professionals that differentiate vastly in knowledge and motivation. Selecting right people for the job is a difficult task for every PM. In university environments we have to accept resources without knowing their background, and appreciate first and foremost their individual preferences. Thus, each student could decide which team to join. The PM has intervened only when there were too many students on the team, by persuading some of them to join a team that was understaffed. Again, this was on amicable grounds because each student must be interested in the subsystem and comfortable with the work.

The mechanisms of control are limited at best, and deadline is firm. However, we did not find these projects difficult to manage. For each team and project it was important to find a balance between individual ambition or lack of it, and learn to cope with team inertia as to avoid conflicts among students. We took two different approaches here:

- One, in which PM (i.e. laboratory supervisor) appoints team leaders, and drives the process by providing guidance to the team leaders on a regular basis. The team leaders were appointed by the PM based on her assessment of who within that laboratory session of students has the technical skills or prior experience, and desire to lead a team.
- And the other, that is primarily concerned with risk management, and let students appoint team leaders. The

idea here is that students know each other, and who they have confidence in. PM provides guidance upon request or when deemed necessary. It is expected that lectures and tutorials provide enough information for the student to perform her work.

In retrospect, the 1st approach appeared as generating momentum initially, but soon the amount of activity at the top and at the bottom became disproportionate. It created a two-tier organization which is not desirable as students must understand that they are equally responsible for the outcome. The 2nd approach detected more problems and provided more direct insight as most students were preoccupied with the technical issues. Working with a whole team rather than only team leaders served them better and answered more questions. The word working means participating in but not driving the meeting or process in general. The online forums proved invaluable, being active late at night, even during the one week midterm break. They made problems and ideas public and let us interact with our students at any time, even while away from University. These findings, however, do not negate the importance of PM or team leader as motivator or driver.

Was RUP practical in this environment? It is difficult to answer that question with a simple yes or no. Some teams succeeded in doing so, i.e. those who understood the what and the how. For other teams, the complexity and interdependence was too high, and they defaulted to waterfall. But some also wanted to design and build a better subsystem and spent more time on rework and exploring. We do not regard that as a critical problem, though. Students should be confident and comfortable with their progress. Racing against time should not be their first priority because they are only learning how to document ideas, design software, and work as a project team. Therefore, we also permitted resubmission of work products beyond the deadline. Students used that option frequently, to get a better mark, without abusing it.

Table 2 Outcome (C=completed, I=integrated, N=not, P=partially, 3=Java 3D)

Session	CME	CC	DB	N	SCH	SU	T	Web
1		I		I	I	I		C
2	I		I				I–3	C
3	I	I	NI	I	I	NI	NI	C
4	NI	I	I	NI	I	I	I	C
5	I	I	I	I	I	I	I–3	C
6	I	I	NI	PI	I	I	I	C
7	I	I	I	NI	PI	I	NC	C

Eventually, the outcome was as summarized in Table 2. We marked all Web subsystems as completed, because they do not integrate with the PBSS. We also got 2 mini systems, because 2 Scheduler teams from Session 6 and 7 decided to build the subsystems to which they interfaced (i.e. the DB, Navigator, and Tracker). They simply enhanced the code they used in testing. Is the outcome as expected? For us teachers, the answer is unfortunately not as simple as completed project, and our aim was not to make the project trivial. This project course requires dedication from the student and that is reflected in the high ratio of completion per project. These lead us to conclude that the framework was balanced and effective and the project interesting and well accepted. The most difficult subsystems turned out most complete and team size did not become a negative factor.

Overall, the students demonstrated good time management skills but some had difficulties to take into consideration dynamics of the project as a whole. Working and sharing on such a large scale was beyond their experience, as it was new to NTU. Among the 49 teams only 2 can be described as problematic. One of them lacked motivation, and the other struggled all along. Both teams failed to engage and interact with other teams on the project. The database teams did not perform well, and this is due to the frequent change or nonexistence of requirements, because the other teams took a long time to finalize them. On the other hand, all city map editors were completed, some surprisingly flexible. For example, the street that intersects other streets could have its parts individually selected and moved, and the part would dynamically adjust its size. One Setup team started with 7 participants but shrank by 1 after the midterm break. Still, they managed to produce about 260 pages of quality documents, and the best setup subsystem, while loosing 2 periods to public holidays.

Our goals of improving the quality of work products other than code, and the high ratio of active participation have certainly been achieved. The complaints that this kind of laboratory project required a lot of learning aside from the curriculum prompted us to organize the outstanding two briefings. The concerns that the project that failed to integrate would have a large negative impact on the final mark (60% for the individual effort, 40% for the team) were raised towards the end. Given that, except for RMI, we did not introduce any new technology that had not been used in the past, we take those complaints and their intensity as a measure of success and one more indicator of the broad commitment. We found a strong sense of accomplishment and maturity among those who managed to integrate their PBSS. *This is incredible. We use machines from all over the lab, and our system works!*

6. REFERENCES

[1] Bruegge, B., and Dutoit, A. H. Object-Oriented Software Engineering: Using UML, Patterns, and Java. 2nd International Edition, Prentice Hall, Upper Saddle River, NJ, 2003.

[2] Liu, C. Enriching Software Engineering Courses with Service-Learning Projects and the Open-Source Approach. (ICSE 2005) (St. Louis, MO). ACM Press, New York, NY, 2005, 613-614.

[3] MySQL, www.mysql.com

[4] Kruchten, P. The Rational Unified Process: An Introduction, 3rd Edition, Addison-Wesley, Reading, MA, 2003.

[5] Lemon, W. F., Liebowitz, J., Burn, J., and Hackney, R. Information Systems Project Failure: A Comparative Study of Two Countries. Journal of Global Information Management, 10, 2 (2002), 28-39.

[6] Vliet, van H. Some Myths of Software Engineering Education. (ICSE 2005) (St. Louis, MO). ACM Press, New York, NY, 2005, 621-622.