In P. Romero, J. Good, E. Acosta Chaparro & S. Bryant (Eds). Proc. PPIG 18 Pages 215 - 229

A Competence Model for Object-Interaction in Introductory Programming

Jens Bennedsen and Carsten Schulte

IT University West,
Fuglsangs Allé 20, DK-8210 Aarhus V, Denmark
jbb@it-vest.dk
Department of Computer Science, Free University Berlin
Takustrasse 9, D-14195 Berlin, Germany
schulte@inf.fu-berlin.de

Abstract: Assuming that understanding object-oriented programming requires the understanding of object-interaction, this article outlines the development of a theoretical model that provides a framework to assess a hierarchy of competences related to object-interaction. A newly developed test allows for an in-depth analysis of this hierarchy, including its relationship with other (e.g. more 'traditional') factors that impact students' understanding of object-oriented programming. Based on a study at two learning institutions, we conclude that the proposed model is an effective tool for describing different competence levels. The analysis of how different factors influence students' object-interaction skills shows a correlation between object-interaction and imperative programming, as well as self-efficacy; the correlation between object-interaction and math, however, was weak. We found that the degree of visibility of object-interaction in the program text is the most critical factor for understanding object-interaction. The analysis is followed by a discussion of the implications of the findings for teaching.

Introduction

The problems novices encounter when learning how to program have been studied for many years. Most of the studies have revealed that students in general find it challenging and difficult to learn how to program, regardless of the taught paradigm..

Research results concerning difficulties in the process of learning programming – both in multi- [1-3] and one-institutional [4-6] studies – have shown a low performance of novices. .

Many studies find similar, general learning problems (see e.g. [7] (p.55) [1] (p.15) [8] (p.214)), and point to the lack of general understanding of the program execution.

According to du Boulay ([9] p. 283ff), this problem addresses one of the five areas of learning programming, namely the understanding of the notional machine, which is an abstract model of the machine executing programs (i.e. the meaning of a running program). Understanding an object-oriented notional machine requires comprehending the interaction of objects during runtime. Therefore, students need to

understand that (1) an object-oriented program consists of several different objects, (2) their connection within certain structures, and (3) the dynamics of these object structures [10]. In this paper, we address this specific learning problem, which is also highlighted by Guzdial [11]: "A specific problem that students encounter is creating collaborative objects – students have difficulty creating and understanding connections between objects" (p. 182).

The above-mentioned studies establish the fact that students find the notional machine problematic, but neither do they answer the question which particular aspects are the difficult ones, nor do they explain how students develop a way of understanding a notional machine for object-oriented programs. In this study, we therefore try to reveal the current state of students' understanding of object-interaction and the variance in this understanding in order to find correlations with other factors whose impact on learning object-oriented programming has already been established. This study aims at providing new empirical insights by analyzing in detail how students' understanding of object-interaction progresses.

Consequently, we develop a theoretical model that illustrates the increasing understanding of object-interaction, starting from a basic level progressing to advanced, more complex modes of understanding. The model describes the gradual progression from a novice's to an expert's perspective. It is a competence model, describing learning progress, allowing to evaluate and diagnose a students' current understanding, and to plan further suitable teaching steps. In short, it is useful to build such a model as a sequence of several progressing steps. As a famous example of such a model one can think of Bloom's taxonomy of the cognitive domain [12]. According to Bloom, understanding at a certain level requires understanding of all lower levels. It remains to be seen whether a competence model of object-interaction can have this taxonomic feature. Of course one can distinguish simple from more complex interaction patterns, but increasing complexity is interwoven with increasing diversity: for example number of objects, number of interactions, and the use of advanced features like inheritance and polymorphism. The taxonomy can thus be a partial or total ordering of the levels.

This diversity is one major factor why learning object-oriented programming is such a complex problem for computer science education. A model describing the development of expertise – even in a sub-field like interaction of objects –is useful; but it seems unlikely that there is one ideal model. Unavoidably such a model, even if it can be in part empirically validated, is based on a certain perspective – and other successful models built on other perspectives may exist. A competence model is in part a normative approach.

