2006-1601: ENGAGING SOFTWARE ENGINEERING STUDENTS USING A SERIES OF OBJECT-ORIENTED ANALYSIS AND DESIGN WORKSHOPS

Martin Zhao, Mercer University

Martin Q. Zhao is Assistant Professor of Computer Science at Mercer University. He has been teaching software engineering, database, and programming courses since 2001. Before that, he worked as a software engineer for two years and participated in the development of Web-based applications for clients including Nortel Networks and Bank of America. He received his Ph.D. and MS degrees from Louisiana Tech University and Beijing Institute of Information and Control.

Laurie White, Mercer University

Laurie White received an MS and Ph.D. in computer science from the University of Florida. She taught CS at Armstrong Atlantic State University, in Savannah, GA, for 10 years before coming to the Department of Computer Science at Mercer University in 1999.

Engaging Software Engineering Students Using a Series of OOAD Workshops

Introduction

It has always been a challenging task to prepare capable software engineers to meet the high demands of the industry. With the fast growth of computing technologies, future software engineers are expected to have a good working knowledge of object-oriented system design, distributed or web-based computing, multithreading, and database connectivity. In a typical computer science and computer engineering program, the only required course that may cover these topics is Software Engineering.

Classical software engineering textbooks [11, 10] emphasize process models and highlevel development activities but neglect to discuss how technologies are used in a real world project. Many new textbooks [4, 3] present up-to-date object-oriented development processes and demonstrate how design principles work through case studies. Using a well prepared series of workshops can give students hands-on experience of the concepts and practices demonstrated in those cases. This approach can effectively engage students in various stages of an incremental development process so as to teach them software engineering by really doing it.

In this paper, the effectiveness of such an approach in a Software Engineering class will be discussed in detail. A simple voice mail system with a sound OO design is adopted from a popular OOAD text [8] as the baseline design of the sample application used in the workshops. Throughout the semester, this system is expanded functionally to introduce advanced programming techniques. Basic OOAD activities and key software development best practices will also be demonstrated in the series of workshops.

Background and Rationales

Typically, Software Engineering is designed as a senior capstone course in a Computer Science program for students to integrate knowledge gained from the required core courses offered in a four-year period. According to CC2001 [1], this course is supposed to cover software system design, software processes, key activities in software development lifecycle, and software project management. The traditional approach to teaching a Software Engineering course, as reflected in classical textbooks [11, 10], usually starts with an introduction to software process models, which is then followed with discussions on highlevel activities in various phases of a generic software lifecycle template that can accommodate all possible programming paradigms. Although updated many times since their original editions, those texts are not well adapted to the latest paradigm changes (such as object orietation) and the cutting-edge technologies. Not enough effort seems to have been made to show how the models and principles discussed in the texts can be applied to real world projects.

A team-based software project is commonly included in a contemporary software engineering class to give students hands-on experience of the issues that they may encounter in a real-world

development environment. It is commonly accepted that the best strategy is to guide the students to learn software engineering by really doing it. [3, 12] Some new textbooks [4, 3] devote more detailed coverage on latest OOAD methodologies and demonstrate how exactly they can be used in the construction of medium-sized software systems by using case studies throughout the book. The use of a complete and comparatively complicated sample *application* has many advantages over using a number of simple and mostly isolated *programs*. First of all, it can set up a context for the discussion of various facets of OOAD techiniques that are applied in different stages in the development process. It can also facilitate discussions on topics relevant only to the whole system, such as layer and partition. It makes it particularly helpful in demonstrating the significance of an iterative/incremental development approach. A non-trivial application is necessary to allow for the inclusion of more advanced programming techniques (such as multi-threading and database connectivity) that are widely used in real-world software systems.

