Translating Problem Domain into Object Oriented Programming: What is the Problem?

[1]Lawal Bashiru, [1]Olowonusi Victor Oluwaseyi, [1]Adunlejika Aderogba Joseph & [2]Mohamed M. Elzahaf

[1]Department of Computer Science, Federal College of Education (Technical) Gusau –Nigeria.
[2]Department of Computer Science Faculty of Medical Technology, Libya – Derna Shiha Sharakia Derna - 1149

Received May 21, 2017 Accepted June 15, 2017

ABSTRACT

Object-Oriented development starts with Object Oriented analysis where problem domain is modeled as collection of objects. However,when some programmers modeled the real world domain to be represented in the program, the beginning of their troubles starts from their slickly cognitive knowledge in programming and lack of problem solving abilities. They imagine the model and coded it without explicit analysis and design. Their problems may be generally attributed to their lack of problem solving ability. They don't know how to create abstraction, because they don't know how to divide problem domain into object types, how to model the relationships and attributes with behaviour of each type, mainly due to their inability to create abstraction. This paper builds on previous work, trying to understand what problems programmers have when translating problem domain into Object oriented Programming (OOP), as well as possibilities of dealing with them. The emphasis is on beginning programming students. This and other causes to student difficulties are discussed in this paper. Some possible solutions such as curriculum and pedagogical review, dynamic teaching techniques, creating an integrated supportive techniques and environment are suggested to reduce the problems.

***Key words*:** Object Oriented Programming

INTRODUCTION

Many reports of the experience of those attempting to describe problems confronting noviceprogrammers in learning to program include a long list of problems with different methods used. This has cumulated into a range of cognitive theories for high attrition rates such as: *inability to read and understand code* (Lister, Adams et al., 2004); *inability to problem – solve* (McCrecken, Almstrum, et al., 2001); *the difficulty of understand the purpose of programmers and their relationship with the computer; difficulty in grasping the syntax and semantics of a particular programming language* (Robin, Rountree et al., 2003); *misconceptions of programming constructs* (Soloway& Spohrer1989).

Researchers have proven that the OO paradigm is a more appropriate technique to solve the inherent complexities of software systems, than the traditional procedural paradigm in recent years (e.g. Johnson and Moses 2008).The important reasons for this assertion according to Okur (2006) are related to OOP having some conceptual edge like; "Abstraction, Inheritance, Polymorphism and Encapsulation", which enhance system maintenance and reusability. This benefit has in some ways, put extra tensions on both programming tutors and students of the subject.

Also, it is probably true that programming with object orientation is harder than the procedural style (Kolling 1999a, Donchev &

Todorova2008). Students' performance in OOP is a matter of concern and programming courses in general have a high attrition rate (Govender & Graysm 2006; Lahtinen et al., 2005). This may due to the fact that the teaching and learning of OOP is multidimensional and complex which is associated with many difficulties (Okur 2006). And perhaps the "novelty" of the subject is too much for some students (if it hasn't been taught at schools); or perhaps the idea of writing code to be executed by a computer is too alien. For OOP, there is the additional conceptual gap of code being executed by collaborating objects, rather than a monolithic program. Teaching OOP requires the right language, a good pedagogical plan and appropriate resources (hardware & software, teaching materials & tools, lab assistants). Students need to be encouraged and supported. Getting it all right is not easy (Lawal 2012)!

Now why do students face more difficulties in their attempt to program in OOP? Or to be more precise, what are the problems thatstudents encounterwhen translating problem domain into OOP? *The problem domain is the subject area of a particular programming effort, such as "accounting", "elevator control", or "word processing".*Unfortunately, not much work has been done to find answers to these questions!

This paper builds on previous work, and it aims at providing answers to the question: what problems do programming students have when translating problem domain into OOP? The emphasis is on beginning programming students. An important source of information for the study has been a very recent study conducted by the researcher in which, I investigated the problems encountered by tutors and students in teaching and learning of OOP. Another vast source for the study is a research relating to the study of students' problems in learning to program, written by Jan ErrikMoström (2011). Also, the study conducted by Gomes and Mendes (2007) titled: *Learning to program: problems and solutions* was an excellent reference material for the study. In addition, other related studies containing valuable information are used and are well acknowledged.

