The Role of Planning in Object-Oriented Programming for Beginners

Christina Schweikert [1]
[1] Division of Computer Science, Mathematics and Science, St. John's University, Queens, NY USA

Abstract - *Programming languages, environments, and tools have evolved over time and various programming paradigms, including procedural, functional, object oriented, and scripting languages, have been developed. Despite the advancements, programming remains a difficult task for some novices. Learning an object-oriented language, such as C++ or Java, as a first language presents additional challenges for students and instructors. Incorporating the concept of planning into the learning of object-oriented programming may facilitate beginners' understanding of implementing the solution to a problem, as well as the design and implementation of objects. Focus is also placed on properly integrating objects into a problem solution.*

Keywords: object-oriented programming, knowledge representation, planning, computer science education

1 Introduction

Learning object-oriented programming continues to be a challenging task for many students in introductory programming courses. Object-oriented languages, such as C++ or Java, have additional layers of abstraction, due to the use of objects, that may be more easily grasped by some students than others. There has been ongoing discussions within the computer science community about how objects should be presented. [3,4,5] Many textbooks that attempted the "early objects" approach have released "late objects" versions due to demand. In order to understand how to implement classes, one must first understand data types, methods, parameters, return values, among other concepts. A professor could take a mixed approach in which programming is taught starting from the basics, but demonstrates built-in classes early on - for example, from the Java library. It is becoming more critical that, in addition to learning the programming language, students develop strong problem-solving skills. There have been several visual and interactive environments for exposing students to programming, including Scratch [7], Alice [2], and BlueJ [1], to name a few.

2 Plan Knowledge Representation

Utilizing the concept of plans in programming is rooted in the idea that "experts" in a field develop specialized knowledge through their experiences, which could include a set of actions needed to accomplish a goal. For example, a master in the game of chess has built up years of "experiential knowledge" that is drawn upon when a chess master faces a new, but similar, situation to one encountered in the past. It is then that the expert draws upon their experience, or set of plans, to make the moves necessary to achieve the goal – in this example, winning the game. We can represent this experiential knowledge as a plan. The concepts of scripts and plans for knowledge representation originated in the area of natural language processing (NLP). A script is a structure that contains a predetermined sequence of events that applies within a particular context. Plans account for general knowledge that can be used in new situations. Plans contain a set of choices needed to accomplish a goal. For example, when discovering a plan while reading a book, one can make guesses about the intentions of an action in an unfolding story and use these guesses to make sense of the story. [10] Experienced programmers can remember programs better than beginners if the programs have some meaningful structure, or plan. [11] Experts can recognize plans they have become familiar with through experience, such as a program that searches for a value in a list. Concepts from the research area of planning in the field artificial intelligence can be applied when dealing with applications on a larger scope and scale [6, 8]; however, here the focus is strictly on novice programming.

3 Class Design using Plans

When learning to program, beginners often have difficulty in properly designing and implementing classes, and using objects. This includes errors in determining the appropriate attributes and methods for a class, as well as failure to properly utilize objects in a problem solution. To get learners accustomed to class design, we can utilize a "plan" to design a class, along with component plans for creating attributes and methods, as well as inheritance relationships. Plans can also be used to integrate resulting objects into a program. The idea is to help beginners design classes and incorporate objects as part of a larger solution to a problem. Plans, and their components, are then implemented in the chosen object-oriented programming language. As an example, a simple Inventory Plan will necessitate an Item Object. Based on the specification of the Inventory Plan, the proper attributes and methods will be created and integrated into the Item Object. Here, the Item Object would require itemID, itemName, and price attributes, as well as a displayItem() method.

4 Learning with Plans and Objects

A prototype system, Web Plan Object Language (WPOL), which uses the concept of programming plans

within an object-oriented paradigm has been designed. [9] Representing programming knowledge, even simple tasks such as computing an average, as plans can help develop students understand how the solution to a problem is translated into code. Most problems are complex and involve multiple plans that need to be integrated together. Plan integration refers to the relationship between plans, such as plans that are sequential (appended), branched, embedded, or interleaved. The system consists of 3 phases of learning: Plan Observation, Integration, and Creation. The phases are described in the following sections.

