Project-Driven Learning-by-Doing Method for Teaching Software Engineering Using Virtualization Technology

http://dx.doi.org/10.3991/ijet.v9i9.4006

Kun Ma, Hao Teng, Lixin Du, and Kun Zhang
University of Jinan, Jinan, China

***Abstract*—Many universities are now offering software engineering an undergraduate level emphasizing knowledge point. However, some enterprise managers reflected that education ignore hands-on ability training, and claimed that there is the isolation between teaching and practice. This paper presents the design of a Software Engineering course (sixth semester in network engineering) at University of Jinan for undergraduate Software Engineering students that uses virtualization technology to teach them project-driven learning-by-doing software development process. We present our motivation, challenges encountered, pedagogical goals and approaches, findings (both positive experiences and negative lessons). Our motivation was to teach project-driven Software Engineering using virtualization technology. The course also aims to develop entrepreneurial skills needed for software engineering graduates to better prepare them for the software industry. Billing models of virtualization help pupils and instructors find the cost of the experiment. In pay-as- you-go manner, two labs and three step-by-step projects (single project, pair project, and team project) are designed to help the students to complete the assignment excitedly. We conduct some detailed surveys and present the results of student responses. The assessment process designed for this course is illustrated. The paper also shows that learning-by-doing method correlates with the characteristics of different projects, which has resulted in a successful experience as reported by students in an end of a semester survey.**

***Index Terms*—Learning by doing, software development process, software engineering, teaching, virtualization**

I. INTRODUCTION

Software engineering is a mature discipline that plays an increasingly important part in a technology-enabled connected world. These changes in this new century are probing the boundaries of software engineering education and training. It is often claimed that new technologies can help educators to make a difference in software engineering teaching and learning at the undergraduate and graduate level [1]. Traditional software engineering focuses on the knowledge point teaching of requirement, design, development and testing of a desktop computer or Web application. Common language usage is either C++ or Java. However, the spoon-feeding method of teaching often tends to make the traditional classroom teaching dull and tedious. It ignores the training of application and manipulative ability. Some enterprise managers complained that college graduates could not meet the requirement for talent raining out of touch with business needs, and they are not familiar with the popular software engineering tools. Current status of the talent cultivation in software engineering has been disjointed to it [2]. Therefore, the motivation of our course is more turned toward professional ability training that the engineers face in the real world, including emerging development method and usage of recent software tools.

Currently, Virtualization Technology (VT) is one of significant technologies that provide maximum benefits and opportunities for research and education [3]. The infrastructure in the cloud provides an experimental environment for testing and deploying applications. An innovative idea is how to benefit teaching of software engineering from the emerging virtualization techniques [4]. Accordingly, this paper will showcase how alternative virtualization technologies have been already successfully leveraged in teaching of software engineering to senior students.

We have made significant development efforts to architect a lightweight virtualization platform, which will direct students to test and deploy applications that they develop. By using such a platform, computational resources to be allocated are in a pay-as-you-go manner. This change is beginning to percolate into undergraduate curriculum. We emphasize the design and implementation of an application with lowest cost that is evaluated by the billing models caused by virtualization.

There are many other challenges as the frontiers of software engineering education and training [5]. For example, how to apply the software engineering knowledge in software development process? How to make students understand software engineering principles through interactive development? How to apply some emerging techniques in teaching of software engineering?

The contributions of this paper are several folds. First, this paper describes the experiences and lessons on integrating virtualization into senior software engineering course, allowing testing and deploying applications in virtual machines. We started this professional curriculum to the students who majored in network engineering at University of Jinan (UJN), which was ordinarily taken by students in the sixth semester. To make the course more relevant to current computing challenges, we made virtualization an important theme of software engineering. We wanted to emphasize virtualization at all levels, from concept to application. For the past two years, we have used the virtualization as the experimental environment. We had

33 students in 2013, and 23 students in 2014. Moreover, we had an assistant, who was responsible to the laboratory education.

Second, we want the students to understand principles of software engineering, and overcome any resistance and obstacles that students meet during collaborative development process. Therefore, three step-by-stop leaning-by-doing projects are introduced.