An appropriate case study may be chosen based on the specifics of the curriculum at a particular institution and the background of the students who take the class. At Mercer University, the Computer Science Department is hosted in the Liberal Arts College. All majors of its CAC accredited BS program in Computer Science are required to take a three-hour software engineering course, which also serves for Computer Engineering majors from the Engineering School. Occasionally, majors from other CLA programs may also take this course which counts towards a CS minor. Students typically as a minimum have already taken CS1, CS2, and CS3 prior to taking software engineering. The class size has varied from 12 to 18 in the past five years. The class is usually assigned to small teams of 3 to 5 students based on mutual interest in project topics proposed by each individual. The following issues must be addressed in the class to help the students to be successful in their team projects:

- All of the students who took this course in the past five years had learned C++ or Java or both from their previous courses, with some of them ranked highly in local or regional programming contests. However, since there is not a system analysis and design course in our curriculum, they have not been exposed to applications with more than a few classes.
- Certain advanced programming skills have not been covered in previous courses, such as advanced GUI programming, distributed computing, multithreading, and database connectivity, which are needed in developing complex software systems.
- The students have not been trained to use tools (such as UML toolkits and IDEs) which are widely used in developing large-scale software systems.
- They are not encouraged to use third-party software components in their programming assignments.

Previously, a series of smaller cases were used in this class as labs to help the students pick up these advanced techniques. Although the exercises helped students to understand the individual topics, they could not fully demonstrate how the techniques can be integrated into the system analysis and design process, which is the area where students needed the most help. To address this issue, a comprehensive case study that incorporated OOAD activities and the desired technologies was developed in Fall 2005. The development process was presented as a series of workshops. Each workshop focused on one development phase or a specific technology, and was used after one or two lectures on the same topic. A few elements in the part of the system that a workshop focused on were intentionally left missing for the students to have some hands-on exercises. Along with exploring individual topics covered in the workshops, the students also

experienced the incremental development process of a full-featured multi-tier system. The complete case study worked well as an example for the team projects.

Overall Strategy

Lectures and guided laboratories are common practices in teaching a wide range of computer science and engineering courses. Lectures are a necessary component in teaching a software engineering course to present concepts, principles, and technologies, which are necessary to understand the background of a development scenario. But they are not efficient in demonstrating what artifacts are to be generated under the given scenario and how to use CASE tools to generate them. The coverage of what to do and how to do it is essential for the students to fully understand the fundamentals and carry them onto their projects.

Workshops are considered as a more effective tool in conveying scenario-specific materials to the participants. When used after a lecture that introduces a new topic (such as requirement elicitation), a workshop can zoom in on a scenario in the sample application and set the participants into the context of a related project activity (such as analyzing a problem statement). Depending on the background of the participants, the class can either continue to elaborate the topic of concern, or introduce related activities and tools used (such as drawing use case diagrams using a modeling toolkit), and guide the participants to have some hands-on exercises.

A series of workshops, with each focusing on one or more significant aspects of the development of the sample application, can walk the students through a typical software lifecycle and detail the key activities and artifacts to a level that can best help the students get prepared for their team projects. The students can gain deeper insight into the topics discussed in class and have necessary exercises on technologies and skills needed to be successful in developing useful software systems.

Although sharing a common goal of giving hands-on exercises with traditional labs used in programming courses, workshops make themselves unique in that they emphasize the specific scenario being a part of a complete process/system. Workshops are more application-oriented and concern about how the skills fit in a well-defined context, whereas labs are usually targeted on building basics skills to solve an individual problem. The level of detail may also be different as required by the two formats.

High-Level Description

The selection of a case study that fits the needs for such a series of workshop is then the most important decision to make. An ideal case study will be a meaningful application in the real-world, which usually allows for using technologies such as GUI, client-server, multi-threading, and database. But the scope and complexity should be limited to fit in a classroom setting: No complicated business logic should be pursued. The application can be built using platforms and tools that are readily available at the institution.

Although many sample cases may satisfy the general requirements as listed above, a brief research on some textbooks that have a running case study reveals that none of them seem to fit

our needs without a change. Braude [3] presented an interesting role playing game (RPG) application with an extensible framework, but it is considered incomplete since it has just a GUI. Bruegge and Dutoit [4] used a multi-user, Web-based system, but the complexity in both application logic (generic online tournament organization) and technical aspects (such as Java RMI) would require too much background introduction.

A simple voice mail system (referred to as VMS in class) with a sound OO design as presented in a popular OOAD text by Horstmann [6] is selected as the baseline design. This original system is a typical classroom type of example. It has a simple graphical interface that resembles the layout of a phone receiver, with a keypad and textareas for a speaker and a microphone. System prompts and user messages are displayed as text in the corresponding textareas. A fixed number of mailboxes can be created when the system starts up. A user can interact with the phone GUI to connect to one of the mailboxes and leave a message. With a correct passcode to a particular mailbox, a user can perform certain owner operations, including listen/save/delete a message and change greeting and/or passcode.