The aim of this article is therefore focused on three core issues:

i) develop a theoretical model to describe a hierarchy of competences related to object interaction;

ii) create and validate a test instrument for such a hierarchy;

iii) examine relationships between the hierarchy and other (e.g. more 'traditional') factors influencing students' understanding of object oriented programming.

A Competence Model for Understanding Object-Interaction

As mentioned above, the understanding of object interaction comprises different aspects. Therefore, a hierarchy must be set up from different aspects: the number of objects involved, the complexity of the method-call sequence, and the complexity of structural changes (creating or deleting objects and references). Inheritance and dynamic binding play an important role, which can be considered as an additional complexity. Moreover, some of the information may be directly visible (e.g. new marks the creation of an object), while other information is only implicitly given (e.g. deciding whether two variables refer to the same object). Because of these different factors, there may be many different hierarchies.

Our aim is to describe a taxonomy that reflects the development of expertise. Based on the literature (see [10] for references) and our experience in teaching object-orientation, we have developed the following four-levelled hierarchy. The hierarchy is intended to be a taxonomy, where persons understanding level *n* also understand level *n-1*. In the following, the word taxonomy includes this hierarchical feature.

1. *Interaction with objects*
   The student can understand simple forms of interactions between a couple of objects, such as method calls and creation of objects. The student is aware that the results of method calls depend on the identity and state of the object(s) involved.
2. *Interaction on object structures*
   The student is able to comprehend interaction on more than a couple of objects, including iteration through object structures and nested method calls. The structure is created and changed explicitly via creations, additions and deletions.
3. *Interaction on dynamic object structure*s
   The student knows the dynamic nature of object structures, understands the overall state of the structure and is aware that the interaction on the structure or elements of it can lead to side-effects (e.g. implicit changes in the structure).
4. *Interaction on dynamic polymorphic object structures*
   The student takes into account polymorphism in dynamic object structures and is able to understand the effects of inheritance and late binding on dynamic changes in the object structure. Side-effects of late binding (different method-implementations, different actual objects referred to by the same variable) are also considered.

Of course, this model presents only one possible sequence of increasing complexity. For example, one could include polymorphic interaction patterns before dynamic interaction, or conceptualize the introduction of inheritance as an additional step in the hierarchy. Instead of theoretically discussing these aspects or details of the hierarchy, we decided to test the proposed model empirically and hope that valuable insights about the further development of the levels of the model may thereby be gained.

Create and Validate a Test Instrument

We do not know whether the proposed model describes the development of students' understanding. Our aims therefore are i) to evaluate the competence model and ii) to evaluate its usefulness (i.e. relationship to other learning issues). For the latter we will explore relationships between the hierarchy and other (e.g. more 'traditional') factors influencing students' understanding of object-oriented programming. Some questions regarding these relationships in the test instrument are thus included.

The Test Instrument

We constructed a questionnaire with easy-to-evaluate questions. Essay questions and tasks to draw a kind of object diagram were deliberately excluded, since these questions require interpretations by the researchers and knowledge not necessarily known to all students.

Questions for each level of the proposed taxonomy were created, focusing on object-interaction. We furthermore used questions to examine relationships of understanding of object interaction with traditional factors influencing program understanding. These questions include imperative aspects, self-efficacy, math grade and programming experience.

Category name	Description of questions included in the category	Value
Total	Number-of-objects + Object interaction + Imperative aspects	Percentage of correct answers
Number-of-objects	7 questions on the number of objects created during execution. We used the number of objects as an indicator for understanding the object structure as a prerequisite for the object interaction	Percentage of correct answers
Object interaction	32 questions on the object-oriented part of the notional machine; focused on the result (output) of interacting objects	Percentage of correct answers
Imperative aspects	4 questions from [2] (questions two, four, five, six)	Percentage of correct answers
Self-efficacy	3 Questions from Ramalingam and Wiedenbeck [17]	From 0 (not at all) to 4 (very much)
Math grade	The high-school math grade; reported by the students	A-F grade
Programming experience	The student's number of programming courses, number of object-oriented programming courses and the number of programming languages he knew	Three numbers in the rage 0 - 10

Table 1: The different parts of the test instruments

In summary, there are 32 object interaction questions ("What is the output of this program at this point?") covering all four levels of the hierarchy, seven number-of-objects questions, four imperative aspects questions and several questions asking for background information (see Table 1). The questions in the categories in Total are referred to by a letter ('a'-'o') and their occurrence in the program – e.g. the example in Figure 1 is referred to as h1, h2 and h3 for the first, second and last question. The

questionnaire can be found at http://www.daimi.au.dk/~jbb/questionnaire.pdf (including the assignment of each question to a level).