Third, clouds make the cost and efficiency explicit. For many instructors and pupils, the virtualization platform will be a novel environment, in which students are given quotas for virtual resources available to them. This new experimental environment is different from traditional software engineering lectures. In the cloud, users rent resources for testing and deploying applications. This alternative offers an experimental testing and deploying environment, which is the tradeoff between efficiency and cost. Accordingly, we will design this course to teach software engineering using this technology.

Beyond technical aspects, the discussion of the pedagogical benefits will help instructors and pupils find motivations to give a practical try to lightweight virtualization solutions in software engineering course. Therefore, our objectives are to foster the kind of hands-on practical ability in the perspective. The success of this software engineering curriculum is a typical case of the integration with cloud platform.

The remainder of the paper is organized as follows. The related work is discussed in Section II. In Section III, we introduce an overview of technological and pedagogical goals and methods. Section IV presents the course design. In Section V, we evaluate our experience in several dimensions, such as student work, student satisfaction, and case teaching. In Section VI, we analyze several negative lessons learned. Brief conclusions and future research directions are outlined in the last section.

II. RELATED WORK

Several schools are now teaching software engineering as part of their undergraduate curriculum. Next, we summarize these efforts in contrast with our own.

Teachers in some universities teach software engineering using object-oriented principles. The success of the course comes from the way in which object oriented design, software engineering process and the case study work so well together. They are used to teaching waterfall-like model in the textbook, but ignore the changes of software development process with some new techniques in reality. It leads to some inconsistency between the knowledge hierarchy and practical needs.

Recently, there are some popular ways to teach software engineering abroad. The first way is the Massive Open Online Course (MOOC), which has recently gained high popularity among various universities and even in global societies [6]. Students from a variety of geographical locations, can learn software engineering online. However, this way has some drawbacks on assignment grading due to the lack of enough interaction between teachers and students. For example, the Coursera MOOC on ”Software Engineering for SaaS” taught by Fox and Patterson adopted an auto-grader based on a test suite prepared by the teacher [7]. However, the grading effectiveness heavily depends on the quality of the prepared test suite. Monash University owns large-scale students who enroll in software engineering. Teachers has developed a course centered around a large but enjoyable software project.

”Do more with less” (higher pass rates and higher student satisfaction) is the basic principle to achieve even higher quality results. Teachers in Microsoft attempt to take the development of an interactive game as a typical case with a strategy for teaching university students the dynamics of a software project [8]. Teachers in Jackson State University taught software engineering through the use of innovative mobile application development [9]. Developing applications for mobile devices provides the opportunity to meet both of these requirements while introducing undergraduate and graduate students to topics and skills that can contribute to their employability.

III. OVERVIEW

A. Technological and Pedagogical Goals

First, we wanted instructors and pupils to see the actual cost of resources that are caused by the applications they develop. This cost is considered as the evaluation criterion of the quality of the software development process. Second, we wanted students to experience project-driven learning in practice. They will have fun in the collaborative labs and projects. Third, we wanted students to experience developing and testing in different virtual machines. Three machines are assigned to each pupil to do the experiment. Considering the number of students, using purely physical university IT resources (40 machines per room) would have forced unacceptable infrastructure costs. Last, we wanted the students to master principles of software engineering by solving the concerns they meet during the software development process.

B. Technological and Pedagogical Methods

The focus on this paper is inspired by the opportunities and challenges of cloud to be applied in software engineering course, rather than the technology itself. We answer four questions: Can we effectively make the students understand knowledge points of software engineering, and present the cost of this process using cloud economical billing models? Can students manage virtual machines to test and deploy applications they develop? Can students write a testing script to evaluate the applications they develop? What difficulties do senior students confront with during collaborative labs and projects, and what will they do to solve these issues?

C. Billing Model

Billing is the most important module of the virtualization platform. In the pay-as-you-go manner, we designed the post- paid billing model of virtual resources. We created the on- demand instances to let students pay for the resources by hour or number of times. This price is just to evaluate our expense on the development process. The price details are listed in Table I. The configuration of the guest VM is one CPU, 1G memory, 10G non-system disk, and 10 M bps bandwidth. Since it is encouraged that students use green computing techniques, the cost of the guest VM with lower CPU utilization is low fee-based.