This sample application can serve as a good starting point for the case study to be used throughout a series of workshops as proposed. It will be extended functionally to include such features as socket-based client-server communications, a server that can handle multiple client requests simultaneously, mailbox and message persistence using a relational database. The final system can reflect a real-world type of application with a 3-tier architecture. The workshops can walk the students through an incremental development process, and demonstrate key OOAD activities and related technologies from inside out.

A series of workshops are then designed to demonstrate key OOAD activities and relevant modeling and programming skills when new features are incrementally added into the working prototype. The following topics are selected for a total of 10 workshops to cover:

1. Requirement elicitation using use case modeling
2. Requirement analysis using structural and behavioral modeling (class, sequence, and statechart diagrams)
3. System architecting demonstrating concepts such as cohesion and coupling, layer and partition
4. Tool-based software development – using IDE to create and build projects, importing from third-party components
5. Distributed computing with socket – the monolithic baseline design is extended to a client-server system
6. Design patterns
7. Data persistence with DBMS and JDBC
8. Enhancing the server with multi-threading
9. Systematical and automated unit testing using JUnit [2]
10. Wrap-up, and packaging deployable system builds

Highlights of Selected Workshops

To achieve the best results, workshops need to be arranged in line with lectures and student team projects. The first two workshops focused on requirement engineering, which were given right

after a few lectures on system modeling using UML and key requirement elicitation and analysis activities. Technically, these two workshops demonstrated a reverse-engineering approach: building system models (including functional, structural, and behavioral models) from the original baseline system. The Visual Paradigm for UML 5.0 toolkit [13] was used to train the students with tool-based modeling skills that are needed in their own projects.

Workshop 1 practices functional modeling using use cases. The students were asked to execute the baseline VMS system before class to identify different types of users and different ways in which VMS can be used by each type of users. Class discussions based on their observation led to a use case model as illustrated in Figure 1, which summarizes the high-level system functionalities as six use cases initiated by two actors (Caller and mailbox Owner). The *include* relationship as introduced in a previous lecture was demonstrated here to clarify questions the students still had. The students were then asked to formally document each use case using a standard template, which they would then use in their own projects.

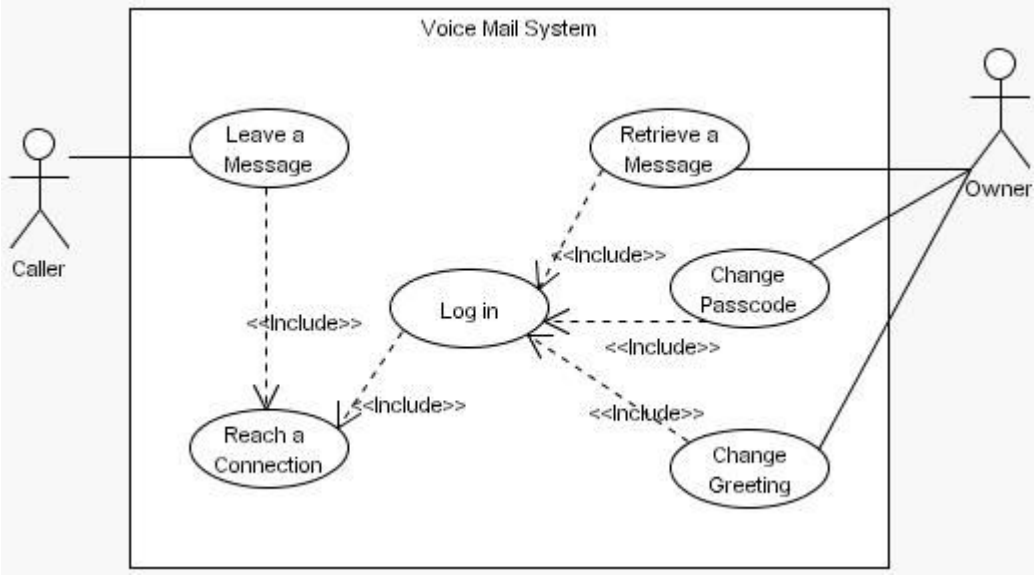

Figure 1 – Use Case Diagram Used in Workshop 1

The use case model helped the student to grasp the system requirements from a user's view. As a question in Workshop 1, the students were required to reflect on what objects are needed for the system to fulfill the functional needs. Candidate classes proposed by students were discussed in Workshop 2 to identify the list of *analysis classes* that were agreed upon by all. A class diagram was then used to model the *domain classes* as well as relationships between them (Figure 2).

