Student Understanding of Object-Oriented Programming as Expressed in Concept Maps

Kate Sanders
Department of Math and Computer Science
Rhode Island College
Providence, RI USA
ksanders@ric.edu

Jonas Boustedt
Dept. of Mathematics, Natural and Computer Science
Högskolan i Gävle
S80176 Gävle, Sweden
jbt@hig.se

Anna Eckerdal
Department of Information Technology
Uppsala University
Uppsala, Sweden
Anna.Eckerdal@it.uu.se

Robert McCartney
Dept. of Computer Science and Engineering
University of Connecticut
Storrs, CT USA
robert@cse.uconn.edu

Jan Erik Moström
Department of Computing Science
Umeå University
901 87 Umeå, Sweden
jem@cs.umu.se

Lynda Thomas
Department of Computer Science
Aberystwyth University
Aberystwyth, Wales
ltt@aber.ac.uk

Carol Zander
Computing & Software Systems
University of Washington, Bothell
Bothell, WA USA
zander@u.washington.edu

ABSTRACT

In this paper, we present the results of an experiment in which we sought to elicit students' understanding of object-oriented (OO) concepts using concept maps. Our analysis confirmed earlier research indicating that students do not have a firm grasp on the distinction between "class" and "instance." Unlike earlier research, we found that our students generally connect classes with both data and behavior. Students rarely included any mention of the hardware/software context of programs, their users, or their real-world domains. Students do mention inheritance, but not encapsulation or abstraction. And the picture they draw of OO is a static one: we found nothing that could be construed as referring to interaction among objects in a program. We then discuss the implications for teaching introductory OO programming.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computers and Information Science Education—*Computer Science Education*

General Terms

Human Factors, Experimentation

Keywords

CS1, object-oriented, empirical research, concept maps

1. INTRODUCTION

In this paper, we report on an investigation of student understanding of object-oriented (OO) concepts. Our questions were: (1) What do students who have recently learned about object-oriented programming see as the most important OO concepts? and (2) How do they express the relationships among those concepts?

We asked students at six institutions in three countries to draw concept maps summarizing their knowledge of OO programming. Concept maps are directed graphs in which each node is labeled with the name of a concept, and each edge is labeled with a description of the relationship between its endpoints. While the most frequently-mentioned concepts were not surprising, other concepts that we hoped to see appeared only rarely. In addition, while some maps indicated a good understanding of concept relationships, many were vague, and some suggested misconceptions.

In Section 2, we present related work on OO concepts, student misconceptions, and concept maps. We give our methodology in Section 3, our analysis and results in Section 4, some implications of our study for the teaching and learning of OO concepts in Section 5, and conclusions and future work in the final section.

2. RELATED WORK

This study builds on several previous projects. We took as a starting point for our analysis Armstrong's list of **OO concepts**: object, class, method, message passing, inheritance, polymorphism, encapsulation, abstraction, and in-

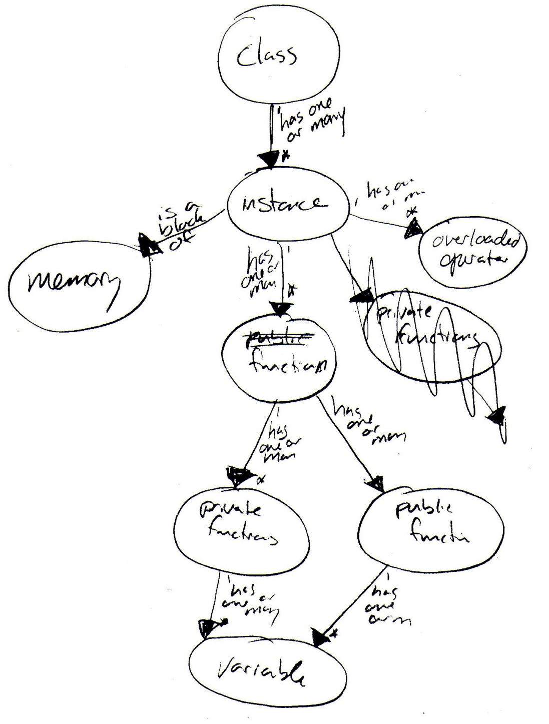

Figure 1: Sample concept map

stantiation (or instance). [1] We added "modeling," since it is another key idea that not all novices grasp. [4]

In addition to looking for these key concepts, we also looked for evidence of **misconceptions** reported in the literature: confusion between the ideas of instance and of class [6, 7]; confusion about the difference between a class and a collection of instances [19]; and the belief that objects are just records for storing and retrieving data and methods simply perform assignment. [7] A recent study examining CS1 programs by two of the authors found most of these misconceptions, with the exception that students showed little tendency to see classes as mere data storage. [15]

