International Journal of Business, Human and Social Sciences
ISSN: 2517-9411
Vol:12, No:8, 2018

Understanding the Programming Techniques Using a Complex Case Study to Teach Advanced Object-Oriented Programming

M. Al-Jepoori, D. Bennett

***Abstract*—**Teaching Object-Oriented Programming (OOP) as part of a Computing-related university degree is a very difficult task; the road to ensuring that students are actually learning object oriented concepts is unclear, as students often find it difficult to understand the concept of objects and their behavior. This problem is especially obvious in advanced programming modules where Design Pattern and advanced programming features such as Multi-threading and animated GUI are introduced. Looking at the students' performance at their final year on a university course, it was obvious that the level of students' understanding of OOP varies to a high degree from one student to another. Students who aim at the production of Games do very well in the advanced programming module. However, the students' assessment results of the last few years were relatively low; for example, in 2016-2017, the first quartile of marks were as low as 24.5 and the third quartile was 63.5. It is obvious that many students were not confident or competent enough in their programming skills. In this paper, the reasons behind poor performance in Advanced OOP modules are investigated, and a suggested practice for teaching OOP based on a complex case study is described and evaluated.

***Keywords*—**Complex programming case study, design pattern, learning advanced programming, object oriented programming.

I. Introduction

OOP was formally introduced in 1967 by Ole-Johan Dahl and Kristen Nygaard when they created the Simula language at the Norwegian Computing Center. They introduced a new way of modeling and programming complex tasks. Nygaard identified that many people claimed to be experts in OOP and in teaching OOP. However, Nygaard formulates one of his favorite messages: "To program is to understand" and promoted the use of complex examples to teach object-oriented design and programming right from the start [1]. He used the example of a crowded restaurant as a case study with many interacting objects.

The problems of teaching students object oriented design and programming is still difficult. This research investigates the performance of students who have already done three different programming modules, yet still struggle to grasp the concepts of OOP and its advanced features such as multithreading, and the use of Design Patterns.

This paper is structured as follows: it begins by introducing some of the previous and current treads in teaching OOP, and presents one of traditional case study used in teaching OOP. Section III presents the proposed practice for teaching OOP at the university level. In Section IV, the teaching plan is evaluated, while the final section provides a summary and conclusions.

Muna Al-Jepoori is with the Canterbury Christ Church University, United Kingdom (e-mail: muna.al-jepoori@canterbury.ac.uk).

II. Programming and Technology

A. Trends in Teaching OOP

According to Madsen [1] in the early 2000s, there were hundreds of books published on OO. Nygaard found that most of those books did not do a good job in teaching the fundamental concepts of OO, so he started a project named Comprehensive Object-Oriented Learning (COOL) to develop first-class teaching material on OOP.

Currently, some of the widely used books in higher education are using different approaches to teaching OOP. The most popular approaches are based on whether to start teaching Classes and Objects orientation from the start of the course or postpone these concepts to a later time. These approaches are namely identified as the "early objects approach" and the "late objects approach", respectively. Authors of textbooks have not actually agreed on the best approach, as it depends on the particular course and the whole plan of the teaching programming in a certain institution. For examples, the Deitel & Deitel textbook [2] on OOP has two different versions, one is the objects-early approach and the other adopted the objects-late approach. Barnes & Kolling have identified their book [3] as "object first" to denote the early introduction of objects.

Simple examples dedicated to presenting particular concepts are used in teaching different programming languages. However, case studies have long been an important component of teaching. With the use of a case study, programs are written to demonstrate a wider understanding of several concepts in programming languages. Many textbooks provide case studies as an optional programming challenge. Unfortunately, with time limitation, such case studies are not normally attempted and some textbooks have removed challenging case studies, or replaced them with smaller case studies. For example, one well-known case study used in teaching OOP is the Lift Simulation. This was introduced in the early versions of Deitel & Deitel programming textbooks such as [5], but has been removed from later versions.

As developing complex software systems from scratch is expensive, time consuming, and error-prone, Software Reuse became the obvious solution that contributes to easy software development. The use of Design Patterns has made it easier to

create reusable software components and provide for the production of software that can directly be reused or that are open for extension to add new functionality to the software. The real movement in software design patterns started in 1994 after the publication of the book "Design Patterns: Elements of Reusable Object-Oriented Software" [6]. Design Patterns aid the development of reusable software [7]. Therefore, it is important that Design Patterns are taught as part of teaching programming.

B. Using Lift Simulation as a Case Study and Eclipse as IDE

Nevison & Wells [8] stated that:

> "Well-chosen case studies can provide the complexity to motivate object-oriented Programming while also providing a context where concepts can be presented in a reasonably simple setting within the more complex environment."

