Students’ Understandings of Storing Objects

Juha Sorva
Department of Computer Science and Engineering
Helsinki University of Technology
Espoo, Finland
Email: jsorva@cs.hut.fi

Abstract
This paper reports a phenomenographic study of how introductory students view objects that have been created and stored by an object-oriented program. By analyzing student interviews, we identify five categories of description, each representing a different kind of understanding of the phenomenon. Of these categories, some represent viable understandings that we would like our students to have. Others are partially incorrect and indicate that some students mistakenly focus their awareness on aspects that are unhelpful or even harmful for constructing a viable mental model of storing objects. This paper brings together two previously disjointed branches of computer science education research: the study of misconceptions and the phenomenographic research approach. The phenomenographic approach used in this study extends traditional phenomenography by including partially incorrect understandings in a phenomenographic outcome space, and explicitly treating them as such. This approach offers a new way of studying misconceptions and linking them to correct understandings of a phenomenon.

Keywords: objects, students’ understandings, misconceptions, phenomenography, constructivism, CS1

1 Introduction
1.1 Background
Object-oriented programming is complex and abstract. According to constructivist theory, beginning programmers construct different kinds of mental models of the underlying layers of abstraction (Ben-Ari 2001). Some of these student-constructed models will be ‘correct’ or at least viable in practice, some of them will be unorthodox but relatively harmless in terms of skill development, and some of them will be just plain wrong in the ‘cold hard world’ of computing, with its well-defined concepts and deterministic programs.

Du Boulay (1986) notes that some students’ difficulties are associated with their limited understanding of “the general properties of the machine [they] are learning to control, the notional machine”. Indeed, it is very plausible that non-viable mental models arise out of students’ implicit assumptions about the machine that their programs are supposed to instruct. A notional machine for object-oriented programming is significantly more complex than one for procedural programming (Sajaniemi & Kuittinen 2007). While object-oriented programming is routinely taught to beginners, not all aspects of the notional machine are always adequately addressed in teaching. Programming instructors often face the task of helping students construct viable models of the notional machine and steering them clear of non-viable ones. In order to do this, it is useful to be aware of the educationally critical aspects of the notional machine and to know what pitfalls to look out for.

This paper will not attempt to tackle the entire object-oriented notional machine or students’ understandings of it. Instead, it will explore a particular subquestion as described below.

1.2 Research Question
During Spring 2007, we initiated an explorative study of how introductory programming students view program execution, the notional machine, and computer memory. The research presented in this paper is part of this project, and focuses on a specific, narrow research question, namely:

In what qualitatively different ways do introductory-level students understand the idea of an object stored in memory?

In other words, the point of interest is what students think happens when an object is created and ‘placed in the computer’. What is it that exists within the computer after such an operation? This research does not focus exclusively on correct understandings of the phenomenon or on unviable ones (misconceptions), but will discuss them both. As indicated by the research question, the point of view is qualitative, and the aim is to discover and enumerate different understandings, not to assess how common particular understandings may be.

This paper is structured as follows. The next section introduces some related work both on misconceptions and partial understandings of object-oriented concepts and on the phenomenographic research approach used in the present research. The methods used for collecting and analyzing data are described in Section 3. Section 4 presents the results and Section 5 discusses their implications as well as the validity of the research. Section 6 concludes the paper and looks at directions for future work.

2 Related Work

2.1 Misconceptions and Partial Understandings
Several studies have been reported that explore the ways novice programmers misunderstand concepts in object-oriented programming.

Holland et al. (1997) noted several misconceptions introductory students have about objects. For instance, students may conflate the concepts of object and class, or they may mistake an instance variable storing a name string for an object id. Holland et al. discuss the possible sources for these misconceptions and suggest potential pedagogical solutions. This work is based on anecdotal but intuitively appealing evidence gathered while developing distance education courses.
Fleury (2000) interviewed students on an introductory programming course and found that students form their own rules of what happens and what works in Java programming. For instance, some students thought that the dot operator could only be applied to methods and that programming. For instance, Fleury makes the interesting observation that ‘rules’ students construct about programming are essentially more limited versions of the viable rules we would like them to learn: “the removal of ‘only’ [...] from each of the student constructed rules makes it a true statement” (Fleury 2000). A variation theorist would attribute such misconstrued rules to a lack of perceived variation, which limits the student’s ability to understand the phenomenon in question.

Phenomenography does not prescribe a particular method for gathering or analyzing data. Nevertheless, traditions within the phenomenographic community contribute towards something that could be called a ‘typical phenomenographic research methodology’. In a research project of this kind, interviews are used as a data collection method. Data collection is followed by an intensive qualitative data analysis. During analysis, the researcher, in dialogue with the data, delimits the phenomenon of interest. Different ways of understanding or experiencing the phenomenon are enumerated as an outcome space consisting of a (smallish) number of categories of description. The intention is not to point out which specific kinds of understanding each individual has, but to identify different ways in which the phenomenon can be understood or experienced on a collective level. These categories of description, which arise from the data, represent partial ways of understanding the phenomenon, are logically connected to each other, and are often presented in the form of a hierarchy or tree. An individual person may understand a phenomenon in a number of the different ways represented by the categories of description.

