Incorporating Team Software Development And Quality Assurance In Software Engineering Education

Peggy Doerschuk

Abstract - One of the most important things that students can learn in a course in software engineering is how to effectively work in a team to develop software that is too large for a single individual to produce. It is also important that students learn the value of assuring software quality at each step of the development process. This paper describes how to incorporate a UML-based team project into an object oriented software engineering course. The project gives students hands-on experience in software development and quality assurance at each stage of the software lifecycle, including analysis, design, implementation, and integration. The project is divided into separate interacting parts, with each part assigned to a different team member. Students on each team assume alternate roles of software developer and quality assurance inspector/tester at alternate phases of the lifecycle. This paper describes the approach and an example project, including the problem requirements, timetable of deliverables, and sample deliverables.

Index Terms – Software engineering education, Team software development project; Quality assurance, Object oriented software engineering.

INTRODUCTION

This paper describes how to incorporate team software development and quality assurance in an advanced undergraduate/entering graduate course in software engineering. The paper includes (1) the motivation for this type of instruction; (2) the methodology used; (3) a sample project with sample deliverables; and (4) a discussion section.

MOTIVATION

Software engineering addresses programming in the large. It is concerned with the process of developing large software systems that are not only correct but also easy to understand, modify and maintain. Since most commercial software is too large to be developed by one individual, experience in team software development is very valuable to software engineering practitioners. For many students, a software engineering course is their first exposure to a well-defined process for developing software in a team.

Developing software in a team increases the complexity of the development process significantly. It introduces the problems of communication and cooperation. Accuracy in team developed software is harder to ensure because no one individual may fully understand the entire software system and because errors are often introduced as a result of poor communication and cooperation among team members [4]. These factors make quality assurance even more important in software development in the large.

METHODOLOGY

The author teaches a combined senior level undergraduate and entry level graduate course in software engineering. The course is an introduction to software engineering. In the course students learn object-oriented software engineering principles and techniques and gain hands-on experience through a group project. The grade for the course is based on a midterm exam, a final exam, several homework assignments and quizzes, and a project. The project gives students experience in software development and quality assurance at each stage of the software lifecycle, including analysis, design, implementation, and integration. The project lasts for the entire semester and is worth 30 percent of the grade.

For the project, students are grouped into teams. Each team elects its leader, whose primary responsibility as leader is to be the point of contact between the team and the teacher. The project is designed to be small enough to be completed in one semester by the team but large enough that it cannot be completed without the combined efforts of all of the team members. Deliverables are due at each phase of the project, including analysis, design, implementation and integration. Implementation and integration are done in two phases, with the main functionality implemented and integrated in phase one and the more advanced features implemented and integrated in phase two. Students demo their software to the class at the end of each phase. The two-phase approach teaches the students the benefits of iterative software development. It also allows students to correct errors and misunderstandings that are uncovered during phase one. Students must use the Unified Modeling Language in their project analysis and design. The implementation can be in either C++ or Java.

1 Peggy Doerschuk, P. O. Box 10056, Dept. of Computer Science, Lamar University, Beaumont, TX, 77710, israel@sal.lamar.edu
The approach taken for the project includes two key elements. The first is a methodology for dividing the project into separate interacting layers, with each layer assigned to a different team member. This makes it imperative for team members to reach agreement on the interface between layers, requiring students to learn effective teamwork and communication skills. The second key element is to have students on each team assume alternate roles of software developer and quality assurance inspector/tester at alternate phases of the lifecycle. This gives students practical experience in both software development and quality assurance. It also demonstrates how incorporating quality assurance at each phase improves the quality of the product.

Methodology for Dividing the Project into Separate Parts

A significant percentage of errors in large software systems is found in the interface between interacting modules [4, 7]. For this reason, it is important to give students experience in designing modules that can interact correctly with modules designed by other individuals. To enforce this, students are required to break the class design into three layers: the view layer, the business layer and the access layer. The layered design methodology is presented in [1]. The view layer contains all the classes that implement the interface with the user. They are responsible for getting information from the user and displaying results to the user. The business layer contains the classes that implement the business logic. The access layer contains the classes that store and access information. For this project, students have the option of storing information in files or in a database. For projects that use files, the access layer classes manipulate the files directly. For projects that use a database, the classes in the access layer contain methods that access the database.

