ENHANCING SOFTWARE ENGINEERING METHODOLOGIES INTO J2EE PROGRAMMING PROJECTS

Ming Wang, California State University, Los Angeles, ming.wang@calstatela.edu
Xuesong Zhang, Southeast Missouri State University, c653buc@semovm.semo.edu

ABSTRACT

Traditionally, a programming course focuses on algorithm and data structure and no software engineering methodologies are introduced. A new way to teach programming is to integrate software engineering methodologies into project assignments. This paper presents how to implement object-oriented design, code review, test plan and personal software process into Java 2 Platform Enterprise Edition (J2EE) projects. The purpose is to help students develop good programming habits in an early stage. Positive responses have been received from the students who have taken the course. The paper concludes that software engineering methods can be successfully integrated into the development of projects in a J2EE course.

Key words: Software engineering, Programming languages, Object-oriented design, Object-oriented programming, Personal software process, Study and teaching

INTRODUCTION

In the traditional programming that concentrates on algorithm and data structure, the instructor provides an initial set of requirements for a programming project. However, these initial requirements are sometimes incomplete or ambiguous and students have to refine them by using their own judgment. The submission of an assignment usually includes the source code, output and executable code, and the grade is based on the correctness of the algorithm and output. No software engineering methodologies are applied to the entire project development process. Alternatively, a new way to teach programming is to include software engineering methodologies such as object-oriented design, defect debugging, unit testing, and the personal software process.

Typically, two major themes in the IS curriculum are system development and programming. There is no course that focuses on project design, testing and the personal software process. The new approach is to fill up the gap between system development and programming. The IS 2002 Model Curriculum and Guidelines (2) propose that an IS programming course needs to present object-oriented and procedural paradigms with software engineering methodologies. Software engineering methodologies stress software quality, solid design, effective process, and careful testing. Enhancing the methodologies in a programming project helps students avoid poorly designed programs and ineffective developmental process. To develop good programming habits in students, it is important to expose them to a professional approach for designing programs and to the disciplines of software engineering at an early stage in their careers.

This paper describes how to integrate design, testing plan, code review and personal software process into a J2EE programming class. J2EE has become more than just a programming

language now, but a well-established object-oriented programming system with a large collection of libraries and developmental tools. Writing a J2EE program always involves predefined or user-defined classes. Therefore, it becomes even more important to introduce students to software engineering methodologies when they programming with J2EE.

ANALYSIS AND DESIGN

Software analysis and design specifies how a program will accomplish its requirements. Proper design in software development can reduce the number of defects and developmental time, and produce better quality software (3). Introducing project design into the J2EE classroom helps students write better quality Java projects. It takes more time to teach students to develop projects with J2EE than with other languages because of J2EE's complexity (the number of class files and the length of the files). To ease the project developmental process, the instructor needs to facilitate students to conduct class design when a new project is assigned. In class, the instructor can help students utilize software design methodology to define the scope of the project and the input/output data requirement. The interaction between the instructor and students will confirm the project requirements until all key decisions have been addressed. Using Unified Modeling Language (UML) in Rational Rose 2002, students are able to design classes with data, methods, class names, and relationships between the classes. Figure 1 illustrates the UML class diagram for a bank account transaction project.

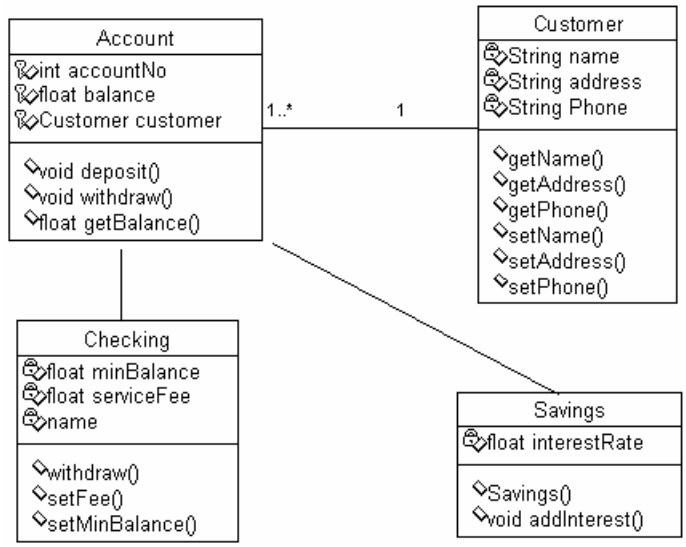

Figure 1. UML inheritance class diagram (Rational Rose 2002)

With these class diagrams, Rational Rose J2EE automatically generates class interface files with data and method headers inside. Students can define the methods of a class based on the generated Java interface file. The advantage is that the defects of a class are detected and corrected early before driver's design and development. The beauty of object-oriented technology is reusability. A set of related classes developed for one project can be stored as a package in the J2EE library and be reused in other projects later.

