Using Coding Patterns in a Model-Driven Approach to Teaching Object Oriented Programming

James H. Paterson, School of Engineering and Computing, Glasgow Caledonian University, james.paterson@gcal.ac.uk
John Haddow, University of Strathclyde, john.haddow@strath.ac.uk
Ka Fai Cheng, School of Engineering and Computing, Glasgow Caledonian University, k.cheng@gcal.ac.uk

Introduction

Learning object oriented programming presents a range of concepts which many students find difficult to grasp. **Objects-first** teaching approaches introduce at an early stage the key concepts of object oriented programming and design. However, "objects-first" is not a well-defined term, and many different interpretations of the approach have been described. One interesting approach is that of Bennedsen and Caspersen[1], who describe a model driven approach to teaching introductory programming. This approach explicitly includes a conceptual modeling perspective in which **coding patterns** are introduced for the implementation of classes and of associations between classes.

Following a similar approach we have introduced a set of activities within object-oriented programming classes which focus explicitly on the transition from conceptual model classes to code.

Evaluation

An initial evaluation was done to look for evidence of the impact of the model-based activities on the students' ability to develop their own conceptual models. Project work done following on from the activities was reviewed to identify the level of incidence of the set of common design faults reported by Thomasson et al.[3]. This is a very small scale review of the work of 20 students who were organized into groups of 3 or 4.

The fault types are described as *"non-referenced classes"* **(NRC)**, *"references to non-existent classes"* **(NEC)**, *"single attribute misrepresentation"* **(SAM)** and *"multiple attribute misrepresentation"* **(MAM)**. Details of these fault types can be found in the reference.

The results are shown in the table, together with the equivalent results obtained by Thomasson et al.. The striking feature apparent in the table is that *no* non-referenced class faults were observed in the work of our students, in contrast to the high incidence of this fault in the previous study. This suggests that these students at least have a clear understanding that a class must be associated with other classes in order to play a part in a system.

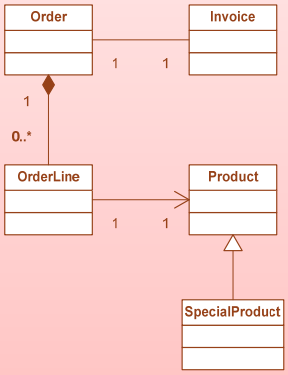
Fault	This work	Ref. 3
Total number of NRC	0	629
% of designs with this fault	*0%*	*89%*
Total number of NEC	4	116
% of designs with this fault	*20%*	*31%*
Total number of SAM	31	277
% of designs with this fault	*60%*	*50%*
Total number of MAM	4	42
% of designs with this fault	*20%*	*15%*

References

1. Bennedsen, J. and Caspersen, M. 2008. Model-Driven Programming. In Reflections on the Teaching of Programming, Lecture Notes in Computer Science Vol. 4821, 116-129, Springer-Verlag Berlin / Heidelberg,
2. Paterson, J.H. and Haddow, J. 2007. Tool support for implementation of object-oriented class relationships and patterns. ITALICS, Special Issue on Innovative Methods of Teaching Programming, Vol 6, No 4, 108.
3. Thomasson, B., Ratcliffe, M., and Thomas, L. 2006. Identifying novice difficulties in object oriented design. In Proceedings of the 11th Annual SIGCSE Conference on innovation and Technology in Computer Science Education (Bologna, Italy, June 26 - 28, 2006). ITICSE '06. ACM, New York, NY, 28-32.

Model-driven programming activity

All tasks are based on this model and each task involves reflection on multiplicities, navigability and the association semantics.



Task 1 **Implement the association between *OrderLine* and *Product*** using a specified coding pattern with the aid of PatternCoder.

Task 2 **Implement the association between *OrderLine* and *Order*** using a suggested coding pattern with the aid of PatternCoder.

Task 3 **Implement the association between *Order* and *Invoice*** considering the semantics of this association carefully and using a suggested coding pattern with the aid of PatternCoder.

Task 4 **Implement the association between between *Product* and *Special Product*** with the aid of PatternCoder in selecting a coding pattern and creating code.

Task 5 **Add a *Customer* class which has associations with both *Order* and *Invoice*** and implement its association with Order with the aid of PatternCoder.

Task 6 **Modify the *Customer* class to implement the association with *Invoice*** using a coding pattern which has been seen previously but without the aid of PatternCoder.

The PatternCoder tool

PatternCoder is based on the idea of coding patterns, and provides support for learning and exploring these patterns. It is available as an extension for the BlueJ Java IDE. Further details are in reference 2.