In the lecture prior to Workshop2, the concepts of *Boundary*, *Control*, and *Entity* objects were introduced, which were applied to the analysis of the analysis classes. The students found it easy to distinguish different types in the given scenario, which helped them to understand the interactions between actors and a boundary object (such as `Telephone`), and between a control object (such as `Connection`) and the other two types of objects as represented in sequence diagrams (Figure 3). This workshop also provided an example of how to use statechart diagrams

to describe the sequence of states that a complex object (such as the `Connection` object) goes through in response to external events.

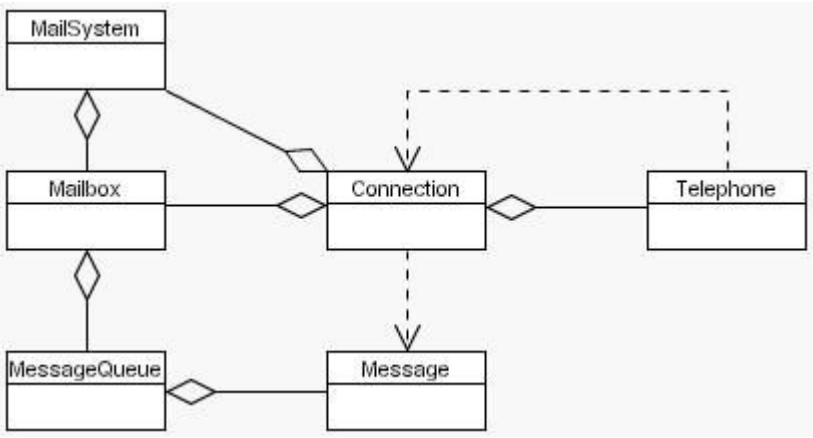

Figure 2 – Analysis Class Diagram Used in Workshop 2

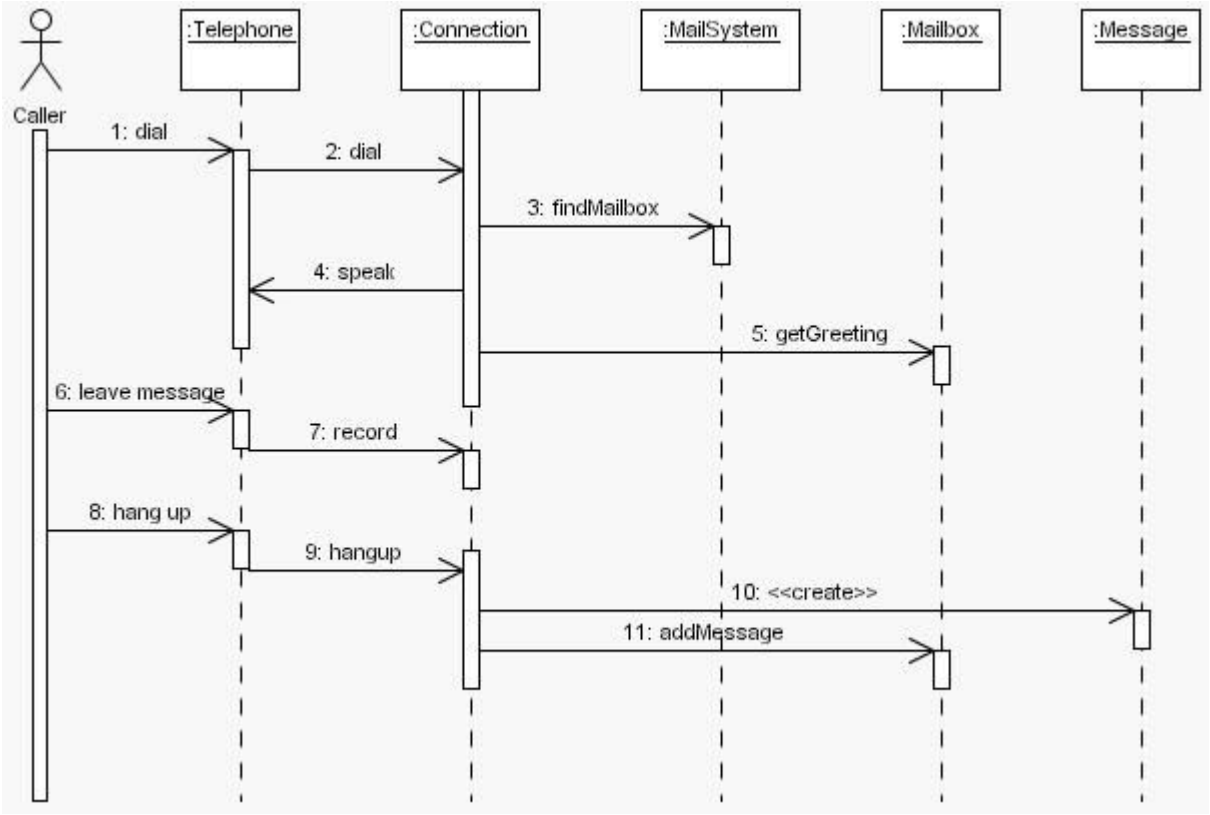

Figure 3 – Sequence Diagram for the `Leave a Message` Use Case