TESTING PLAN

Testing is a major part of software development. In the industrial world, more time is spent in testing than in any other phase of software development. A good testing plan will detect software defects effectively. A complete testing plan will include scenarios for every possible situation the program will encounter. Testing a project involves running it multiple times with various input data and observing the results.

Traditionally, the instructor provides a set of sample testing data for each project assignment for grading purposes in the academic environment. Running a program with specific input and producing the correct results only establishes that your program works for that input. It is important for students to develop a complete testing plan for their project by filling out a test plan form in Table 1 after their project design but prior to coding. These plans are swapped with another student in the class for testing and comments. Later when the project is returned, the testing plans with testing results and comments are returned to the students who developed the plans. This certainly provides an opportunity for students to re-exam their own testing plans and to learn from others. It is hard for any individual student to develop a complete testing plan at the beginning level. In order to reveal a complete testing plan of a project, the instructor needs to make a summary of the students' testing plans to the class. The top part of the test plan form used by students is shown as follows:

Table 1. Test Plan Form

Test Run#	Test Data	Expected Result	Actual Result	Comment

CODE REVIEW

Most students rely on the debugger built in J2EE to detect syntax errors right after coding. From software engineering's point of view, code review needs to be conducted prior to using debugging tools. Humphrey from Software Engineering Institute (5) indicates in his book that a typical engineer will find only about 2 to 4 defects in an hour of unit testing but will find 6 to 10 defects in each hour of reviewing code. Students need to do code review before their first compilation since most software defects result from simple oversights and goofs. Code review first will certainly save students compiling time. To do a code review, students study the printed source code to find errors. The key to conducting effective code review is having an effective code review procedure for personal use. Table 2 shows Humphrey's Code Review and Checklist modified for students to use.

Table 2. Design Review Guidelines and Checklist

Purpose	To provide guidance in conducting an effective design review	Program unit							
		1	2	3	4	5	6	7	8
General	• As you complete each review check the box at the right. • Complete the checklist for one program unit before you review the next one.								
Completeness	Verify that all functions in the requirements have been included in the design. This includes all inputs, computations, and outputs.								
Classes	The following design decisions are appropriate: • Number, purpose and type of methods • Exceptions • Generic features • Private data types								
Data Types	Member data and types are explicitly defined or clear in context								
Logic	• Verify that the messages in drivers sequencing is proper • Verify that all loops are properly initiated, incremented and terminated • Verify that all expressions compute the intended result								
Special Cases	Check all special cases: • Ensure proper operation with empty, full, null, minimum, maximum, negative, zero values for variables • Protect against out-of-limits, overflows, and underflow conditions • Ensure "impossible" positions are impossible • Handle incorrect input conditions								

PERSONAL SOFTWARE PROCESS

Personal Software Process (PSP) is a method used to help software engineers to track time and measure product quality. PSP is a set of methods that shows engineers how to use a defined, measured, planned and quality-controlled process to improve personal performance (5, 6). Hou & Tomayko (4) indicate that the PSP directly addresses quality and efficiency in software development. Prechelt (7,8) studies correlation between PSP methodologies and defect log data analysis in the software process with statistical hypothesis tests and also measures effectiveness of PSP training in realizing PSP potential benefits by comparing between PSP-trained programmers and those receiving other technical training. The result is that PSP users experience

improvements in their ability to estimate the size and time it will take to build a project, and greatly reduce their defect rates as a result. In the academic environment, Börstler et. al. (1) discusses the challenges and lessons learned in teaching PSP, which is the adaptation of a continuous improvement model to the specific needs of an individual software developer who wants to be more productive and produce higher-quality software. Positive PSP results at five different universities in the U.S. are presented in the study. Table 3 shows the form students record the time they have spent in the project development by phase.

Table 3. PSP Project Planning/Summary Report

Time in Phase (min.)	Plan	Actual	To Date	To Date %
Planning				
Design				
Design Review				
Code				
Code Review				
Compile				
Test				
Postmortem				
Total				

Table 4 illustrates the contrast of two sets of data indicating the time spent in each phase from the two students. The table indicates that Student A has spent less time on design with more time on compilation. However, Student B has spent more time on design and less time on compilation. Overall, Students B has spent less than half the time that Student A has spent on the project. Using student data is able to convince students of the importance of the design phase and draws their great attention to the design phase.

Table 4. Data Sample Comparison between Two Students

	Planning	Design	Code	Compile	Test	Total Time	Hours
Student A	5	7	57	1405	50	259 minutes	4.32 hrs.
Student B	12	35	52	16	12	129 minutes	2.15 hrs.