Each layer is assigned to a different team member or members. This makes it imperative for team members to reach agreement on the interface between layers, requiring students to learn effective teamwork and communication skills. Another advantage of isolating the view and access layers from the business logic is that it allows for revisions to be made to the view and/or access layers without changing the business logic. This comes in handy for groups that initially implement the view layer in ASCII text in phase 1 and later revise the view layer to incorporate a graphical user interface in phase 2.

Incorporating Quality Assurance at Each Phase

The project incorporates quality assurance at each phase of software development. Each team is divided into two subteams A and B. The subteams alternate between roles of developer and inspector/tester at alternate phases. That is, Team A develops the Analysis documents and Team B performs the inspection of the analysis documents. Then Team B develops the design and Team A inspects the design. The roles are changed again for phase 1 implementation and integration and once more for phase 2 implementation and integration. This gives students experience in both software development and quality assurance. Because the quality assurance is not performed by the same individuals who did the software development at each phase, the quality assurance is more effective in detecting and correcting errors [4].

A formal inspection is performed on the analysis documents. On the due date, the development team turns in a copy of all the Object-Oriented Analysis deliverables outlined in Table 1 below to the instructor, along with a copy for each member of the inspection team. The inspection team meets with the development team to perform the inspection. This process is typically started during class time, but students must complete the inspection outside of class. The inspection team must write a report that lists each error detected, its type and severity level. The development team makes any appropriate corrections to the analysis documents and writes a follow-up report that details what errors were corrected and how, and a clarification of each item that was incorrectly identified as a fault. At the end of the inspection, the team submits to the professor the written report of the inspection team, the corrected analysis documents, and the written follow-up report of the development team. The grade for the entire team for the analysis is based on the corrected analysis documents. This encourages inspectors to detect errors aggressively because in so doing they are improving the grade of the team for that phase.

For the design phase, the subteams reverse roles. This means that the design is not developed by the same students who developed the analysis. Because the students who inspect the analysis documents know that they must use them as their roadmap for the design phase, the inspectors seek to make the analysis documents as correct, complete, explicit and unambiguous as possible. This makes for a higher quality analysis. The same inspection process is followed at the end of the design phase, with students reversing the roles of developer/inspector.

At the implementation phase, a walkthrough is performed. The developers turn in a copy of documented source code for the instructor and for each member of the walkthrough team. The walkthrough is initiated in class and completed by students after class. Because of time limitations, a formal walkthrough report is not prepared. Corrections to the source code are made as a result of the walkthrough. The walkthrough is particularly helpful in identifying problems at the interface between modules written by different members of the development team.

While the development team is working on the implementation, the quality assurance team develops test cases that will be used after the source code has been integrated. The test cases are graded by the instructor and returned to the teams before integration. They are used for execution-based testing after integration is complete. The team makes corrections to errors detected during execution-based testing. Deliverables at this stage are the corrected tested source code and a report by the testing team of what errors remain.

Implementation and integration are done in two phases, with basic functionality completed during phase 1 and more advanced functionality completed during phase 2. Students switch roles between developer/tester in these two phases.
Thus all students gain experience in non-execution-based testing and execution-based testing.

**A SAMPLE PROJECT**

This section includes sample data from an example project that the author used in a Software Engineering course in spring 2003. Included are the project requirements, schedule of deliverables, and sample deliverables.

**The Project Schedule**

The project is designed to be completed during a fourteen-week semester. Deliverables are due at each phase of the development cycle. A formal inspection is completed on the analysis and design documents. An informal walkthrough is completed on the implementation phases. A typical schedule of deliverables is shown in Table 1.

**TABLE I**

<table>
<thead>
<tr>
<th>Week</th>
<th>Due</th>
<th>Deliverables</th>
</tr>
</thead>
</table>
| 5    | OOAnalysis | • use case diagrams  
      |      |   • scenarios  
      |      |   • class diagram  
      |      |   • state diagrams  
      |      |   • activity diagrams  
| 5    | OOAnalysis inspection | • written report of inspection team  
      |      |   • corrected OOA documents  
      |      |   • written follow-up of development team  
| 7    | OODesign | • sequence diagrams  
      |      |   • collaboration diagrams  
      |      |   • detailed class diagrams  
      |      |   • client-object relations diagrams  
| 8    | OODesign inspection | • written report of inspection team  
      |      |   • corrected OOA documents  
      |      |   • written follow-up of development team  
| 11   | Phase 1 implementation | • copies of documented source code for walkthrough team  
      |      |   • black box test cases  
      |      |   • functional test cases  
| 12   | Phase 1 integration and testing | • corrected, tested source code  
      |      |   • test results  
      |      |   • in-class demo  
| 13   | Phase 2 implementation | • copies of documented source code for walkthrough team  
      |      |   • black box test cases  
      |      |   • functional test cases  
| 14   | Phase 2 integration and testing | • corrected, tested source code  
      |      |   • test results  
      |      |   • in-class demo  

**Sample Deliverables for the Analysis Phase**

For the Object Oriented Analysis phase, students must submit use case diagrams, a class diagram that includes only data but not methods associated with each class, scenarios for each use case, and any state diagrams and activity diagrams that are appropriate. A use case diagram and sample scenarios for the project are shown in Figure 2.