Traditional Factors Included in the Test Instrument

Many factors have been tested as a predictor for success in introductory programming (e.g. [4, 6, 13-16]). In this study, we include imperative programming, object-oriented self-efficacy, math grade and programming experience, as they appear to have a positive impact, as reported by most authors.

Students' competences in the imperative aspects of programming have been assed by [2]. In order to evaluate and compare the respondents' imperative aspects, four of the questions from [2] (questions two, four, five and six) were included (Imperative aspects).

Ramalingam and Wiedenbeck [17] developed a self-efficacy scale for computer programming. We used the three items (#13, 16 and 14) from their scale that have to do with understanding object-orientation as a scale for self-efficacy (Self-efficacy).

Several studies have shown that a student's math grade is a predictor of success for a programming course [4, 5, 14]. The high-school math grade was reported by the students and converted to a common A-F scale (Math grade).

In this study programming experience was measured by a student's number of programming courses, number of object-oriented programming courses and the number of programming languages known (Programming experience).

Object-Interaction Questions Included in the Test Instrument

The questions on the object-oriented part of the notional machine focused on the result (output) of interacting objects and the number of objects created during the execution. We used the number of objects as an indicator for understanding the object structure as a prerequisite for the object interaction. We choose this type of question to avoid drawing object diagrams, since this would imply additional prerequisites on the students and an interpretation of the drawings.

There are 32 Object interaction questions. All of the questions faced the students with a fragment of code and asked for the result of the execution of code; an example can be seen in Fig. 1. All places where the students were supposed to give an answer were marked with grey boxes.

The first question in Fig. 1 tests level two in the hierarchy, since objects are created in an object structure, but it is obvious from the program text what the structure should be. The second question tests understanding of object-interaction at level three, because

```
l.set(0,new A(5));
```

changes the reference of the first element to a new object with another value than the original one.

In order to test the impact of polymorphism, several questions with simple object interaction but using polymorphism were included. It had no impact (see section 0).

```
public class A {
    private int a;

    public A(int newA) { a = newA; }
    public int getA() { return a; }
}

import java.util.*;
public class TestClass {
    static public void main() {
        List<A> l= new ArrayList<A>();
        l.add(new A(76));
        l.add(new A(-10));
        l.add(new A(6));
        l.add(new A(43));
        l.add(new A(12));
        int sum = 0;
        for(A a:l) {
            sum+=a.getA();
     }
        System.out.println(sum);
     // what is the output here? <<FIRST QUESTION>>


        l.add(new A(701));
        l.set(0,new A(5));
        sum = 0;
        for(A a:l) {
            sum+=a.getA();
     }
        System.out.println(sum);
     // what is the output here? <<SECOND QUESTION>>

   }
}
     How many objects are created? <<THIRD QUESTION>>
```

Fig. 1. A typical question. The text between << and >> was not a part of the original text.

The Empirical Study

The Study

In the following we describe how the test instrument was validated, who participated in the study, the reliability of the test instrument and the time used when using the test instrument.

We conducted a pilot study among five students. The students not only had to answer the questionnaire, but were later interviewed about their perception of the questions. Based on the pilot study, we designed the final instrument.

125 students answered the final questionnaire: 50 students from the Free University of Berlin and 75 from the University of Aarhus. The students from Free University Berlin participated in a course called "Algorithm and Programming 3," whereas the students from the University of Aarhus participated in "Programming 2." Algorithm and Programming 3 is the third semester-long course on programming that the students from Berlin had taken. During the first semester, they learned a functional approach, and in the second semester, they studied basic object-oriented concepts. Programming 2 is a second-quarter course; the students from Aarhus' previous course were studying basic object-oriented programming.