SUMMARY

Software engineering methods were introduced into J2EE student projects gradually during the course. UML class design was applied to the object-oriented design including encapsulation, inheritance and polymorphism in projects. Testing plan and debugging was introduced to test data types, predefined classes and exceptions in drivers. PSP was introduced for graphics, applets and Servlet/JSP projects. Figure 2 shows how software engineering methods are applied to appropriate projects.

As outcomes of the J2EE course, students are able to practice software engineering methods in their projects individually. Their project submission includes class design, testing plan, code

review checklist and PSP summary form as well as the source code, executable code, and output. The object-oriented design diagram is required to be turned in with their project submission.

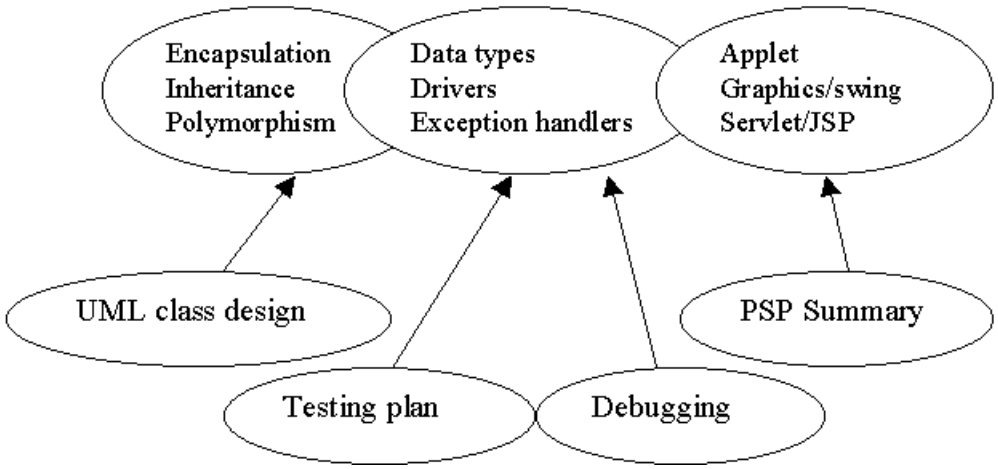

Figure 2. Software engineering methodologies applied to J2EE projects

ASSESSMENT

The impact on students was evaluated in an anonymous survey conducted at the end of the course. Practice software engineering methods was considered as a major contributor to their understanding of object-oriented disciplines and the fundamentals of J2EE programming. 63 questionnaires were returned out of 72 that were distributed to students in the different sections of the same course. 87% of students strongly agreed or agreed that UML class design helped them understand object-oriented disciplines and the fundamentals of Java programming. 82% of students strongly agreed or agreed that testing plan helped them build high quality projects. 61% of students strongly agreed or agreed code review was necessary before first compilation. 69% of students strongly agreed or agreed that PSP was an interesting experience in learning software engineering methodologies. However, 25% of students felt that recording time on PSP summary forms was time consuming.

These results indicate that integrating software engineering methods in the development of projects in J2EE course was successful and was well accepted by students. Students experienced the software engineering process activities that are almost impossible to get in the traditional classroom. The course established a bridge between application programming and system development courses. In addition, the course implements the IS programming course description in IS 2002 Model Curriculum and Guidelines (2).

REFERENCES

1. Börstler, J.; Carrington, D; Hislop, G.; Lisack, S; Olson, K; Williams, L, Teaching PSP: Challenges and Lessons Learned, IEEE Software, Sep/Oct. 2002, Vol. 19 Issue 5, p42, 7p

2. Gorgone, J., Davis, G., Valacich, J., Topi, H., Feinsytein, D. and Longenecker, H. IS 2002 Model Curriculum and Guidelines for Undergraduate Degree Programs in Information systems, Association for Information Systems, 2002.
3. Grove, R. F. Using the Personal Software Process to Motivate Good Programming Practices, SIGCSE Bulletin, ITiCSE 98 Proceedings, Volume 30, Number 3, September 1998.
4. Hou, L. and Tomayko, J., Applying the Personal Software Process in CS 1: An Experiment, The Proceedings of the Twenty-nine SIGCSE Technical Symposium on Computer Science Education, February 1997.
5. Humphrey, W. S. Introduction to the Personal Software Process, Addison-Wesley, 1997.
6. Humphrey, W., The Personal Software Process: Status and Trends, IEEE Software, Nov/Dec. 2000, Vol. 17 Issue 6, p71, 5p
7. Prechelt, L. Accelerating Learning From Experience: Avoiding Defects Faster, IEEE Software, Nov/Dec. 2001, Vol. 18 Issue 6, p56, 6p
8. Prechelt, L.; Unger, B., An Experiment Measuring the Effects of Personal Software Process (PSP) Training, IEEE Transactions on Software Engineering, May 2001, Vol. 27 Issue 5, p465, 8p