Many textbook and software developers such as Karg [9] have recommended the case study of "Elevator Simulator" design and implementation. This cases study can be used in the different stages of learning; it can be made simple enough to implement basic concept, and complex enough to implement advanced program with diverse requirement that involves using many advanced features of OOP and Design Patterns.

Since the introduction of this case study in the 1990s, it has been used in many textbooks such as that of Deitel & Deitel [5] and the book that introduces the Greenfoot Integrated Development Environment (IDE) [4].

Different IDEs provide different way of implementing the Lift Simulators. Greenfoot for example, makes it easier to implement a Graphical User Interface (GUI) and provides help in understanding the different concepts in programming. Eclipse and Microsoft Visual Studio on the other hand, helps programmers learn the basics of programming without relying on visual GUI; such IDEs are currently the GUI of choice for teaching Java at the university level. IntelliJ has been recommended by the product recommendation community Kearney [10] as the top IDE for Java development; however, IntelliJ can be costly. Eclipse is free and it uses a custom compiler, which is often faster than the normal Java compiler, especially for incremental compilation. Eclipse is more suited for real world applications where high performance is required, it also many useful plugins such as ObjectAid for creating UML diagram and WindowBuilder for using GUI components. Bluej [3] and Greenfoot [4] are mainly used at schools and colleges as they make it easier to visualize objects and their interaction. This helps in early introduction of objects. Greenfoot also provides high-level classes that help in quick development of Games and animations.

III. Practice for Teaching OOP at University Level Courses

Currently, there are four modules at the university, in which programming is taught at the different levels as follows:

Year 1: There are two modules in year 1; Introduction to programming module, and Application Development module - the aims are to develop the students' understanding of the fundamental programming concepts required by all programming paradigms. As well, to provide students with the problem-solving skills to design, implement, test and debug a software solution to a given simple problem. The modules also prepare students to take a specification and implement a reasonable solution using stepwise refinements and identification of common elements to create functional decompositions.

Year 2: OOP - the major aim is to demonstrate a practical and theoretical understanding of the Object-Oriented paradigm of programming. This module also includes data structures, and more complex algorithms.

Year 3: Advanced Programming - the major aim is to understand the software design process using the mega pattern Model-View-Controller (MVC) and other design patterns, implement advanced software programming features, employ efficient execution based on sound algorithmic design, and to produce design documentation and carry out software testing.

Based on the above strategy, it has been noted that many of the students attending Year 3 Advanced Programming find it difficult to understand Design Patterns. Many students find themselves unable to develop complex algorithms, and implement advanced programming features. Therefore, it seems possible that an early objects approach may be more appropriate to follow than late the objects approach. As the evidence for and against the adoption of early-objects is non-conclusive, the late-objects approach is used for other reasons in the institution. However, as this does not solve the issue with students struggling with the topics in the class, a different approach to the compartmentalised teaching of concepts as separate topics is used.

The following new strategy has been adopted to solve the problem with advanced programming as it stands. The intention is to consolidate the basic knowledge the students have acquired in the first two years of the degree program, and build on this basic knowledge to prepare students for the job market. This involved:

- Providing a comprehensive revision about the basic syntax of Java as the student used C# in the first two years.
- Use one of the well-known case study that has been used in many text books as optional exercise - in this case the Lift Simulation, a.k.a. "Elevator Simulation" in American text books. This case study can be made simple enough to demonstration basic concepts in Java, and complex enough to include most of the advanced features of Java including Multi-threading and Design Pattern. Kölling provided partial solution for this case study in [4], the view of implementation in Greenfoot is presented in Fig. 1. Deitel and Deitel also provided an old sample solution of such simulation using GUI and text messages in the early editions of OOP books published before 2000 in both Java and C++ textbooks [5]. The solution provided for Lift Simulation was written with code duplication, and design pattern were not fully implemented. The case study

has been removed from Deitel & Deitel books in the later versions of the C++ and Java books. Therefore, we use the assignment of List Simulation of special lift environment and functionality with a particular scenario; students have to study the delivered lectures, and follow lecturer guidelines to be able to design and implement a suitable solution for this particular scenario.