By executing the existing baseline system, modeling the required functionalities using various UML diagrams, and later reading the corresponding code base, the students achieved deeper insights on how the OOA activities can help developers to bridge the gap between an external view of *what need to be done* and a conceptual model of the system, which describes at a high

level *what objects the system needs to have*. These insights would be very helpful in subsequent workshops (especially 3 and 6) that focused on system design activities.

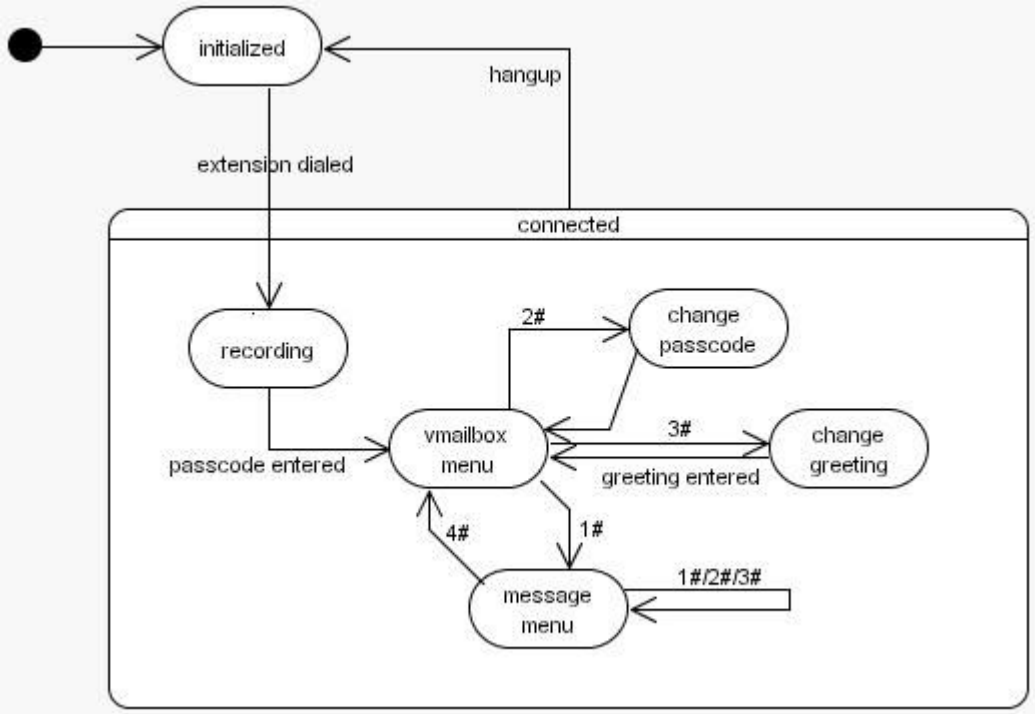

Figure 4 – Statechart Diagram for the `Connection` Class

Workshop 3 was used to demonstrate concepts of systems architecture. The monolithic baseline system was divided conceptually into a client subsystem and a server subsystem, and a data store subsystem would eventually be added for data persistence. In this workshop, the emphases were on system level decomposition, layered architecture, as well as relevant UML diagrams. The details about socket-based distributed computing and data accessibility were discussed in Workshops 5 and 7.