Phenomenography was originally developed outside the ‘hard sciences’, and it seems natural that it traditionally does not problematize the ‘correctness’ of understandings. Many phenomenographic studies aim to discover an array of different, perhaps unorthodox, but nonetheless valid perceptions of a phenomenon. If you investigate people’s perceptions of, say, love, discussing the ‘correctness’ of those perceptions is a lost cause. However, in hard sciences such as computer science, there are concepts that are relatively well defined, and we can say that some understandings are correct and others incorrect. Outcome spaces in phenomenographic projects that deal with such concepts tend to include only understandings that are ‘correct’ in the sense that the researcher deems them to be ‘true statements’ about the phenomenon, i.e., understandings that conform to the intended learning outcome. Wholly, or even partially, incorrect understandings (‘misconceptions’) seem to be commonly discarded by phenomenographers investigating learners’ understandings of computer science concepts. The incorrect understandings are considered not to be related to the phenomenon under study but rather to some other, perhaps imaginary, phenomenon. For instance, an incorrect understanding in which objects are considered parts of a class could be discarded from the outcome space.

Even where it is possible to distinguish between correct and incorrect understandings, concentrating only on the correct understandings is both enough and convenient for some phenomenographic projects, and allows the researcher to delimit the phenomenon more cleanly and to produce neater hierarchies that are arguably easier to report, understand, and make use of. Nevertheless, in computer science education the question of how people misunderstand phenomena is interesting and pedagogical.
cally relevant, and the kind of in-depth interviews suitable for phenomenographic analysis often produce intriguing data about incorrect understandings that learners have. It is this author’s impression that the research community is divided in terms of how it views the compatibility of the phenomenographic research approach with studies of incorrect understandings. At one extreme, there are those who argue that incorrect understandings can and surely should be included in phenomenographic research; in fact, making judgements on the correctness of people’s understandings is questionable within the phenomenographic framework. An opposite take on the matter is that including incorrect understandings in outcome spaces compromises both the validity of phenomenographic research and the usefulness of the results, which are unlikely to cover very many of the plethora of different misconceptions that people have. This author finds that the prospect of including incorrect understandings in phenomenographic outcome spaces is a promising (yet not unproblematic) one, and will explore it in this paper.

3 Research Setting and Methods

The results presented in this paper on students’ understandings of storing objects arose as a part of a wider project that investigates students’ understandings of memory and program execution. As this research project explores relationships between people and various phenomena, and aims for qualitative results, phenomenography naturally suggested itself as a research approach. This choice of research approach was also influenced by personal circumstances and the proximity of experts in phenomenography.

Following phenomenographic tradition, the advice of experts, and personal intuition, semi-structured interviews (Kvale 1996) were chosen as the data collection method. The following subsections describe the selection of interviewees, the structure of the interview sessions, and the methods used for analyzing data.

3.1 The Students

The interview subjects were from a semester-long university course in introductory programming. The course teaches programming in Java in an objects-early way, yet without going deep into object-oriented design or complex object interactions. Apart from drawings in lectures, no tools visualizing the notion of machine or computer memory are used in the course, and these topics are given rather little attention during the course. The course is taken by engineering students who are not computer science majors. The author of this paper (that is, the interviewer), while a teacher at the same department that gives the course, did not directly participate in running this course.

Approximately one-third through the semester, all students participating in the course were required to complete an online questionnaire about their programming background prior to the course they were taking now and about their attitudes, experiences and workload during the course. Of the several hundreds of respondents, a small subset was selected and invited for interviews based on this background questionnaire. In order to capture a wide range of qualitatively different understandings, the interviewees were hand-picked so that there were interviewees with different kinds of programming background (though most had no prior experience), different kinds of attitudes to programming and different experiences with the course.

Initially, 14 invitations were sent to the selected students via email. Each student was promised two movie tickets as a reward for a one-hour interview. The invitations stressed that the interviews were not a part of the course and would not affect grading in any way. Nine of the students agreed to take part, but one later cancelled due to scheduling problems. Two more students were selected and sent invitations; both agreed to take part, and the target total of 10 interviewees was reached.

3.2 The Interviews

The author of this paper interviewed the students approximately half-way through the programming course the students were taking. The interviews were recorded in audio. The interviews were done in Finnish; all interview quotes in this paper have been translated from the Finnish originals.

Each interview began with a short discussion of what program execution means in general. The bulk of the interviews was spent discussing more specific issues related to program execution, computer memory, method invocation, etc. This discussion was organized around two example Java classes. First the students were shown a relatively simple class representing elevators. Where time allowed, a composite class representing buildings that contain elevators was also discussed. Each class also had a main method and formed an executable program. The students were shown the code on paper and asked to describe what they saw there. Unless they spontaneously described what happens when the given programs are executed, they were prompted to do so.

During each one-hour interview, a number of programming topics arose. Not quite the same set of topics was discussed with each student. However, there were a number of focal topics that the interviewer introduced in each interview. One of these, of interest in this paper, was the act of storing ‘objects’ (though students did not necessarily use that term) in the computer. When the subject of creating objects came up, each interviewee was asked to elaborate on this in more detail with questions such as: What does it mean to have an object created? Does the object exist after it is created? What does the computer know about an object that it has created?