4.1 Plan Observation

In this phase, solutions to sample programs are visually demonstrated, step by step, using plans to design necessary objects and to design other program tasks. Plans are integrated to form the final problem solution, and then the plans are transitioned into code. The Observation Phase begins with the problem description and identification of the major components. The Student Average Plan is used as an introductory example.

Student Average Plan Description: Compute the average of a student's assignment, midterm, and final scores. A student's id, assignment, midterm, and final scores will be input. The student's id, along with the computed average, will be displayed.

A portion of the Observation Phase of a sample Student Average Plan is illustrated in Fig. 1; this screen shows the implementation of the computeAverage() method of the Student class. The Student Object Plan contains embedded plans for attributes (Data Member Plan), and methods (Member Function Plan). In this case, the language of choice is C++. If using Java, the corresponding terminology would be used. Fig. 2 shows the integration of the Student Object Plan into the Student Average Main Plan and a sample running of the program. "Object Utilities" is included here and consists of a Set Plan and Get Plan, which create set and get methods for the attributes. Constructors (and destructor if needed) would also belong here. (Note: Only a portion of the entire solution process is shown due to space constraints.)

The second example provided is a Sort Students Plan; this program builds on the previous Student Average Plan and extends it by incorporating concepts of inheritance, arrays, loops, decision-making, and sorting. The Student class inherits a newly created Person class, an array of Students is created, Students are sorted by their computed average using Bubble Sort, and the sorted array of objects is displayed. Snapshots from this example are included in Fig. 3, 4, and 5.