**Sample Project Requirements**

Requirements for the project are shown in Figure 1 below. Students complete the basic functionality of the system in phase 1 and report generation capabilities in phase 2 of implementation and integration.

---

**FIGURE 1**

**SAMPLE PROJECT REQUIREMENTS**

**MyFinances Lite Project Requirements**

In this project you will be producing a software product for tracking income and expenses. MyFinances Lite will allow the user to record information on the following types of accounts:

1. bank accounts (checking, savings)
2. cash accounts
3. credit cards

Users create a new account by entering the name, type, and description. For each account, the product will allow the user to record information on deposits, withdrawals, and transfers. For each of these operations on an account, the product will record the date, type (deposit, withdrawal, transfer, check number, fee), payee, amount, category or categories (for a split transaction), memo and updated balance after the operation is performed.

The following categories are automatically included:

1. income:  
   - net salary
   - rent
   - interest
   - other
2. expenses:  
   - car
   - clothing
   - education
   - entertainment
   - fees
   - groceries
   - home furnishings
   - medical
   - utilities
   - other
   - transfers

Users can also reconcile their bank and credit card statements. To reconcile a bank statement, the user enters the opening balance, ending balance, service charge, if any, and date, interest earned, if any, and date. The product will then display all unreconciled transactions on the account. The user marks each transaction that has cleared, and the products records a ‘c’ for these transactions. While reconciling, the user can also enter any new transactions that have not yet been recorded and can make corrections to entries that are incorrect. The product will display the difference between the net cleared transactions and the net amount for that statement. When reconciliation is successfully finished, all cleared transactions are marked ‘R’.

The procedure for reconciling a credit card is the same except that the user begins by entering the amount of charges and cash advances, the amount of payments and credits, the ending balance, and the amount of finance charges, if any.

In addition, the user can generate the following reports:

1. Itemized category report. The user can select the following options: (1) the period of the report; (2) the accounts to include; and (3) the categories to include. The itemized report will show all entries that match the selected criteria, sorted by category and then by date, with totals for each category.
2. Cash flow report. The user selects the same options as above. The report will show the total income and expenses by category for the selected period.
3. Where did the money go report. The user selects the period, and the report shows what percentage of income was spent on each expense category for that period.
Session F1C

SCENARIO FOR GENERATE REPORTS

2. User selects type of report (Itemized Category, Cash Flow, Where did my money go?).
3. User enters the appropriate selection criteria for the report.
4. System displays the information for the selected report.

SCENARIO FOR ITEMIZED CATEGORY REPORT

1. At Step 2 of Generate Report user selects ‘Itemized Category.’
2. At Step 3, user is prompted for and enters the period of the report, accounts to include, and categories to include.
3. At Step 4, System displays all entries that match the selected criteria, sorted by category, then date, with totals for each category.

Alternatives

1. If user enters incorrect data at Step 2, display error message.

FIGURE 2
USE CASE DIAGRAM AND SAMPLE SCENARIOS.

Sample Deliverables for the Design Phase

Deliverables for the Object Oriented Design Phase include a detailed class diagram that includes both data and methods. In addition, students turn in sequence diagrams and collaboration diagrams for each use case in the analysis and a client object relations diagram that shows which classes are clients of other classes. Sample deliverables for this phase are shown in Figures 3 and 4.

Deliverables for the Implementation and Integration Phase

At implementation, the development teams implement their modules and the inspection teams perform a walkthrough before the modules are integrated and executed. Inspection teams also formulate and submit black box and functional test cases that will be used after the modules are integrated. After integration and testing are completed, the team turns in the corrected, tested source code and a report from the testing team as to the results of testing.

FIGURE 3
SAMPLE SEQUENCE DIAGRAM FOR GENERATE REPORT USE CASE

In-Class Demos

Each team presents an in-class demonstration of their implemented, integrated and tested code. This is done first after phase 1 and then after phase 2. The two-phase approach gives teams experience in iterative development and helps teams to recover from misunderstandings and erroneous approaches that may become evident during demonstration of the phase 1 product.