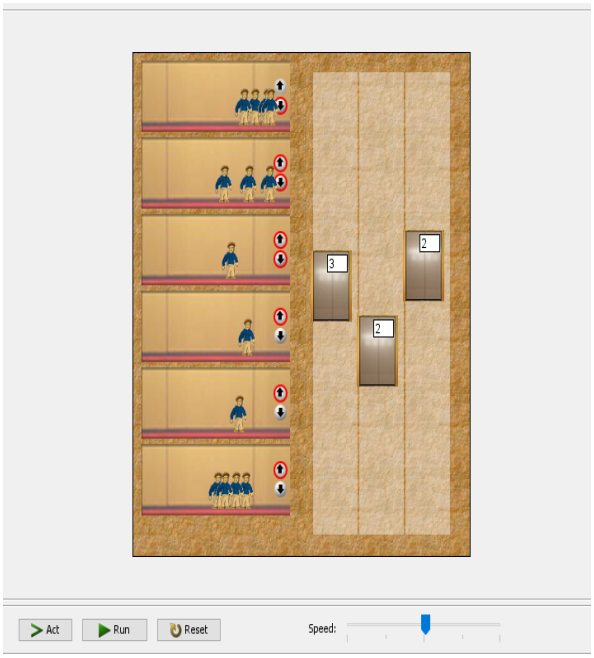

Fig. 1 Lift Simulation using Greenfoot- Barnes and Kolling [4]

- The implementation of Lift Simulation was done throughout the module week-by-week in four contact hours every week; two hours of lecture sessions that was followed by a further two hours' tutorial session. Table I shows the outline of the work done using the Lift Case Study.
- The students were required to design the class diagram, then start coding, and testing the project assuming a building of two floors only. They were required to provide text view reporting on every event triggered by the different actor classes in the model, and as well, GUI view to provide an animated view of the simulated lift. The solution uses the Model-View-Controller (MVC) architectural design pattern. As MVC is a mega pattern, this involved at least using Observer, strategy and composition pattern.

The students were asked to work week-by-week, following the project plan and in the two-hour tutorial session every week, formative feedback was given to each student individually. Office hours were efficiently utilized by students for discussion and extra formative feedback as well; these office hours were mainly used by students who required extra help.

TABLE I
IMPLEMENTATION PLAN

Week	*2-hours tutorial session every week mainly used to work on case study in class*
1	Revising Java basic syntax including control structures, data structures and interfaces
2	Study assignment brief and produce Class Diagram
3	The initial Person class implementation and testing using studs
4	MVC project. Coding initial observer for the text based view, tested for Events triggered by objects of class Person
5	Using Multi-threading for Person Class
6	Programming the lift class, Door class and Button class
7	Synchronized objects and methods- controlling events coding and testing
8	Using Singleton and Iterator design pattern
9	Simple GUI for Controller to generate more person objects
10	Work on Simulated GUI
11	Testing and documentation

IV. Evaluation and Analysis of Teaching Plan

Although the 2016-2017 assignment was different from the 2017-2018 assignment, as it did not completely rely on OOA and coding, nearly 30% of the assessment included writing an essay like report. However, evaluation is made by comparing students' performances with the previous year's assessment results as per Figs. 2 and 3 that show many important improvements.

Fig. 2 shows that in 2017-2018:

- The number of student who fail the module are much lower than previous year.
- The number of non-submissions is also reduced.
- The percentage of students who scored a First is 21.1 much higher than the 14.6 in the previous year.
- The percentage of the total number of students who passed is 75%, which is higher than the 65.9% of last year.

Fig. 3 shows that in 2017-2018:

- The Median is exactly 50%.
- A larger distribution of marks as the maximum mark is much higher than the upper quartile.
- The average mark is higher than the previous year average.

Although the T-Test, did not show a significant improvement in the results, the assessment in 2017-2018 was far more challenging than the assessment in 2016-2017, hence, the smallest improvement in the result is regarded as a significant achievement.

In general, the outcome is that, students' performance was remarkably improved; they produced high quality design, implementation and documentation. Many students went above the required implementation as they enjoyed the fact that they were creating the simulation in an incremental way, testing each unit of code and experiencing real-world practices in software development.