Concept maps are based on the theory "that people think with concepts and that concept maps serve to externalize these concepts and improve their thinking." [13, p. 2] They have been used to help students learn, to gain a static picture of what they know, and to measure changes in student understanding. [5, 12, 18]

There have been several suggestions for how to evaluate concept maps. Some techniques rely on a quantitative analysis where the number of nodes, edges, etc. are counted and compared. Some look at the structure of the maps. Kinchin and Hay, for example, propose a classification of concept maps in which they identify three basic types: net, spoke and chain. [9] Other techniques take a more qualitative approach, looking at the meanings of the node and edge labels, the likeness to a predefined "master" concept map or the quality of the maps as evaluated by "raters." [11]

3. METHODOLOGY

Data were collected during the 2006-2007 academic year at six institutions in three countries. We collected 119 maps from 107 participants. The participants included 71 novices (first-year students who had been introduced to encapsulation, inheritance, and polymorphism), 12 intermediate students, 15 graduating students, and 9 instructors. A week before their final version, 12 of the novices did practice maps. The novices were from 6 different institutions; the other groups of participants were from only one or two institutions each. In this paper, we focus on the 71 novice maps (not including the 12 practice maps).

All participants were given a sample concept map that started with the concepts "kitchen" and "dinner" and the following task:

> Put the concept map here that starts with the two concepts "class" and "instance" with labeled arrows and other concepts that creates a partial map of object-oriented programming, as you have learned it so far.

In our **analysis**, we used some but not all of the techniques suggested in the literature. We counted the numbers of nodes, edges, and unlabeled edges, and computed the diameters of the graphs and the degree of the maximum-degree node in each graph. We examined the maps' structures, as suggested by Kinchin and Hay [9], but found that technique did not yield any useful information, as almost none of our graphs fell into their simplified categories.

We also applied quantitative techniques to the node and edge labels. First, we normalized the data, replacing labels like "contain," "contains," and "may contain" with a uniform "contains." Then we computed the most common node names, the most common edge labels, and the most frequent "sentences." Sentences are built by concatenating a node label, the label of one of its out-edges, and the other endpoint's label. For readability here, we adopt the convention that edge labels are contained in angle brackets. So for example, one of the sentences generated by the sample map shown in Figure 1 is "Class <has one or many> Instance."

We used qualitative techniques in searching for the OO concepts and misconceptions. Some terms such as "method" were explicit in most of the maps; others were present implicitly. For each topic, a pair of researchers agreed on what might indicate an implicit reference, closely examined the maps, and then compared their results. Any differences were resolved by discussion. More details are given in Section 4.

We did not compare the maps to a predefined master concept map. There are many possible good maps, and they differ considerably from each other. Nor did we attempt to rate the maps. For purposes of this analysis, we were more interested in examining what the students know than in ranking them relative to each other.

4. RESULTS AND DISCUSSION

Our analysis produced quantitative results having to do with graph topologies and frequencies of nodes, edges, and node-edge-node combinations. Structural features such as these may provide insights to the overall knowledge organization. Additionally, we examined qualitative details–whether concepts were implicitly present in a map, how groups of various related concepts are linked, and so forth. Finally, we consider how well concept maps worked to examine our questions.

Table 1 summarizes some structural information about the maps, viewed simply as graphs. The maximum degree

71 Novice Maps	Maximum	Minimum	Average
Number of Nodes	30	4	14
Number of Edges	38	4	16
Maximum Degree	17	2	6
Diameter	14	2	5

Table 1: Concept-map structure

	Most common	number
Node names	Class	75
	Methods	67
	Instance	67
	Variable	39
	Object	36
Edges	<–> *denotes unlabeled edge*	414
	<has>	92
	<contains>	85
	<can be>	72
	<is a>	70
Sentences	Class <contains> Methods	9
	Instance <–> Variables	8
	Class <–> Instance	8
	Instance <–> Class	6

Table 2: Most common labels and sentences

Concept	Explicit	Implicit	Total
Class	71	-	100%
Instance	67	0	94%
Method	63	4	94%
Data/attribute/ Instance variable	38	27	92%
Object	34	0	48%
Inheritance	23	13	51%
Encapsulation	2	16	25%
Modeling	5	3	11%
Polymorphism	5	0	7%
Abstraction	1	-	2%
Message Passing	0	0	0%

Table 3: OO concepts found

is the maximum of all node degrees in a map, treating the edges as undirected. The diameter is calculated as the maximum of the minimum path lengths between all pairs of nodes, ignoring edge direction.