In preparation for the class demos, students are given data that is to be used for demonstration purposes. For the MyFinancesLite project, this included a list of transactions on the various accounts, with updated balance shown after each transaction. Using uniform data makes it very easy to determine during the demo whether the implementation is correct.

DISCUSSION

A brief discussion is included here on the following relevant issues: 1. comparison of this project’s approach with extreme programming (XP); 2. demands of the project on the teacher; 3. comparison with traditional approaches to teams; and 4. benefits of this approach.

Comparison with XP

XP is the most frequently used method of agile software development. Agile software development focuses on customer needs by delivering working software frequently in short releases. XP concentrates on producing high-quality code by iteratively testing, coding, planning, starting with the basic set of initial requirements and incrementally adding needed functionality. It fits well with small development teams and vague or frequently changing requirements [2, 3, 5].
This is distinct from the classic life-cycle approach to software development which progresses from the planning stage through analysis, design, coding, and testing, with activities at each stage completed before going to the next stage. XP incorporates pair programming, continuous integration, and writing automated test cases before writing code.

Experience in incorporating XP in education has been mixed [6]. Some of the assumptions of XP are not supportable in the university environment, including an on-site customer, a forty-hour week, and co-location of the development team in a single office. Some educators believe that while some practices of XP can be adopted in the university setting, others do not support educational goals [8].

The project presented here uses the classical life-cycle approach. This fits better than XP with both the fixed nature of the project requirements and the environment at our university. While there may be some inadvertent ambiguities, the project requirements are generally fixed at the beginning and do not change. Also, at our university there is no dedicated block of space or time available for team members to work together.

Nevertheless, some aspects of this project loosely fit some of the practices implemented in XP. First, although XP’s practice of writing automated test cases before coding each iteration is not required in this project, testing (non-execution based as well as execution based) is incorporated at each phase, so great importance is placed on testing. Second, the project’s scheduled implementation, integration and testing in two incremental phases fits with XP’s iterative incremental nature. Third, some of the members of one team used pair programming informally during the implementation, integration and testing phase.

Demands on the Teacher

The project is designed to give students feedback and guidance at frequent intervals, starting from early in the semester. The does place certain demands on the teacher. The requirements document must be prepared and ready for dissemination to students by the second or third week of class. Grading and return of deliverables at each phase must be done in a timely fashion because products at each phase depend on those of the previous phase. The number of groups must therefore be small enough to allow this rapid turnaround. In practice, the author has successfully taught a class with five teams of eight to ten students, but fewer teams would be desirable.

Comparison with Traditional Team Approaches

The author has taught this class for four years and has been incorporating this type of project each year. The division of project teams into development and quality assurance subteams was instituted in spring 2003. This change resulted in a marked improvement in the quality of the deliverables at each stage. It also produced a significant change in team dynamics. The first inspection was traumatic for several of the teams. Group members argued for hours over insignificant issues, and individual egos inhibited team progress. As time progressed, students learned the art of compromise and reaching consensus, and at the end the teams were much more cohesive than in previous years. Overall, the final products were more complete, robust, and correct than before.

Benefits of the Approach

The project gives students hands-on experience in developing software in a team environment. For many students this is the first time they have been assigned something that was too big for them to complete on their own. One of the biggest challenges the students face is how to cooperate with others to complete the project. Dividing the design into three layers with each layer assigned to different individuals forces them to work together to reach agreement on the interface between their modules. This is very valuable experience, since a large percentage of errors are found at the interface between modules [4, 7].

The division of the teams into development and quality assurance subteams has been very beneficial to our students. Students see first-hand how the final product can be improved by performing inspections or walkthrough at each phase of the lifecycle. They also see that more errors are caught and corrected when quality assurance is performed by individuals who are not on the development team. The overall quality of the final projects improved greatly with institution of this change.

Students recognize that the project gives them valuable experience in software development that will be very useful in their careers. The use of an object-oriented design tool such as TogetherSoft’s Together ControlCenter, which was used in this project, gives them experience using UML and design tools that are likewise valuable additions to their resumes.

Students find the course project very demanding but also very worthwhile. After the project was introduced into our software engineering class, our students petitioned our department to offer the course every semester. It is now a required course that is taught every semester.

REFERENCES


FIGURE 4
SAMPLE PARTIALLY COMPLETE CLASS DIAGRAM.