TABLE I.
THE PRICING OF THE CLOUD BILLING.

On-demand instances	Usage
Creating of guest VM	CNY 1 per time
Start or stop VM	CNY 0.1 per time
Guest VM (CPU utilization < 30%)	CNY 0.15 per hour
Guest VM (CPU utilization > 30%)	CNY 5 per hour

At the end of the semester, the details of billing were sent to the students. Although instructors and pupils could use this platform we developed for free, the billing was just helping them understand the economic model of cloud computing. Our budget was for CNY 100 per student per semester. But we estimated that cloud usage could easily exceed the expectation based upon our experience. We anticipated that most of our usage would occur shortly before the deadline. We have also made extra budget for students who made mistakes and needed additional resources could still complete the assignment. We wanted the students to fully utilize resources using the least expensive.

IV. COURSE DESIGN

We wanted students to understand the principles of soft- ware engineering and experience software development (coding, testing and deploying) using the virtualization platform. Therefore, we designed our assignments to be sufficiently measurable that it would be obvious why one would want to rent virtual resources rather than use one's own machines. We choose some testing scripts to evaluate the performance of applications students develop because the cost and efficiency benefits are more apparent using billing models. In this study, we use these testing scripts to evaluate the performance of each application.

It was important to us that students used the same tools that professionals use as much as practical. One reason was teaching students the mature tools. Another was that we did not want students to form the impression that they only do simulation experiments. We wanted students to understand that the development process is pervasive when they do in the enterprise.

Originally, our programming language of the assignments was Java. This relied on that Java course is taught before the sixth semester in our university.

A. Labs

Our course had two hours labs to utilize the popular soft- ware engineering tools in year 2013 and 2014, which is shown in Table II. The first lab is experiencing version control tool, and the second lab is experiencing UML tool. An integrated tool may be employed that assists the student in learning not only how to develop applications collaboratively but also how to manage the project team. All the labs served as tutorials and preparatory work for the project assignments.

B. Projects

We had only one team project in the year 2013. We adopt a waterfall-like model to develop applications collaboratively. Some feedbacks of students indicate that students cannot learn much as we expect. Therefore, we switched the project to three groups in 2014. We want the students to experience more than before. Besides, we added an demonstrative case to teach students by hand in the year 2014.

The first project is single project with two class hours. Each pupil will think about the historical software assignment of the leading course in the prospective of software engineering. The second project is pair project with four class hours. In this project, one student will cooperate with another peer to develop an application. The assignment is usually accompanied by both students. The third project is agile team project with six class hours. Several students (from 3 to 5) are grouped to develop the applications together.

We have an intermediate check at the end of the pair project, which is just an acquaintance of the project progress to direct the students to answer questions and complete the assignment successfully.

C. Lectures

Beside labs and projects, we had 48 class hours of lectures (average 3 class hours per week) to teach software engineering knowledge and cases. These lectures are just up to one semester. We had one written examination at the end of the semester. Students were assessed primarily on their projects and the examinations.

TABLE II.
ASSIGNMENTS USED.

Semester	Line of Code	Description	Time
2013/2014	0	Lab 1: version control tool	2 class hours
2013/2014	0	Lab 2: UML tool	2 class hours
2014	200	Project 1: single project	2 class hours
2014	200	Phase 1: object-oriented coding	after class
2014	0	Phase 2: unit testing	1 class hour
2014	0	Phase 3: performance analysis	1 class hour
2014	500	Project 2: pair project	4 class hours
2014	0	Phase 1: role assignment	2 class hours
2014	500	Phase 2: object-oriented coding	after class
2014	0	Phase 3: code review	2 class hours
2013/2014	1500	Project 3: agile team project	6 class hours
2013/2014	0	Phase 1: project start	2 class hours
2013/2014	0	Phase 2: rapid iterative process (requirement analysis, design, development, and testing)	4 class hours

V. EXPERIENCES LEARNED

This section evaluates our experience in several dimensions. First, we evaluate the quality of student work of labs and projects. Second, student satisfaction is also surveyed to improve the teaching of software engineering. We emphasize case teaching and collaborative development, because it is tied directly to our pedagogical goal. When using this platform, the cost per student becomes visible to both instructors and pupils in a way that is not currently common in other courses.