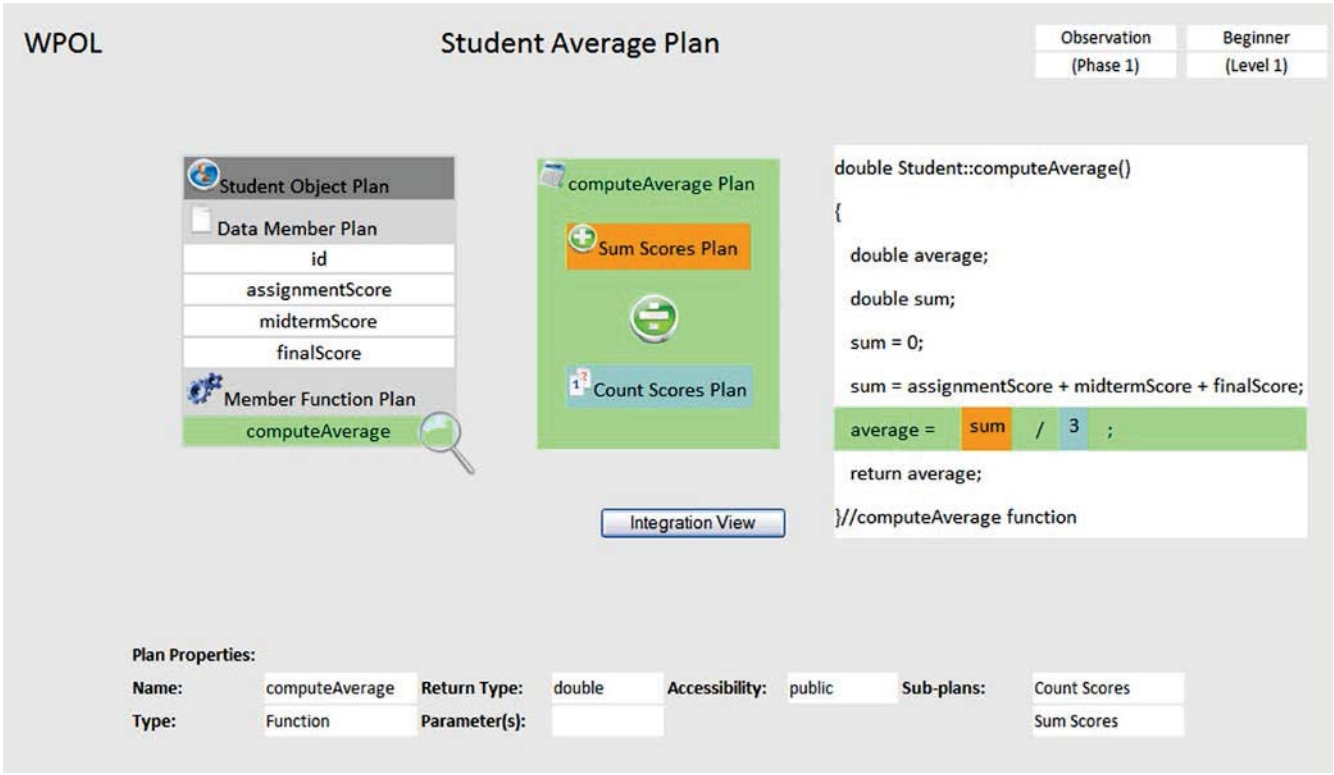

Fig. 1 – Code View for computeAverage Plan

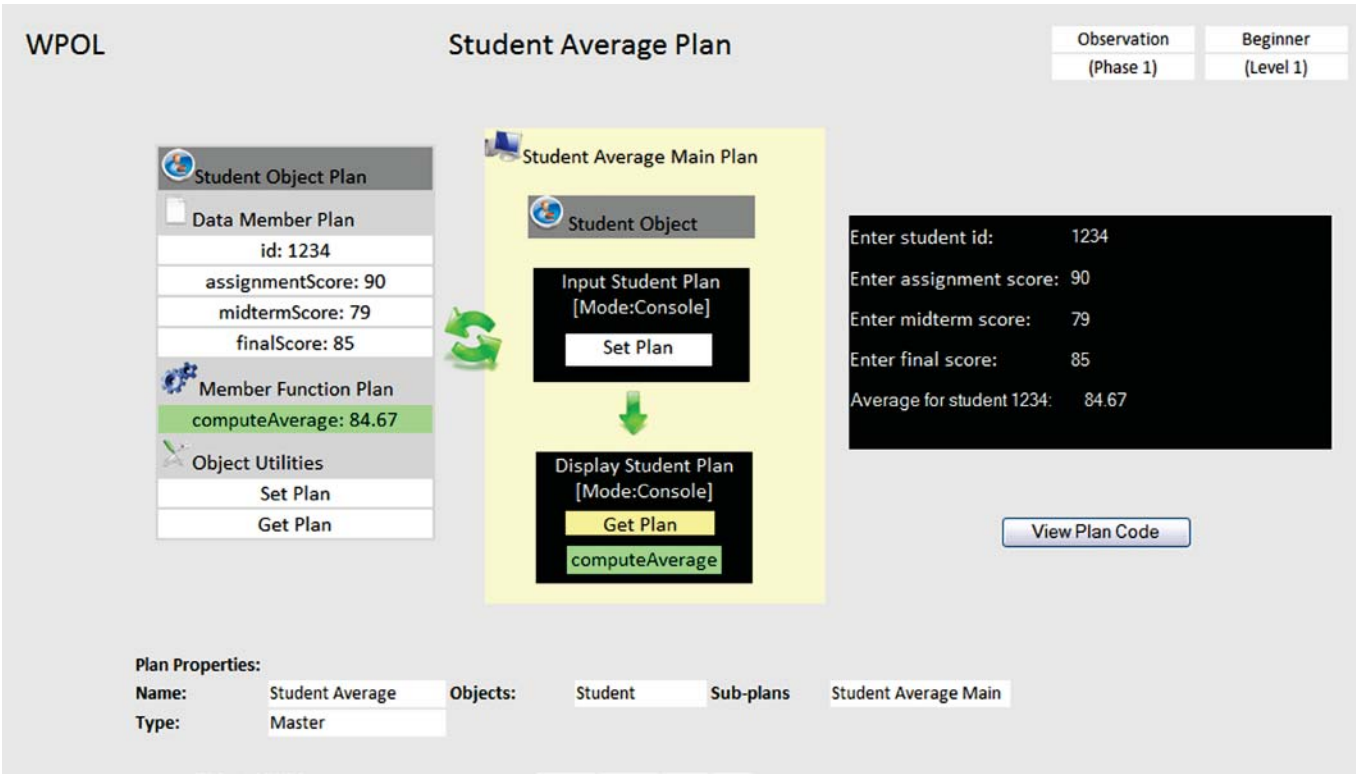

Fig. 2 - Integration View for Student Average Main Plan

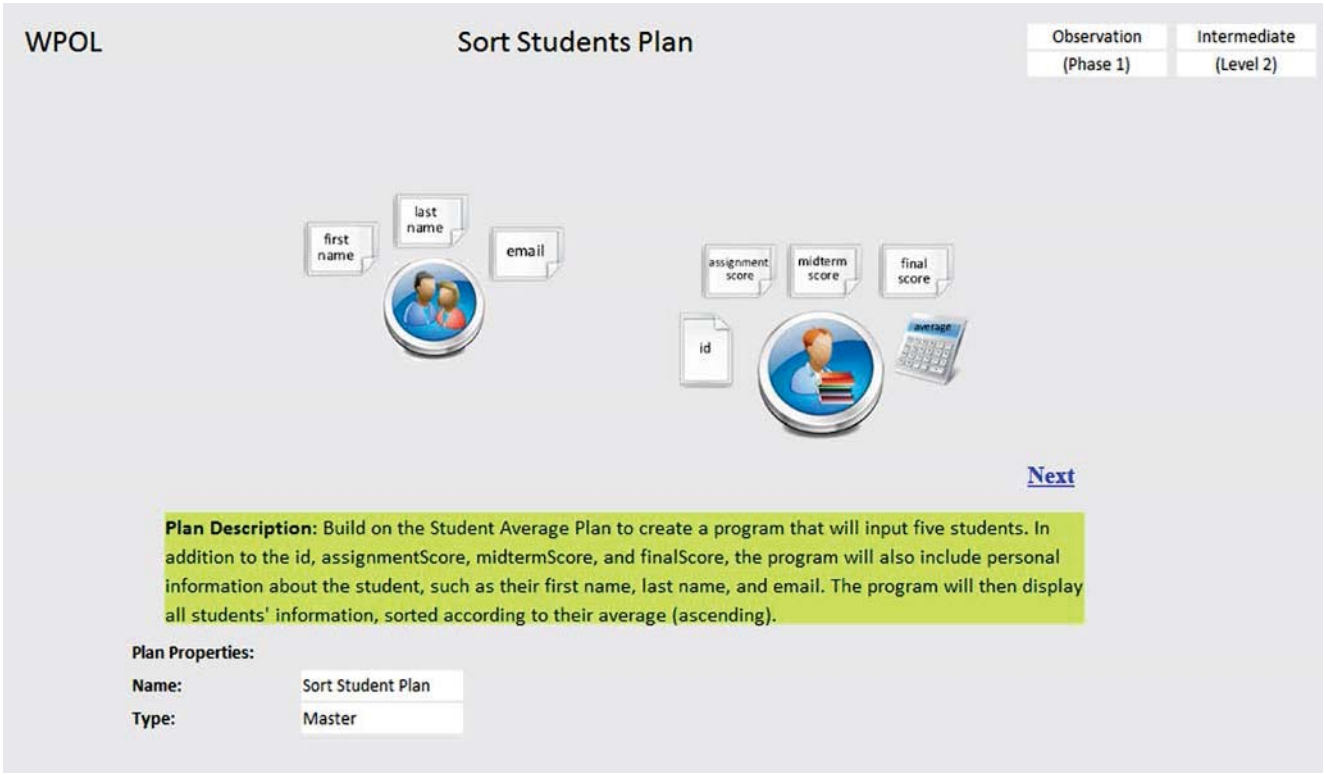

Fig. 3 - Sort Students Plan Description

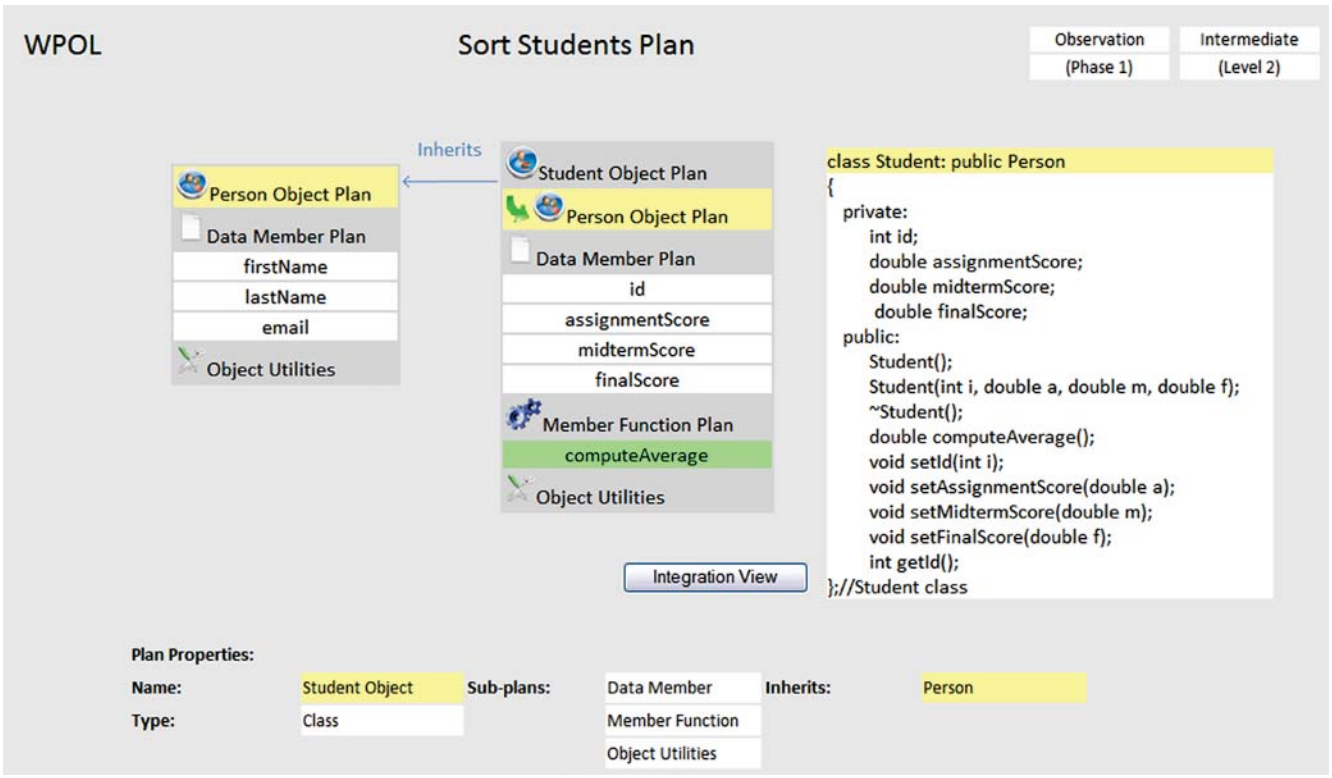

Fig. 4 - Implementation of Inheritance (IS-A) Relationship

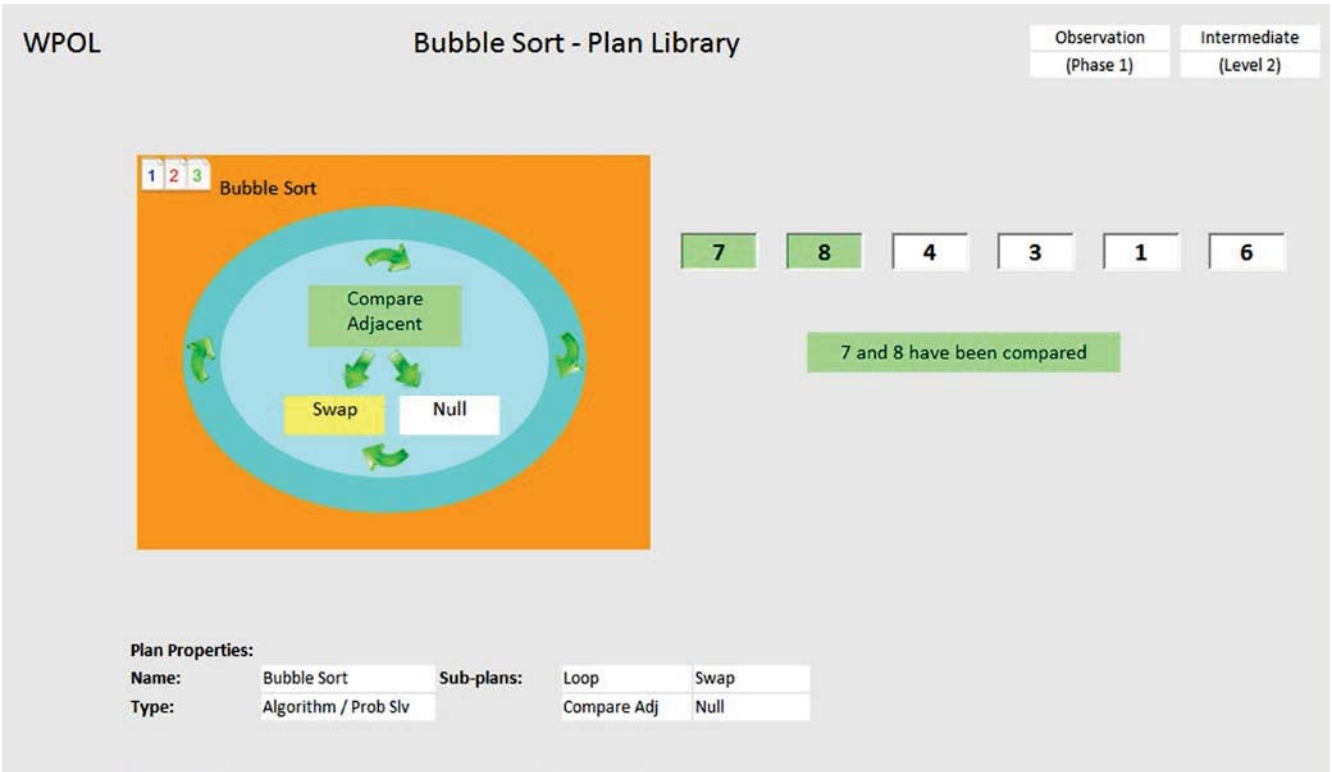

Fig. 5 - Compare Adjacent Plan Demonstration

4.2 Plan Integration

In the Integration Phase, a student's ability to properly integrate plans to form a solution is tested. The purpose of this phase is to reinforce concepts of plan integration and object design. The student is provided with a description of a program to be completed. Then, the student is presented with plans and asked to select which plan(s) should be integrated. The correct integration mode (Appended, Branched, Embedded, or Interleaved) must also be selected. This reinforces the students' understanding since they are taking an active role in creating the solution, and incorrect solutions are explained. A sample incorrect plan integration is shown in Fig. 6.

4.3 Plan Creation

In the Creation Phase, students can customize plans and design new objects (classes). Plans are customized by setting plan properties. This phase facilitates students in creating a program template. For example, an Object can be created by setting the properties of an Object Plan, as well as the properties of its sub-plans. This includes setting class attributes and methods. A screenshot from the creation of an Object Plan for a Book class is demonstrated in Fig 7.