Overall, this paper can be seen as a summary of my attempts to better understand the problems that students experience when translating problem domain into OOP. Some possible solutions are also provided, so as to reduce the problems and improve the programming skills and culture of beginning programming students.

THE PROBLEM

1. Students' cognitive abilities and attitude

When programming students set out to design a program in OOP, experience have shown that majority of fresher in OOP do not have the ability to create abstraction. They are often confronted with decision problems such: (i) how to phrase program specifications to make class discovery, behaviour, interface and interaction more apparent. (ii) are there mental tools to help in writing specifications and interpreting them to better discover the classes involved, interfaces to be exposed and how they interact with each other? Their problems may be attributed to their inability to model solution from a given problem domain. As a result students experience difficulties in completing simple programming problems (McCracken et al., 2009; Lister et al., 2004).

In another study conducted by Moström et al., (2008) on concrete examples of abstraction, as manifested in students' transformation experiences, it was reported that abstraction plays an important role for many students

when learning to program. However, students do not know how to create abstraction which is a very important skill for allowing students to view a subject from a number of different angles and of gaining practical experience (Moström et al., 2008).

Because of their inability to create abstraction, Moström et al (2008) observe that it becomes difficult for students to model a real life situation with a program. They often find it difficult to determine which class should be the client of another, which class types, should be returned from methods, and which ones should be parasitized. In the end, they get confused to tell from a problem domain, which class should interface, and which ones should be separated by interfaces (Lawal 2012).

2. Class/Object misconception

The study conducted by Sanders et al.,(2008) to study student understanding of OOP as expressed in Concept Map shows that students seem to associate class and behaviour more strongly than class and data. The result also reveals that it is unclear whether students whether students really understand the connection between class and object (Sanders et al., 2008). Students find it most worrisome to divide the problem domain into types of objects, modeling the relationship between the types, and model the attributes and behaviour of each type when they have to fabricate classes that don't exist in the real world (Lawal 2012).

3. Lack of problem solving abilities

Freshmen in programming, including novice programmers spend a lot of time learning the basics of OO design. Some have overdosed on learning about abstraction, encapsulation, programming by contract (i.e interfaces), inheritance, composition, cohesion, coupling etc. however, nobody ever talks about decomposing a problem finding a solution and modeling the solution utilizing the aforementioned OO design (Lawal 2012). Many programming tutors are of the opinion that a lot of students are able to learn the syntax and the functionality of the program but run into difficulties when solving complex task (e.g real-world exercise in OOP) because decomposition of a problem does come not naturally to some (Lawal 2012). Most students lack the ability to break problems into modular tasks, interfaces and decompose problems into sub problems/tasks.

Gomes and Mendes (2007) opines that the primary cause to the difficulties many beginning programming students encounter in programming is their lack of generic problem solving skills. Gomes and Mendes state that problem solving requires multiple abilities that students don't have, namely: *problem understanding; relating knowledge; reflection about the problem and the solution*. For OOP, decomposition is the key and it seems to get the least focus (Lawal 2012)!

Robins et al., (2003) observe that the difficulties experienced by novice programmers to solve programming problems seems to be that they are limited to surface knowledge of program and generally approach programming "line by line' rather that at the level of bigger program structures. Their problem is also attributed to the fact that they spend little time in planning and testing code and often fail to apply correctly those basic concepts they have obtained (AlaMukta 2004).

4. Lack of explicit analysis and weak thought process

Many programming students are unable to translate problem domain into OOP because they wrongly misinterpret the problem statement and others simply because they are

too anxious to start writing code and don't read and interpret correctly the problem description (Gomez and Mendez 2007).