A. Student Work

First, all the students completed Lab 1 and 2. They were experiencing the use of version control and UML tools. The potential peak for our assignment was approximately 90 virtual machines (3 VMs per student and about 30 students) using over-commit technology. As a rule of thumb, most of the students will do the experiments near the intermediate checking point and the last deadline. This might run out of the virtual machines since the virtual resources (at most 70) are limited. We wished enough students to do the assignment early. So we decided to record the number of active virtual machines for further observation. Fortunately, this has not caused any issues we worried.

Second, we provided students with testing scripts to launch automated software performance test. These scripts were responsible for the necessary billing. Most students got closer and right results, which reached the expecting target basically. That some students submit anomalous results is also useful pedagogically. Students benefit from seeing what happened to them or their classmates at least.

Last, the overall quality of case teaching impressed us. In the year 2013, 42.1% of student projects passed all our test use cases, and many more failed only due to minor bugs that could have been caught with more testing. 93.6% of student projects passed design report test. It is surprising to us that some students who failed programming completed the report test. It is an important reminder for all of us that students who memorized the knowledge points mechanically were lack of the ability in practice.

We want students to complete all the assignment in class except class preparation and some coding. Table III shows average time and energy of students take. It is shown that students invest more time and energy after curriculum reform.

B. Student Satisfaction

We now turn to quantitative evaluation of how well our course ran in terms of student satisfaction. At the end of the 2013 semester, we made a survey to our class. We asked students to rank the two labs and one project in terms of value. The first two labs are impressive. Table IV shows the feedback of student satisfaction. About 70% of students listed that team project was the most important, another 20% considered it general, and 10% did not care about team project. However, the project received some criticism. The students who were dissatisfied primarily focused on the unfamiliarness with collaborative programming. Many students had comments like ”I didn’t know what to do in the team project, since we had no experiences on it”. After we change this project to three step-by-step projects (single project, pair project and team project), this course has won the praise of instructors and pupils. We conclude that students are passionate about the assignment, even given its deficiencies and challenges. In general, the labs and projects were overall definitely still worth it.

We asked students definitely whether they would promote continuing or replacing the current labs or project. The survey results were shown in Table V. 65% of students advised to remain unchanged, and another 25% marked ”Although there are pros and cons, it is better to keep it.” Only 5% marked ”better to drop” without specific suggestions. Last, about 5% selected ”definitely drop”. Only one student choosing ”drop” commented ”These labs and projects are too simple.”. When all things are considered together, we still believe that the difficulty is moderate to most of students.

Several students thought that the project was valuable to upgrade their professional knowledge. One student commented:

”Too many employers are seeking for the graduates who are good at skills of team development. This curriculum is helpful to improve their competitiveness of employment.” Another noted that ”employers at job fairs really seemed to like the professional talents in this field!”.

We designed a questionnaire to get more details. Table VI shows the result. The standard for evaluation is shown as 1: lowest level; 3: basic written knowledge; 5: capable of passing enterprise interview; 8: flexible use; 10: master theory and practice. It is shown that this course works well in the last two years.

TABLE III.
AVERAGE TIME AND ENERGY OF STUDENTS TAKE.

Survey Item	2012 semester (Traditional SE)	2013 semester (Our SE)	2014 semester (Our SE)
Number of students	30	33	23
Hour per week	2+0.5	3+1	3+1
Code line per student	0	3000	4000

TABLE IV.
QUESTIONNAIRE ON SATISFACTION.

Grade	Comments	Support proportion
Positive	Team project was most important	70%
Cautious	Team project was general	20%
Negative	Not care	10%

TABLE V.
QUESTIONNAIRE ON THIS COURSE.