There are 970 nodes and 1145 edges in total. Although the instructions asked for "labeled arrows," not all of the edges were labeled and directed. Of the 1145 edges, 76% were directed and 64% were labeled. Of the 71 maps, 43 (61%) had all edges directed, 22 (31%) had some edges directed, and 6 (8%) had no directed edges. All edges were labeled on 21 (30%) of the maps, 40 (56%) had some labeled, and 10 (14%) had no edge labels.

Table 2 lists the most common node names, edge labels, and sentences generated from the maps. Table 3 lists the OO concepts we looked for and the percentage of maps in which they were found. We considered a concept to be explicit if it appeared (normalized) as a node or edge label. Implicit references are discussed below.

4.1 Class, Object, and Instance

The concept maps suggest that students do not fully understand the meaning of Class, Object, and Instance. As instructed, all of the novices included Class in their maps. Only 67 included Instance (in spite of being instructed to do so). The remaining 4 students may not have read the instructions carefully, or they may have been less comfortable with the term. One student who did include Instance annotated the node, saying "not actually sure what that means." 34 students included Object, including 3 of the students who omitted Instance. 31 maps contained all 3 nodes.

Only 29 of the maps that contain both Class and Instance (43.3%) and 8 of the maps containing both Class and Object (24.2%) contain labeled edges that express what we considered a reasonable understanding of the concepts' relationships. In one way or another, 15 say that Class is used to create Instance, and 6, that Class is used to create Object. Two say that Instance <instantiates> Class, and one that Instance <example of> Class. Ten say that Class <has> Instance or Instance <belongs to> Class, and one that Class <has> Object. One says that Class <declares the instance variables> Instance - limited, but implying some understanding. One says Class <represents> Object, and another says Instance <represents> Object; these students seem to think of Object as the real-world thing being modeled.

Other edges between Class-Object and Class-Instance suggest a lack of understanding. Five students say that Class <contains> Instance, and one says Instance <is part of> Class. One says Class <is an> Instance, one says Instance <can be a> Class, and one says Class <is one of> Instance. One says Class <implements> Object.

The Object-Instance edges also suggest problems. Ten maps correctly equate the two (32.2%, including one that combines Instance and Object in a single node labeled Instance/Object.) Several maps have no direct connecting edge or an unlabeled edge. One says Instance <becomes an> Object. One says Instance <is a copy of> Object.

Four maps appear to identify Class and Object, referring to an Instance <of an> Object. One of these maps suggests a reason for this confusion. It connects Object with Superclass, noting that Object is the top Superclass. The fact that the root class of Java's hierarchy is named `Object` might well cause a Java student to equate Class and Object.

4.2 Class = data + behavior

On the other hand, these students do seem to understand that classes incorporate both data and behavior. They do *not* seem to have the classes-as-data-storage misconception: indeed, their maps focus more on the behavioral aspect of classes than on their data.

We considered Data, Attribute, and Instance Variables as explicit references to data, and Variable, Accessor, and Mutator as implicit mentions. We counted Method (by itself) as an explicit reference to behavior, and Function and references to particular kinds of methods (Accessor, Mutator, Constructor Method) as implicit.

While 88.7% of maps explicitly mention behavior, only 53.5% mention data. Counting explicit and implicit references together, they are nearly the same, with behavior at 94% and data at 92%. But (again counting explicit and implicit together), 71.6% of the maps have an edge between Class and behavior, while only 40.8% have an edge between Class and data.

4.3 Other important concepts

Encapsulation, inheritance, and polymorphism are arguably the three most important features of OO programming [16]; abstraction is fundamental to all of computing; message passing reflects the classic view of an OO program as a set of communicating objects.

We considered **encapsulation** to be implicit in any map that includes something about restricted visibility – primarily maps that include Private and Public connected to Method or Class. Only two students mentioned it explicitly and another 16 implicitly, about 25% of all the maps.

The implicit understandings were sometimes quite detailed; in one map, Private and Public were linked to a node Class* with the labels "Only available to one" and "Available to all." All of these implicit understandings showed Private, Public (and sometimes Protected) linked to appropriate concept(s) as different possibilities. A map that simply used the term "public" in a code fragment was not classified as having implicit understanding.

Implicit understanding of **inheritance** was exhibited by connections among nodes named class, subclass, or superclass, or the use of the word "extends." We counted 23 maps with explicit inheritance, and another 13 implicit – just over half of the maps.