With the understanding of the structural model of the baseline system (Figure 2), the students could easily find out that all classes except `Telephone` need to be included in the server sub-system. A 3-tier architecture was demonstrated using a package diagram as shown in Figure 5. A communication subsystem (`vms.connect`) was introduced, which is responsible for linking the client and server and will be deployed with both of them. The potential of including an administration subsystem (`vms.admin`) for managing mailboxes and auditing messages was discussed, which demonstrated using partitions (`vms.phone` and `vms.admin`) to organize peer subsystems that belong to the same layer. The practice of using value objects (`vms.vo`) [5] that can be accessed globally was briefly mentioned.

Workshop 4 provided a hands-on exercise for the students to get familiar with IDE. Since only one student had previously used an IDE, this was a badly needed training session before directly handling the code base in the subsequent workshops and their projects. Another feature worth mentioning is the inclusion of a simple implementation of the Java text-to-speech (TTS) API [6] that can enhance the phone GUI by "speaking" the system prompts and user messages. In

addition to attracting interest from the students who had not written a program that speaks, this feature also presented a vivid example of reusing software components in a project, and the consideration of purchasing/using existing components versus developing in-house.

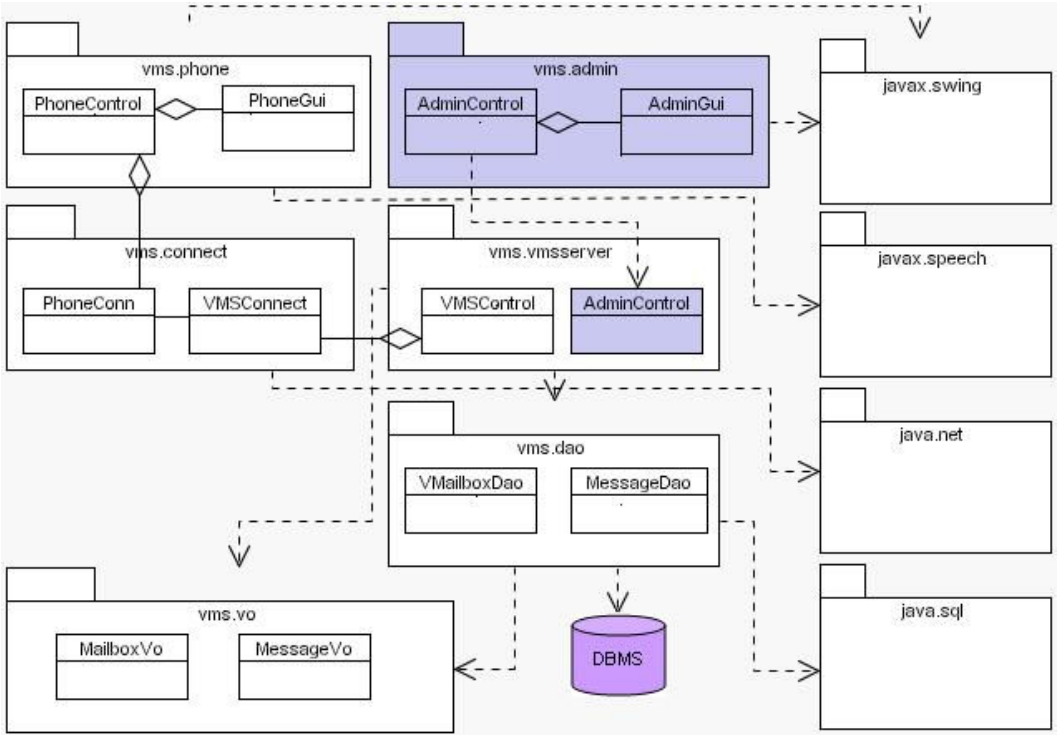

Figure 5 – Package Diagram Used in Workshop 3

Workshop 6 focused on design patterns as used in GUI design. It assumed a scenario that the simple phone GUI in the client subsystem needs to be replaced with an upgraded version: a phone that can flip on and off. By showing how the Factory Method pattern [7] can be used to decouple the main class of the VMS client (that create the phone GUI) and the specific product (i.e., `Telephone`) type that is actually created, this workshop demonstrated the beauty of using design patterns to come up with flexible design. In addition, this workshop also demonstrated the principle of *levels of abstraction* with a detailed design class model as shown in Figure 6. At the highest level of abstraction, a phone client is modeled with an interface `ITelephone`, which is partially implemented with an abstract class `GuiPhoneClient`. Concrete classes `Telephone` and `TelephoneWithFlip` are at the lowest level, which can be created by a concrete phone factory object (`TelephoneFactoryImpl`).

Workshops 7 and 8 focused on the enhancement of the VMS server. In the original design, the voicemail system uses a fixed number of mailboxes generated when the system starts up, and all changes to mailbox greeting and passcode, as well as messages saved in the mailboxes will be lost after the system shuts down. A data storage layer was added in Workshop 7 to persist all mailboxes and their contents, which were then stored in a relational database. The data storage layer used the data object access (DAO) pattern [5], and communicated with the database using JDBC. Workshop 8 assumed that only one caller (i.e., a user who interacts with the VMS system via the phone client) can access a mailbox at a given time; any other concurrent attempts will be

blocked until the active caller ends the call. Workshop 8 demonstrated typical synchronization and coordination mechanisms that can be used to solve such a problem. Features included in these workshops provided working examples for the students to design their own projects.