In a qualitative study conducted by Kaasboll et al., (2004), which was designed to observe students mode of learning OOP in an introductory programming class, students were found to cope reasonably well with the OO concepts. However, when modeled the real world domain to be represented in the program, they imagined the model and coded it without explicit analysis and design (Kaasboll et al.). The main reason for beginning programming student' wrong imagination in modeling according to (Lawal 2012), is their deficiencies in concrete methods of thinking, a very weak thought process, slicklyways of expanding a problem and lack of explicit analysis and design when approaching a problem in order to fully realize a solution before trying to model it with classes and objects (Lawal).

5. The initial learning phase and pedagogy

The traditional teaching methods students were subjected to during their learning of OOP have larger impact on the problems they face during programming (Gomez and Mendez 2007).Programming, and most especially OOP, involves several dynamic concepts that may require visualization tools to supportits comprehension (Guo 2006). Unfortunately, this practice, these days are taught through static means such as: projected presentation, verbal explanations, diagrams, blackboard drawings, texts, and so on, which invariably have negative impact on students' comprehension on programming education (Gomez and Mendez 2007).

Kaplan (2008) observes that most research conducted on programming education has not informed the way programming courses are being delivered. As a result, programming is largely taught the way it was taught some 60 years ago. Gomez and Mendez (2007) state that, the traditional methods of teaching programming these days do not seem adequate for many students' needs. Reasons such as: teaching are not personalized; teachers strategies don't support all students learning styles; the teaching of dynamic concepts through static materials; teachers are more concentrated on teaching a programming language and its syntactic details instead of promoting problem solving using a programming language; influence problems that students encounter when programming (Gomez and Mendez 2007).

Also for a long time, OOP was considered an advanced subject and was taught late in the curriculum. Anecdotal evidence (e.g Stroustrup 1994) indicates that it takes students more than one year to switch their mind-set from a procedural to an object-oriented view of the world. The path to program with object orientation through procedural programming is unnecessarily complicated (Kolling 1999). Students find it difficult to conceptualize the philosophy of object orientation after their mind had been used to procedural (Lawal 2012). Lawal suggested that "object early" pedagogy seems reasonable, though no concrete evidence. The idea is to remain always within the OOP domain, avoiding the confusion that might result from switching to OOP after some "procedural" introduction (Lawal 2012).

6. The nature of Object Oriented Programming

OOP is a very complex programming paradigm, which requires adequate understanding of abstract concepts and this is one of the reasons why many programming students encounter difficulties in programming with this style of programming (Lawal 2012). For example, in OO design, certain components just don't need to know

that others exist. It has to be "elegantly designed" as they say.

Also, design in object orientation requires a programmer to identify the fundamental objects of the problem domain, i.e the things involved. He/She then classify the objects into types by identifying groups of objects that have common characteristics and behaviours. Assigning class responsibilities, creating well defined interfaces in OO design is difficult, especially when we are to model things that do not exist (Stroustrup 1991). Students' lack of expatriate and skills to cope with the nature of the subject is another major obstacle they face in representing real world entities in OOP (Lahtinen et al., 2005).

In addition to grouping of objects that have common characteristics and behaviours, students need to identify the right classes, assign to them the correct state variables and methods, and workout the process by which to achieve the required results. All these skills are required for developing solutions to problems in OOP, which most do not have (Lawal 2012). Most beginning programming students also lack the basic OOP skills including modeling/design, procedural coding and general program development from the beginning. As a result, students cannot tackle and solve simple programming problems involving several classes (Lawal 2012).

7. Object Oriented Programming Language and Environment

Many beginning programming students face the problem of wrong initial introduction to using programming language and environment in learning OOP. This is one of the reasons why some of these students find OOP paradigm difficult. Kolling (1999a) states that it is not the object orientation in principle that causes the problem, which students experienced during the introductory programming classes, rather, the tools available to teach it. Programming languages used are too complex and programming environments, if they exist at all are confusing (Kolling, & Rosenberg1996a).