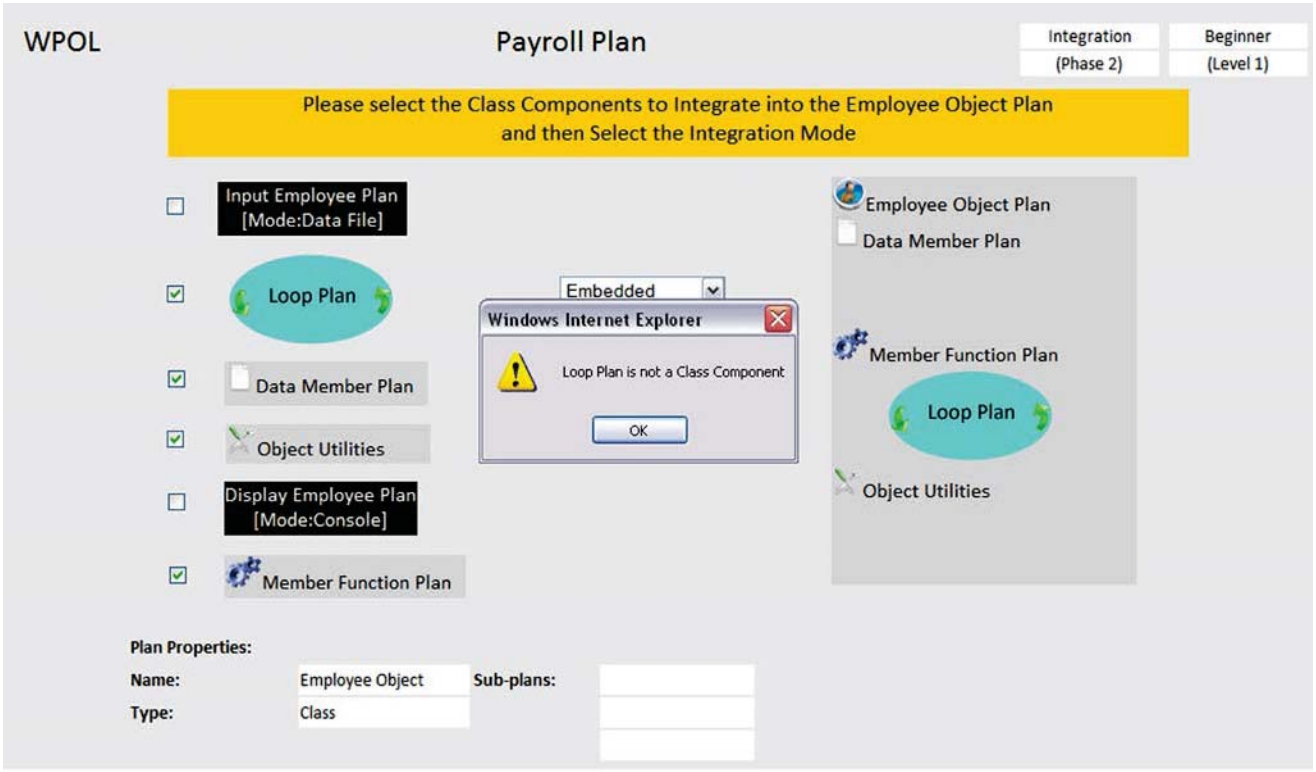

Fig. 6 – Selection of Incorrect Plan

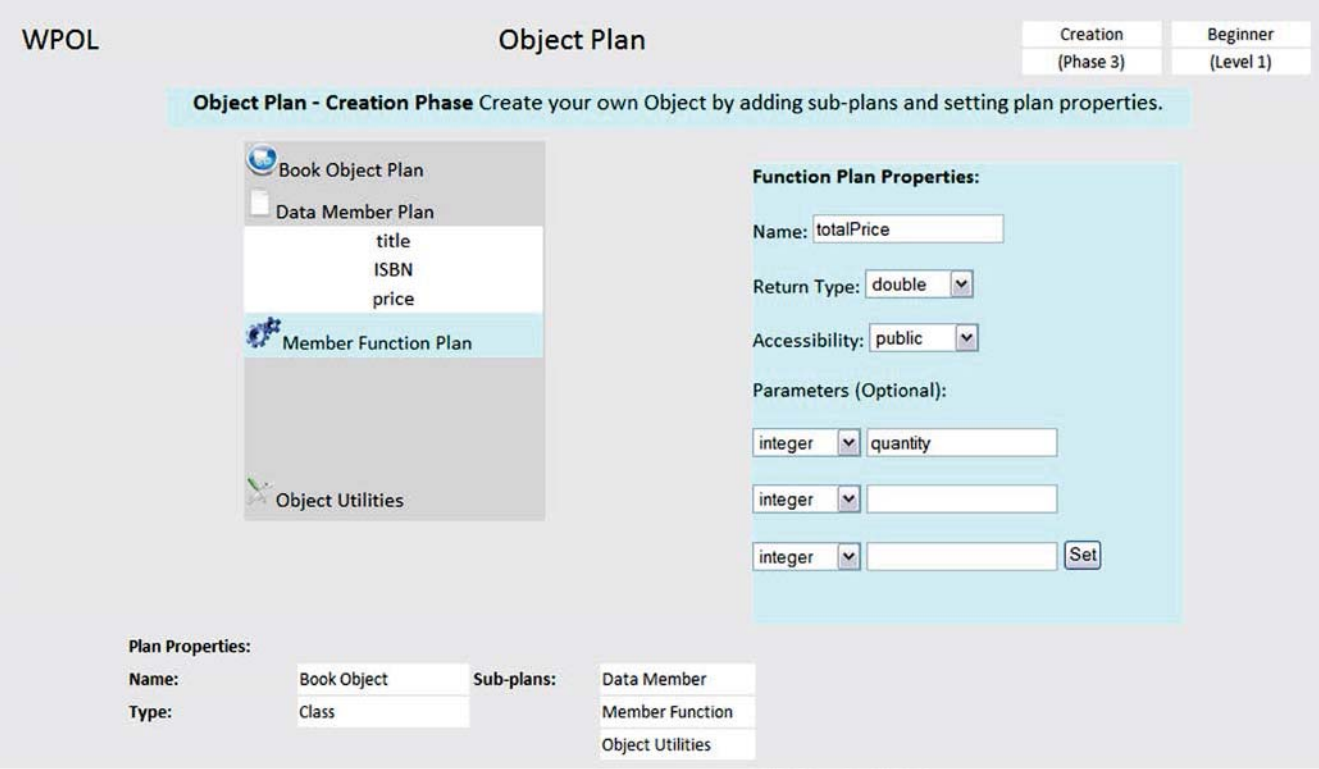

Fig. 7 – Function Plan Properties

5 Conclusions and implications

As computer science educators, we are constantly seeking ways to enhance students' experience of learning programming, and to enable better assimilation of programming concepts, such as object oriented programming. This project seeks to capture the way expert programmers represent programming knowledge and visualize this knowledge representation for novices to enhance their learning of programming in the object-oriented paradigm. Objects can be introduced early with a visual environment and plan representation that reinforces object design and object oriented concepts. A contribution of this research project is using the concept of plans to teach object-oriented programming and problem solving. This approach enhances novice programmers' ability to design, implement, and integrate objects into their programs. Another contribution of this work is a proposed learning environment that utilizes the planning approach with three phases of learning: plan observation, plan integration, and plan creation. The proposed environment is easily adapted to any object-oriented language, such as Java and C#.

6 References

[1] D. J. Barnes & Michael Kölling, Objects First with Java: A Practical Introduction using BlueJ, Fifth edition, Prentice Hall / Pearson Education, 2012.