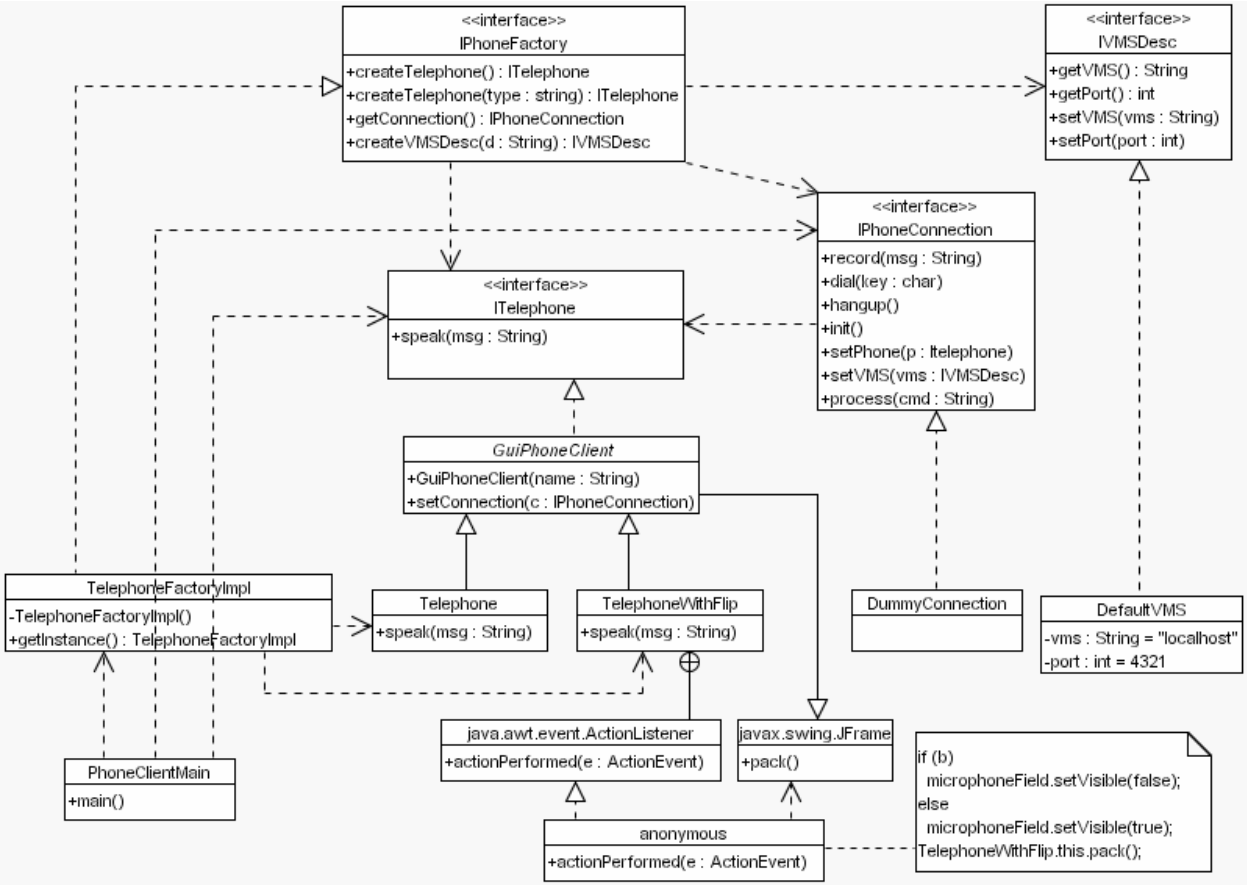

Figure 6 – Design Class Diagram Used in Workshop 6

Discussions and Conclusions

The series of workshops fit the purpose of the Software Engineering course very well. They can bridge the gaps between lectures about concepts and principles involved in the subject and mastering of the principles and related skills in a team-based term project. Individually, each workshop provides an opportunity for the students to have hands-on exercises on what they have just learned from the class. Working as a whole, they help to set up the context of the development of a moderately complex software system. They set up meaningful scenarios for the participants to consider design issues such as system decomposition, architectural flexibility, using existing software components or writing code of their own, and choosing from alternative technologies. They can complement the simple and isolated examples used in lectures, and facilitate discussion of the course materials to the depth and extent unmatched by what lectures and normal labs can bring about.

Students showed great interest in participating in class discussions and hands-on exercises during the workshops. They were enthusiastic in learning to use modeling tools [13] and IDE [9], enjoyed

completing the telephone interface that can flip and really speaks. Results showed a much better grasp of the OOAD activities than previous classes: all the teams could use UML diagrams in their design; two of the three teams developed a client-server application and used a database for data persistence. During their presentations and final system demonstration, students frequently referred to scenarios of the workshops as their shared common knowledge. One team of four students developed a Web-based application, March Madness, which is designed to facilitate "a bracket competition during the NCAA Men's College Basketball Contest".