Some programming languages and environment used for delivering OOP were really developed for expert software engineers, making extremely difficult for beginning programming students to cope; others not developed at all but grew out of historic coincidences (Kolling 1999a).

PROPOSED SOLUTIONS

This section describes how to reduce some of the problems highlighted above as confronting students in translating problem domain into OOP. In my own humble opinion, this can be achieved through: an evolutional review of introductory programming curriculum and pedagogy, dynamic teaching techniques and the development of and utilization of programming supporting centre, which will support and help to boost students' programming skills and culture. To achieve these general objectives, below are the proposed solutions:

- **Curriculum and pedagogy**

There is need to review the curriculum and mode of delivery of OOP at introductory level. This can be achieved in the following ways: first, the introductory course should be of 'object early' approach. This method supports the introduction of OOP at the beginning stage, but should include basic procedural concepts, design issues, methodical program development, etc. Later courses can cover data abstraction and algorithms, and present OOP in a problem-solving context. Further advanced courses could add graphical interfaces, web technologies, concurrency and the process of specification and the

process of specification and validation against specifications. As far as the initial introductory course is concerned, it is best to integrate rather than prioritize (Lawal 2012).

- **The teaching approach**

OOP should be delivered so as to cover both OO and procedural concepts in an integrated way. Neither side should be considered expendable, even if thought to be "harder". Modeling & design should be taught alongside the other procedural/OOP and general program development skills in an integrated way.

The teaching of OOP at introductory level should be made dynamic, visualization based approach in which contents and tasks are presented in a systemic way of learning to program. The emphasis is to ease the introduction of basic object oriented programming concepts and principles, and to engage students in active learning achieved by thinking actively, and think carefully before coding while analyzing problems and designing solutions leading to an effective acquisition of coding proficiency as supported by Guo (2006) and Lawal (2012).

OOP is amenable to "visualization", starting with graphical depiction of the classes and their relationships, using graphical applets to show code effects instantly; and using other means to "visualize" interactions between classes.

It is also a good idea to provide a series of worked examples for students to study and modify progressively adding new concepts. Research has shown that students begin to grasp OOP when they study examples with classes that represent familiar real world entities, e.g Persons, Cars, Vending Machine, etc (Lawal 2012).

- **Development of learning context and environment in programming**

There is need to develop a holistic approach to learning context and learning environment that will support students' programming experiences and practice. A typical example is the recently developed*NoobLab Online Learning Environment* byNeve et al., (2013). This system motivate students in their learning of programming; conventional lectures were minimized; instead, students were encouraged to spend the majority of their time engaging in practical explorations. An initial deployment of the system on entry programming courses at Kingston University - United Kingdom, evidently shows an improvement in programming skills (Neve et al., 2013). The system is now expected to become the programming developing tool for teaching introductory programming in the School of Computing and Information Science at Kingston University, London - United Kingdom.

Another milestone of success was recorded by Guo (2006), when she evaluated students' comprehension and satisfaction in learning OOP in a process based approach, using logbook as a visual learning aid to teach OOP at introductory programming class. The approach was aimed at engaging and motivating students to record (hand-written) programming activities, which assist them to think logically while analyzing problems and designing solutions. Guo (2006) observes that though students see the logbook to be time consuming, the approach has helped students tremendously to follow a sound programming process which helped construct their program in a systematic way and that the effect is worthwhile.

In complementing the efforts of both Neve et al (2013) and Guo (2006), a novel approach with associated supportive techniques is proposed to boost students' programming skills. The said approach must incorporate

e ISSN 2348 –1269, Print ISSN 2349-5138
Cosmos Impact Factor 4.236

the following characteristics, which will help students to:

- think logically while analyzing problems and designing solutions
- develop a systematic programming practice
- be aware of personal weakness, strength and ability
- track progress from actual evidence

- **Programming tutor and student responsibilities**

Programming tutor should establish a means of providing permanent student supervision. The ideal move is to have tutor to follow students' evolution, clarify doubts, and propose problems and activities. This seems to support Gomez and Mendez (2007) proposals.