[2] Conway, M. et al. Alice: Lessons Learning from Building a 3D System for Novices. in Proceedings of CHI (2000). ACM Press, 486-493.

[3] C. Hu, Rethinking of Teaching Objects-First. *Education and Information Technologies* 9, 3 (September 2004), 209-218, 2004.

[4] C. Hu, Just say 'A Class Defines a Data Type'. *Commun. ACM* 51, 3 (March 2008), 19-21, 2008.

[5] A. Ehlert and C. Schulte, Empirical comparison of objects-first and objects-later. In *Proceedings of the fifth international workshop on Computing education research workshop* (ICER '09). ACM, New York, NY, USA, 15-26, 2009.

[6] S. Lucci and D. Kopec, Artificial Intelligence in the 21st Century, 2nd ed. Dulles, Virginia: Mercury Learning & Information, 2015.

[7] J. Maloney, M. Resnick, N. Rusk, B. Silverman, & E. Eastmond. The Scratch Programming Language and Environment. Trans. Comput. Educ. 10, 4, Article 16 (November 2010), 15 pages.

[8] S. J. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, 3rd ed. Upper Saddle River, New Jersey: Prentice Hall, 2009.

[9] C. Schweikert, Study of novice programming: Plans, object design, and the Web Plan Object Language (WPOL). Ph.D. dissertation, The Graduate Center, City University of New York, 2008.

[10] R. Schank and R. Abelson, Scripts, Plans, Goals and Understanding: An Inquiry into Human Knowledge Structures. Hillsdale, NJ: Erlbaum, 1977.

[11] E. Soloway, K. Ehrlich, J. Bonar, Tapping into Tacit Programming Knowledge. Proceedings of the Conference on Human Factors in Computing Systems, NBS, Gaithersburg, Md, 1982.