The key to success when using this approach is the selection of the case application and the topics to be emphasized in the workshops. The workshops set up a context for discussion the practical aspects of the subject and provide a format that can combine in depth discussion and hands-on exercises. The main purpose of the workshops is not trying to let the students to go through all the details and build the system in class, but to demonstrate how the pieces fit together. The ordering of the topics and the scope and level of detail of the hands-on exercises can thus be tailored to best fit the needs of a particular class.

Acknowledgements

VP for UML 5.0 Standard Edition is available to Mercer University through the Visual Paradigm Academic Partnership Program. Thanks to Dr. David Cozart and Dr. Randall Harshbarger for their encouragement and feedback. We would also like to thank the anonymous reviewers, who provided constructive comments that improved the quality of the paper.

Bibliography

1. Association for Computing Machinery (ACM). Computing curricula 2001 Computer Science. http://www.acm.org/eductation/curricula.html
2. Beck, K. *Test-driven development by example*, Addison-Wesley, 2003
3. Braude, E. *Software engineering: an object-oriented perspective*, Wiley, 2000
4. Bruegge, B and Dutoit, A. *Object-oriented software engineering using UML, patterns, and Java*, 2nd ed., Prentice Hall, 2004
5. Eriksson, H. et al. *UML 2 Toolkit*, OMG Press, 2004.
6. FreeTTS 1.2. http://freetts.sourceforge.net/docs/index.php
7. Gamma, E. et al. *Design patterns: elements of object-orient software*, Addison-Wesley, 1995.
8. Horstmann, C. *Object-oriented design and patterns*, 2nd ed., Wiley, 2006
9. NetBeans IDE. http://www.netbeans.org/
10. Pressman, R. *Software engineering: a practitioner's approach*, 6th ed., McGraw-Hill, 2004
11. Sommerville, I. *Software engineering*, 7th ed., Addison-Wesley, 2004
12. Stiller, E. and LeBlanc, C. *Project-based software engineering*, Addison-Wesley, 2002.
13. Visual Paradigm International. *Visual Paradigm for UML 5.0 Standard Edition*. http://www.visual-paradigm.com/