Another important responsibility of the teacher is to prevent that may lead students to give up or loose confidence in programming. To achieve this, a programming support centre can be established to provide the needed support and motivation for the programming students.

The students on their part should acquire basic OO skills including modeling/design, procedural coding and general program development, within their introductory programming course. They should be able to tackle and solve simple programming problems including several classes. They should be aware that learning programming effectively cannot take place through memorization and study programming through text book is not enough to gain proficiency in programming (Gomez and Mendez 2007). Proficiency in programming however, requires continuous and intense practical engagement.

- **The programming language and environment**

Java is perhaps the easiest first language to learn (Lawal 2012). Perhaps, the basic can be taught fairly quickly while the advance features (e.g graphical interfaces and web technologies) can be taught in advanced programming courses later. The BlueJ IDE is ideal for introductory OOP using Java (Lawal 2012, Kolling& Rosenberg1996a). For later courses, students have authoritative online materials on the Sun/Oracle website (e.g the full APIs, tutorials, etc)

CONCLUSION

It is an established fact that beginning programming students do face enormous challenges and difficulties in translating problem domain into OOP and class/object misconceptions, lack of problem solving abilities, lack of explicit analysis and weak thought process among others, are some of the important reasons that cause these problems. It is recommended that programming curricula and pedagogy most importantly at entry level needed a massive review in order to meet the programming demands of beginning programming students.

Teaching OOP requires the right language, a good "pedagogical" plan and appropriate supporting resources (hardware & software, teaching materials & tools, lab assistants). Students need to be engaged, encouraged and supported (Neve et al., 2013; Lawal 2012; Gomez and Mendez 2007; Guo 2006).

Furthermore, an introductory OOP course presents more challenges to the tutor, as there are important "new" concepts that needed to be tackled earlier. Based on this revelation, I intend to extend my research on some important concepts, which unfortunately are receiving the least

attention by most programming tutors such as: *information hiding, coupling and cohesion.* The ultimate aim is to critically examine how these concepts can help to boost the understanding of OOP philosophy of beginning programming students at their early stage of learning to program with OOP. Later stages will focus on action research on learning techniques, such as collaborative, and so on.

References

1. Ala-Mutka, K. (2004) 'Problems in Learning and Teaching Programming – a literature study for developing visualization in the Codewitz-Minerva Project' [online] available from <http://www.cs.tut.fi/~edge/literature_study.pdf> [January 22, 2014]

2. Donchev, T. & Todorova, E. (2008) '*Object oriented programming in Bulgarian Universities informatics and Computer Science curricula Informatics in Education, 7(2), pp.159-72*

3. Govender, I., & Grayson, D. (2006) 'Learning to program and learning to teach programming: A closer look'. *ED-Media 2006 proceedings:1687-1693.*

4. Gomes, A. & Mendes, A. J. (2007).learning to program – difficulties and solution. In Proceeding of the 3rdInternational Conference on Engineering Education – ICEE2007, Coimbra, Portugal.

5. Guo, H. (2006) 'Visual Materials and Learning aids in Teaching Object Oriented Programming'. Proceedings of the 7th Annual Conference of the Higher Education Academy Information and Computer Sciences (29th – 31st August, 2006), Trinity College, Dublin, pp.64-69

6. Johnson, R.A. & Moses, D.R. (2008) 'Object-first vs. Structures-first approaches to OO Programming education: an empirical study'. *Academy of Information and Management Sciences Journal,11(2), pp.95-103*

7. Kaasboll, J., Berge, O., Borge, R. E., Fjuk, A., Holmboe, C., Samuelsen., T. (2004). Learning Object oriented Programming. In Proceeding of 16th Workshop of Psychology of Programming Interest Group. www.ppig.org.

8. Kaplan, R., M. (2005).Teaching Novice programmers programming Wisdom. In Codewitz- Minerva Project.Kutztown University of Pennsylvania Kutztown, PA 19530.