Positive comments	Proportion of course
attendees remain unchanged	65%
better to keep	25%
Negative comments	**proportion of course**
attendees better to drop	5%
definitely drop	5%

TABLE VI.
SELF EVALUATION.

survey item	2013 semester (before)	2013 semester (after)	2014 semester (before)	2014 semester (after)
Overview	2.9	4.1	2.3	4.9
Requirement Analysis	2.8	4.6	2.7	4.9
Project Management	2.7	4.8	2.6	4.8
Design	3.1	4.8	3.0	4.8
Coding	3.7	5.6	3.3	5.4
Testing	2.7	4.5	2.4	4.8

C. Case Teaching

We present an occurrent case-study on teaching an under- graduate level course on software engineering. Our approach has two main elements: start delivery of real project and rapid iterative process. This project originated from a real project of our development groups. We had 4 members (a project supervisor, a technical manager, a system analyst and a coding engineer) in this group. We adopted agile development models. Project start costs us two weeks. The delivery includes recapitulative requirement, software architecture, business pro- cess, statistical statement, scheduling, staffing and quote. We provided the students video recording of our rapid iterative process in class to show project review, daily stand-up meet- ing, automated testing and pair programming in practice.

VI. LESSONS LEARNED

Generally, our experiment using virtualization platform we developed for undergraduate worked well. Even so, what is more worthy of reflection are several lessons learned.

A. Resources Waste

Inefficiency was the waste rather than costs. Some students accidentally left virtual machines running idle. We had written a script running in the hypervisor to automatically shut down virtual machines after a period of inactivity, though we did not always enable this feature initially.

A much larger problem was that some students would restart a new session of their entire experiment after any technical glitch or mistake. Consequently, many students would start a fresh set of virtual machines (killing and removing the old ones) each time they experienced a glitch. That is because solving the problem sometimes is more complicated than creating a new environment. Thus, this could get expensive quickly.

Our virtualization platform was only available on the cam- pus hosts, so students were seeking to use the virtualization platform from dormitories remotely. But these students were generally not aware of tools like SSH client or VNC remote desktop that would let them maintain their session on the instructional machines after the failure on the network. More- over, students assumed incorrectly that they could not continue using their existing session when they reconnected. This leads to the failures to reclaim memories or release resources.

B. Step-by-step Development Process

In the 2014 semester, we adopt step-by-step strategy to enrich the projects, since some students in the 2013 semester complained that the development process is difficult to learn. The step-by-step process is essentially the breaking down of the development to gain insight into the issues with different- sized groups.

In single project, the point lies unit testing and script-based performance testing. In pair project, it is refined in role assignment and code review. Code review refers to some reviews during pair programming. During pair programming, developers write, inspect, and change the code continuously. A code review, in contrast, involves inspecting the code later, usually when the author thinks it is ready for deployment. In this process, both students are used to solving the issues collaboratively. In agile team project, students can experience more than single and pair project, such as project start delivery and rapid iterative process. This change has drawn more attention from students to concern about the experiment.

Figure 1 shows the project tasks left as the day progresses. It is concluded that students will not accelerate progress until the checking point. In the new semester of 2015, we will make the check point every two weeks. This will force the students to do the experiment in each class.

C. Leading Knowledge Training

In the 2013 semester, some students complained that they have not learned some basic software development skills due to failing some leading courses. In the 2014 semester, the course assistants added two class hours lecture to train students basic methods of unit testing, script testing, and IDE debugging.