Unlike other concepts, inheritance often showed up as an edge label (e.g., Subclass <inherits from> Superclass). Of the 23 maps with explicit inheritance, 12 had it as a node and 12 had it as an edge – one had both. For other concepts nearly all explicit mentions occurred on nodes.

Polymorphism showed up explicitly in only five maps and implicitly in none. We looked closely for implicit understanding, looking for things like method resolution, dynamic binding, actual type v. declared type, and so forth: it wasn't present.

Of the five that mentioned polymorphism, three showed a fairly deep understanding: one linked "polymorphism" to "interface," one linked "inherits/polymorphism" to "subclass," and one linked "that can do the same thing in different ways (polymorphism)" between "use interface to organize to standard" and "capabilities describe behaviors."

Abstraction showed up explicitly in only one map, as a node at the end of an unlabeled path Class <–> Inheritance <–> Abstraction, possibly showing an understanding how inheritance supports abstraction.

We decided not to tag for implicit understanding of Abstraction: many OO concepts *are* abstractions, including "Class," which was in all of our maps.

None of the novice maps referred to **message passing** between objects either explicitly or implicitly. We were fairly conservative in what we would consider to be implicit: we did not include references to methods calling other methods, as this could have been inside a single instance, but we would have included an instance invoking another instance's method, communicating with, sending a signal to, or interacting with another object–these were not in the maps. This complete omission is striking; it may reflect an unfamiliarity with the OO paradigm, or a change in the vocabulary used in introductory-level courses (see Section 5).

4.4 Modeling and programming in context

We would like our students to have some understanding of the context of the programs they write. Programs run on a computer, usually along with other programs, are compiled by another program, and store data in memory, they have users, and they are models of the real world. Indeed, there is some empirical evidence that the OO paradigm makes it easier for students to make the connection between a program and its domain.[14]

Accordingly, we looked for three things in the novice concept maps that might indicate a broader focus: the interaction between the program and other software or hardware; the interaction between the program and potential users; and the connection between the program and the real world.

We found strikingly few examples. Out of 71 maps, 25 (35.2%) included information that fell into one or more of these categories. Five were in two of the three, and none was in all three. Most of the novice concept maps focused narrowly on programming.

Only 18 maps referred to the hardware or software context in which the program runs. Three of these mentioned libraries, and one of those referred to the "3d party" who writes the libraries. Six maps referred to memory or data storage in some way, one mentions the keyboard, and seven maps (all from the same institution) mention threads. Only four maps made any reference to users.

Only 8 maps mentioned modeling. Three of the references were implicit: Attributes Describe What is, and Capabilities Describe Behaviors (two nodes in the same map); Object <should> Resemble Reality; and Instance <is a> Single Person, Object, or Event.

4.5 Quality of data and analysis

Collecting data at multiple institutions in multiple countries is a good way to maximize the applicability of results. There are aspects of this work, however, that may reduce the validity of the results.

The concept maps were gathered in different settings: at one institution as a part of an examination, at another as a voluntary task after lecture on a Friday afternoon. The work put into making the concept maps varied, and the average time spent on the maps was just a few minutes. The maps might be different if the students spend more time reflecting on the task. On the other hand, the short time spent on constructing the maps may make students write down what is on top of their mind, reflecting their "practical knowledge" of the subject.

It is also possible that the level of familiarity with concept maps affected the result. We do not know which students (if any) had previous familiarity with concept maps. We do have a small number of data points – 12 students who did the maps twice a week apart. Although they did not receive any feedback in between, preliminary analysis indicates that having done the map once and having time to think about it made a difference in the maps.

5. IMPLICATIONS FOR TEACHING

Learning the concepts of OO programming and the language to describe them is a big job, and these students are just beginners. They seem to fixate on language details, losing the big picture, no matter how much we stress all of these concepts. This may be, at least in part, the natural result of inexperience.

Choosing a new book is probably not the solution. We reviewed several major Java texts, including those used by these students, and they covered all of these concepts (with one exception, discussed below). They covered them in different orders, and with different examples, but, generally in a clear, understandable way. [2, 3, 8, 10, 16, 17]

There are some steps we can take as instructors, however. First, we can emphasize the difference between Class and Instance/Object as often as possible. Stick with a single explanation – the "blueprint" metaphor may be a good choice, as it is popular with these students – and give lots of examples. Second, explicitly tell students that Instance and Object are nearly always the same. Mention that Object sometimes refers to the thing in the real world that is modeled by an Instance, and address the confusion that can be caused by the name of Java's top-level `Object` class.