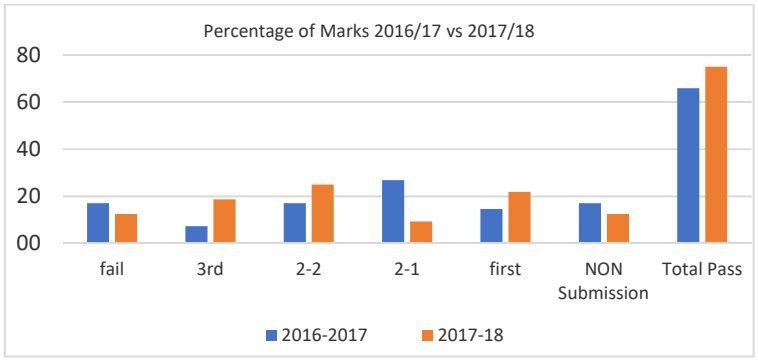

Fig. 2 Percentage of Marks 2016-2017 vs. 2017-2018

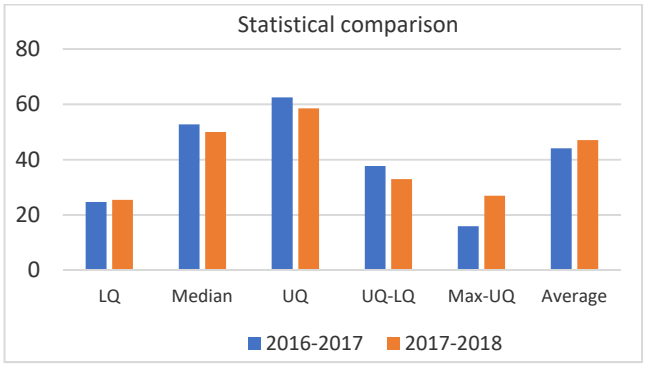

Fig. 3 Statistical Comparison

V. Conclusion and Future Work

This paper reported on the authors experience in teaching OOP using Design Pattern and advanced OOP features. The use of case study to incrementally implement the required features has been very successful. Students had clearly understood the problem at the beginning of the module; this created an environment that encouraged students to implement the material they learned week by week. The experiment of changing the way OOP and Design Pattern concepts was introduced and the way formative feedback was given to every student, has made it easier to identify design and coding mistakes at early stages, and help the students improve their work. However, this was a difficult and time-consuming task for the lecturer to check students' attempts every week and provide formative feedback. Most review was done outside tutorial sessions to prepare feedback and discuss it with the students. Changing the way the programming module is delivered in the first two years of the degree programme to objects-early may help this group of students understand the advanced concepts, and helps lecturers by reducing the frequency of feedback required. However, even with a new programme design, this may not be possible owing to the shared nature of the first year modules across many different programmes, most of which do not lead to a pure software development pathway. For example, our Computer Forensics and Security students do more scripting-based development in later years within the Linux environment, where OO is not part of their required skillset. This means that the most important element is to develop the students' algorithmic thinking in the early modules and an objects-early approach may not support this as well. However, schools in the UK have recently changed their programs, they now include a compulsory Computer Science stream, we may find that the algorithmic skill set becomes part of the students' capabilities before joining our university and an early objects approach may be more suitable. In the meantime, the approach undertaken may be useful in alleviating issues with the final year understanding of more complex topics.

References

[1] Madsen, O. (2002), In Memory of Ole-Johan and Kristen Nygaard, Journal of Object Technology, Vol. 1, no. 4, September-October 2002
[2] Deitel, P., Deitel, H., (2015), Java How To Program (Early Objects) (10th Edition), Pearson (2017), ISBN-13: 9780133807806.
[3] Barnes, D., Kölling, M., (2016). Objects First with Java A Practical Introduction using BlueJ. Sixth Edition, Pearson, 2016. ISBN (US edition): 978-013-447736-7. ISBN (Global Edition): 978-1-292-15904-1.
[4] Barnes, D., Kölling, M., (2016), Object-Oriented Programming in Java with Games and Simulations, Second edition, Pearson, 2016, ISBN-10: 013-405429-6, ISBN-13: 978-013-405429-2.
[5] Deitel, P., Deitel, H., (2000), C++ How To Program, Pearson.
[6] Freeman, E., Robson, E., Sierra, K., & Bates, B. (2004), Head First Design Patterns, O'Reilly Media
[7] Gamma, E., Vlissides, J., Johnson, R., Helm, R., (1994), Design Patterns Elements Of Reusable Object-oriented Software, Addison-Wesley.
[8] Nevison, C., Wells, B., (2004), Using a maze case study to teach: object-oriented programming and design patterns, Proceedings of the Sixth Australasian Conference on Computing Education, p.207-215, January 01, 2004, Dunedin, New Zealand.
[9] Karg, S., (1996), available from,

http://www.angelfire.com/trek/software/elevator.html, last accessed on 20/12/2017.
[10] Kearney, S., (2017), available from: https://www.slant.co/topics/10229/~ides-for-java-on-windows, last accessed on 20/12/2017.

Muna Al-Jepoori holds an MSc degree in computer science (1982 - Aston University-UK) and a PhD degree in computer science (2009 -Bradford University-UK), she worked as manager of several computer centres before started working in higher education. In 1997, Muna started lecturing at Amman University in Jordan, Sultan Qaboos University in Oman, and University of Kent in UK and currently she is a Senior Lecturer at Canterbury Christ Church University – Computing since 2014.

David Bennett holds a BSc (hons) Computer Science from the University of Exeter and a DPhil from the University of York. He is currently a Senior Lecturer at Canterbury Christ Church University. Previously he worked in industry for Philips Sound and Vision and Planit International in Software Development.