D. Objective Evaluation

We simplified the homework. For single and pair project, only a table was submitted as a homework. For

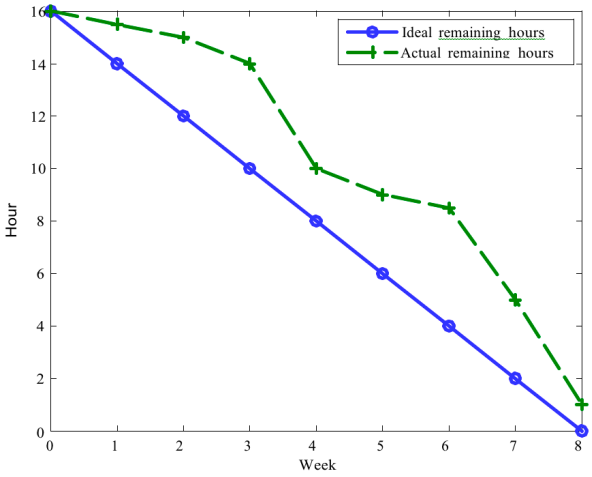

Figure 1. Project tasks left as the day progresses.

team project, a report table was submitted as a homework. We do not want the students to do their assignments until the last minute, and want them to complete it in class.

We make the team project to be a game. The program speed, numbers of testing case, click count of the published project website are all the objective factors to determine the experimental results.

VII. CONCLUSIONS AND FUTURE WORK

We have developed the virtualization platform to teach project-driven software engineering and bill for each student adapted to academic needs. There are still some lessons learned, but we have achieved tremendous benefits.

The cost to each student was calculated by the account. In the year 2013, we only spent CNY 98 per student on average. While in the 2014 version of our course, we only spent CNY 83 per student on average to save resources. The result of student satisfaction survey showed that students were satisfied with the course design as a whole.

Another outstanding success is the step-by-step projects. Students retrospected a program they develop in the prospec- tive of software engineering, to experience unit testing and performance analysis. In pair project, both students have been working together to do code review. Students have fun during team project to complete the assignment. The results of some surveys indicate learning-by-doing pedagogical method is feasible in practice.

We will improve the teaching in the year 2015. We are adding two class hours to discuss on the issues student meet in the experiments.

REFERENCES

[1] Carter, Adam S., and Christopher D. Hundhausen. ”A review of studio- based learning in computer science.” Journal of Computing Sciences in Colleges 27.1 (2011): 105-111.

[2] Jaramillo, Carlos Mario Zapata, and Maria Clara Gomez Alvarez. ”In- corporating Playful Activities in the Software Engineering Teaching.” Developments in Business Simulation and Experiential Learning 41 (2014).

[3] Caminero, A. C., et al. ”Obtaining university practical competences in engineering by means of virtualization and cloud computing technolo- gies.” 2013 IEEE International Conference on Teaching, Assessment and Learning for Engineering (TALE). IEEE, 2013.

[4] Sommerville, Ian. ”Teaching cloud computing: a software engineering perspective.” Journal of Systems and Software 86.9 (2013): 2330-2332. http://dx.doi.org/10.1016/j.jss.2013.01.050

[5] Sureka, Ashish, et al. ”A Case-Study on Teaching Undergraduate-Level Software Engineering Course Using Inverted-Classroom, Large-Group, Real-Client and Studio-Based Instruction Model.” arXiv preprint arX- iv:1309.0714 (2013).

[6] Coetzee, Derrick, et al. ”Should your MOOC forum use a reputation sys- tem?.” Proceedings of the 17th ACM conference on Computer supported cooperative work & social computing. ACM, 2014.

[7] Fox, Armando, and David Patterson. ”CS169: Software Engineering, University of California, Berkeley.” Computer Science Curricula 2013 (2013).

[8] Tillmann, Nikolai, et al. ”Teaching and learning programming and software engineering via interactive gaming.” 2013 35th International Conference on Software Engineering (ICSE). IEEE, 2013.

[9] Skelton, Gordon W., Jacqueline Jackson, and F. Chevonne Dancer. ”Teaching software engineering through the use of mobile application development.” Journal of Computing Sciences in Colleges 28.5 (2013): 39-44.

AUTHORS

Kun Ma, Hao Teng, Lixin Du, and **Kun Zhang** are with the Shandong Provincial Key Laboratory of Network Based Intelligent Computing University of Jinan, Jinan, China.

This work was supported by the Teaching Research Project of University of Jinan (J1344 and JZC12100), and the Doctoral Fund of University of Jinan (XBS1237). Submitted 04 July 2014. Published as resubmitted by the authors 15 October 2014.