The students were given the test with no time limit. The average time needed was 57 minutes; the quickest completed it in 28 minutes and the slowest took 1:25. The Pearson correlation coefficient between the time used to answer the questionnaire and the total result (interaction-, number-of-object- and Lister questions) showed no correlation. We computed Cronbach's-alpha to check the reliability of the final questionnaire: 0.908. A value of at least 0.8 is normally considered to indicate a reliable instrument; we conclude that the instrument is reliable.

Object Interaction Competence Levels

The overall 32 questions regarding object-interaction differ in complexity. The easiest is correctly answered by almost all students; the hardest is answered correctly by only 16% (see Fig. 2)

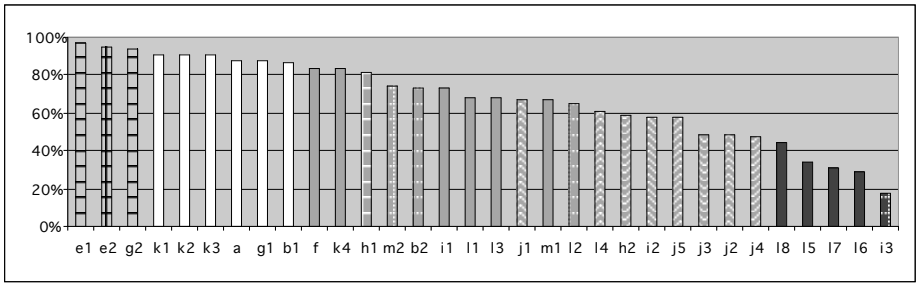

Fig. 2: Percentages of correct answers for all object interaction questions (chequered questions excluded from further analysis, see text. white=level 1, gray=level 2, gray with waves=level 3, black=level 4)

The students answered an average of 72% of all object-interaction questions (Total) correctly (SD=0.21).

Before computing the resulting levels, we checked whether the questions distinguish high and low achievers. Therefore, we computed the correlation between each of the 32 object interaction questions and Total. Because of low selectivity, we excluded six questions from further analysis (the variables: a, e1, e2, h1, g1, g2, see

Fig. 2). Of the remaining questions, 93.6 % answered the easiest question correctly, but only 16% accurately answered the hardest.

Each question was assigned to the interaction level, given by the competence model discussed above (see Table 2). Questions m2, b2 and l2 were deleted in the following analysis, since their purpose was to test the impact of polymorphism when the object interaction was simple (see Section 5.2).

Question i3 (see excerpt in Figure 3) was the hardest question in the test, but did not include polymorphism and was therefore assigned to level 3.

```
// added objects to a list. Overall sum: 79
A a2 = l.get(4); // retrieves an object from a list l
a2.test(5); // changes the objects value from 10 to 5
sum=0;
for(A aIt3:l) {
      sum+=aIt3.getA(); //computes overall sum of values
}
System.out.println(sum); // what is the output here?
// most students (19%) answered same as before  or
decreased by 5 (19%). Correct answer: decreased by 25
(answered by 17%), because a reference to the same object
was included five times in the list on beforehand was
```

Fig. 3: Excerpt from question i3

The question is especially difficult because in the loop (Figure 3) the same object was added five times to the list (instead of five different objects). We thought this indirect, invisible change is indeed hard to detect but maybe not typical for object interaction on level 3. We excluded it as a malformed or misleading question. Table 2 shows which questions were used to compute the four levels.

Level 1	b1, k1, k2, k3
Level 2	f, i1, k4, l1, l3, m1
Level 3	h2, i2, j1, j2, j3, j4, j5, l4
Level 4	l5, l6, l7, l8

Table 2: Assignment of questions to levels of the interaction hierarchy

While 76% of the students answered all questions on level one correctly and no student was wrong on all questions, only 19% of the students answered correctly all questions on level four, and 41% of the students were wrong on all questions on the fourth level. Based on our questions and our assignment of questions to the proposed levels of the model, the empirical results confirm that the levels indeed show increasing complexity and difficulty of understanding object interaction.

In order to analyse whether this increasing difficulty can be seen as taxonomic, we assigned each student to one of the four levels, using two different methods. In both methods we claimed that a student needs to have answered 70% of the level *X* questions correctly for counting level *X* as understood.

The first method computed each level on its own, independent of the results of the other levels: if a student correctly answered 70% of the questions of that level, he