Additionally, we can talk about objects interacting and communicating with each other. This is the one topic that is rarely mentioned in textbooks (for an exception, see [16]). There is no obvious term other than "message passing" and this terminology seems to be out of fashion as it was almost never found in the books' glossary or index.

Lastly, making concept maps seems to be a useful exercise. [9] It gives students a chance to think about OO and see connections.

6. CONCLUSIONS AND FUTURE WORK

Our analysis confirmed earlier research indicating that students do not have a firm grasp on the distinction between "class," "object," and "instance." Earlier results suggested that some students think of classes as just being data storage (like arrays or structs); we found that while many students do connect classes with data, even more make the connection to behavior. Students rarely included any mention of the hardware/software context of programs, their users, or their real-world domains. Although OO may make it easier for students to connect programs and real-world domains, as some have found [14], and both textbooks and instructors make that connection, few students see modeling as one of the most important OO concepts. Students do mention inheritance, but not encapsulation or abstraction. And the picture they draw of OO is a static one: there is almost nothing that could be construed as referring to interaction among objects in a program.

In future work, we plan to compare the first and second maps of students who draw them twice about a week apart, to try to isolate the effects of familiarity with the technique. We would like to follow up some of the maps with more in-depth interviews. And we plan to gather additional maps from upper-level students, in order to examine whether the upper-level-student understanding of OO concepts has been transformed by their additional experience.

7. ACKNOWLEDGMENTS

Thanks to our students for inspiration and data, and to Uppsala University and Umeå University for physical and electronic meeting places.

8. REFERENCES

[1] D. J. Armstrong. The quarks of object-oriented development. *Communications of the ACM*, 49(2):123–128, 2006.

[2] D. Barnes and M. Kölling. *Objects First With Java: A Practical Introduction Using BlueJ*. Prentice Hall, 3rd edition, 2006.

[3] H. Deitel and P. Deitel. *Java How to Program*. Prentice Hall, 7th edition, 2007.

[4] A. Eckerdal and M. Thuné. Novice java programmers' conceptions of "object" and "class", and variation theory. *SIGCSE Bull.*, 37(3):89–93, 2005.

[5] E. Ferguson. Object-oriented concept mapping using UML class diagrams. *Computing in Small Colleges*, 18(4):344–354, 2003.

[6] S. Garner, P. Haden, and A. Robins. My program is correct but it doesn't run: a preliminary investigation of novice programmers' problems. In *7th Australasian conf. on Computer Education*, pages 173–180, 2005.

[7] S. Holland, R. Griffiths, and M. Woodman. Avoiding object misconceptions. *SIGCSE Bull.*, 29(1):131–134, 1997.

[8] C. S. Horstmann. *Java Concepts for Java 5 and 6*. John Wiley & Sons, 5th edition, 2007.

[9] I. Kinchin, D. Hay, and A. Adams. How a qualitative approach to concept map analysis can be used to aid learning by illustrating patterns of conceptual development. *Educational Research*, 42(1):43–57, 2000.

[10] J. Lewis and W. Loftus. *Java Software Solutions: Foundations of Software Design*. Addison Wesley, 5th edition, 2006.

[11] J. McClure, B. Sonak, and H. Suen. Concept map assessment of classroom learning: Reliability, validity, and logistical practicality. *Journal of Research in Science Teaching*, 36(4):475–492, 1999.

[12] J. Nash, R. Bravaco, and S. Simonson. Assessing knowledge change in computer science. *Computer Science Education*, 16(1):37–51, 2006.

[13] J. Novak and D. Gowin. *Learning How to Learn*. Cambridge University Press, 1984.

[14] V. Ramalingam and S. Wiedenbeck. An empirical study of novice program comprehension in the imperative and object-oriented styles. In *ESP '97: Seventh Workshop on Empirical Studies of Programmers*, pages 124–139, 1997.

[15] K. Sanders and L. Thomas. Checklists for grading object-oriented CS1 programs: concepts and misconceptions. *SIGCSE Bull.*, 39(3):166–170, 2007.

[16] K. Sanders and A. van Dam. *Object-Oriented Programming in Java: A Graphical Approach*. Addison Wesley, 2006.

[17] W. Savitch. *Absolute Java*. Prentice Hall, 3rd edition, 2007.

[18] M. Steyvers and J. Tenenbaum. Graph theoretic analyses of semantic networks: Small worlds in semantic networks. *Cognitive Science*, 29:41–78, 2005.

[19] B. Thomasson, M. Ratcliffe, and L. Thomas. Identifying novice difficulties in object oriented design. *SIGCSE Bull.*, 38(3):28–32, 2006.