9. c'Object first with java and BlueJ (Seminarsession)'. *ACM SIGCSE Bulletin, 32(3), pp.429*

10. Kolling M. (1999a) 'The problem of teaching Object-Oriented Programming Part I: Languages'.*Journal of Object-Oriented Programming, 11(8), pp.8-15.*

11. Kolling, M., Rosenberg, J. (1996a) 'An Object-Oriented Program development environment for the first programming course'.*ACM SIGCSE Bulletin, 28(1), pp.83-87*

12. Lawal B. (2011). Study of Teaching and Learning Methods in Object Oriented Programming. An unpublished MSc thesis submitted to Coventry University United Kingdom

13. Lahtinen, E., Ala-Mutka, K. &Jarvienen, H., (2005) 'A study of the difficulties of novice programmers'. In Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, 14-18, June 27-29, Monte de Caparica, Portugal. Also Published in *ACM SIGCSE Bulletin, 37(3), pp.14-18*

14. Lister, R., Adams, E., S., Sue, F., William, F., John, H., Lindholm, M., McCartney, R., Mostr¨om, J., E., Sanders, E., Sepp¨al¨a, O., Simon, B., and Thomas, L. (2004). A multi-nationalstudy of reading and tracing skills in novice programmers.*SIGCSE Bulletin,* 36:119–150.

15. McCartney, R., Boustedt, J., Eckerdal, A., Mostr¨om, J. E., Sanders, K., Thomas, L., and Zander, C. (2009).Liminalspacesandlearningcomputing.

European Journal of Engineering Education, 34(4): 383–391.

16. McCracken, M. &Almustrum, V. (2001) 'A multi-national, multi-institutional study of assessment of programming skills of first-year CS students'. *In Working Group Reports from ITiCSE on Innovation and Technology in Computer Science education.December 01, 2001, Canterbury, UK.*

17. Moström J. E., (2011). A Study of student problemsin learningto program. A published PhD thesis submitted to the Department ofComputingScienceUme˚aUniversity, SE-901 87Ume˚a,Sweden. ISBN978-91-7459-293-1, ISSN0348-0542, UMINF11.10

18. Moström, J. E., Jonas, B., Eckerdal, A., McCartney, R., Sanders, K., Thomas, L., and Zander, C.(2008).Concrete examples ofabstractionasmanifestedinstudents'transfo rmative experiences. In ICER'08: Proceeding of the Fourth international Workshop on Computing Education. Research, pages 125–136, New York, NY, USA, 2008.ACM.

19. Neve, P., livingstone, D., Hunter, G., Alsop, G. (2013). Improving the student experience on computer programming courses with a holistic approach to learning design. Paper presented at the 2ndAnnual Conference on the Aiming forExcellence in STEM Learning and Teaching Higher Education Academy.

20. Okur, M. C. (2006) 'Teaching object oriented programming at the introductory level'. *Journal of Yasar university, 1(1), pp.149-157* [online] available from http://joy.yasar.edu.tr/makale/no2_nol1/05_m_cudi_okur.pdf [12 January 2014]

21. Robins, A., Ronntree, J., Ronntree, N. (2003) 'Learning and Teaching Programming: AReviewandDiscussion'. *Computer Science Education,13(2), pp.137-172*

22. Sanders, K., Boustedt, J., McCartney, R., Mostr¨om, J.E., Thomas, LA. and Zander, C. (2008). Student understandingof object-oriented programming as expressed in concept maps.In SIGCSE '08: Proceedings of the 39th SIGCSE technical symposium onComputer science education, pages 332–336, New York, NY, USA, 2008.ACM.

23. Soloway, E. &Spohrer, J. (1989) *Study of Novice Programmer.* Hillsdale, New Jersey: Lawrence Erlbaum Associates

24. Stroustrup, B. (1994) *The Design and evolution of C++.* New Jersey: Addison-Wesley.

It is not because things are difficult that we do not dare, it is because we do not dare that they are difficult.

~ Seneca