After each interview, the interviewer wrote down some early thoughts about what had been said and about the atmosphere of the interview. In some cases, a brief teaching session also took place, as the interviewer could not bear to let the student leave the room while harboring misconceptions.

3.3 Data Analysis

The interviewer transcribed each interview and added it to a pool of data, which was analyzed with the different understandings of the collective in mind. The transcription and analysis process started right after the first interview, so that the analysis could provide feedback and ideas to the rest of the interviews.

The data was analyzed with a phenomenographic mindset, with the goal of forming an outcome space of qualitatively different categories of description. The initial analysis was done by the author, then given for a second opinion to a colleague who had also read the transcripts. The resulting discussion brought about some refinement in the category definitions, but no radical changes.

During analysis, it turned out that the interviewed students displayed not only correct understandings of storing objects, but also a number of interesting but incorrect understandings. While a typical phenomenographic analysis (at least in the field of computer science education) might have discarded patently incorrect elements from the data, we chose to include them inasmuch as they had something to do with our phenomenon of interest, storing objects in memory.

4 Results

As a result of the analysis, an outcome space with five categories of description was formed. The categories of
Storing an object is understood as type-specific properties of some kind. The object’s instance variables, the name of the variable that the object was most recently assigned to, a class’s program code, and the constructor parameters given upon object creation.

<table>
<thead>
<tr>
<th>Category</th>
<th>Storing an object is understood as being related to...</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PROPERTIES</strong></td>
<td>type-specific properties of some kind.</td>
</tr>
<tr>
<td><strong>INSTANCEVARS</strong></td>
<td>the object’s instance variables.</td>
</tr>
<tr>
<td><strong>VARNAMES</strong></td>
<td>the name of the variable that the object</td>
</tr>
<tr>
<td><strong>CODE</strong></td>
<td>a class’s program code.</td>
</tr>
<tr>
<td><strong>PARAMETERS</strong></td>
<td>the constructor parameters given upon object creation.</td>
</tr>
</tbody>
</table>

Description are labeled according to the main focus of awareness in each category: Properties, InstanceVars, Varnames, Code, Parameters. Table 1 gives an overview of the categories, which are described in more detail and illustrated with quotes in the subsections that follow.

4.1 Category: Properties

Storing an object means storing a chunk consisting of object properties of some kind, as defined by the object’s type.

This category describes a general idea that objects are stored within the computer. Each object is defined by some kind of properties, which describe what the object is like. The computer remembers each object’s properties somehow in order to make use of them later. Anne observes:

**Interviewer:** So an object is created? How do you perceive that?

**Anne:** Well, it creates like... a chunk or thing which has some properties.

The focus in this category is on objects as multiple composite chunks of data stored in the computer. Objects are not perceived as identical; rather, different objects may have different properties. Distinct objects with different properties constitute values along this vaguely discerned dimension of variation, but it may be unclear exactly what causes an object to have certain properties or what exactly constitutes an object property. How or where objects are stored can also be poorly understood, as illustrated by Anne’s response when queried where she thinks objects are after they are created.

**Anne:** Well, it’s some unit of information... a part of the machine or... not a part but...

[pause]

**Interviewer:** Where is the object then, or...?

[pause]

**Interviewer:** If it’s created, then it exists somewhere somehow?

**Anne:** Yeah, well, somehow within the program, within the computer.

Another dimension of variation is discerned that pertains to object types. Objects of different types can exist and it is important for the computer to know what type each object is in order to determine what the object is like and how it behaves. The type of an object is linked to what kind of properties are stored for it and how the object can be used.

Practically all of the interviewees seemed to consider an object’s type to be something that the computer knows for each object, and made implicit use of this fact when describing program behaviour and what properties are stored for each object. However, few if any put this notion explicitly into words. Ian perhaps comes the closest – he first notes that a newly created object is “an elevator object... an object of the elevator kind”, then explains what happens when a method is invoked on the object.

**Ian:** [The computer] sees that “Gee, this [object] is of the class Elevator”. Then it goes: “What do I do now?”. And then the code says ‘dot getFloor’, so [the computer goes]: “That’s there in the methods [of the class]! Okay.”

All the other, more complex understandings represented by the other categories of description extend this rudimentary type of understanding.

4.2 Category: InstanceVars

Storing an object means storing the object’s state as the values of the object’s instance variables.

This kind of understanding extends the general understanding of object properties described above, and adds a more sophisticated understanding of what it is that defines a particular type of object’s properties. A new dimension of variation is discerned that focuses on the values of instance variables. Here is Greg describing what happens when the given elevator class (which has two integer instance variables, floor and topFloor) is instantiated:

**Greg:** In the main method, it creates an elevator – this testElevator – so then it reserves memory slots for the floor and the highest floor, and assigns them values.

Dave explains the contents of an array of elevator objects.

**Interviewer:** You said earlier that there are like many memory slots in the array? So what is in those now when these elevator objects have been placed there?

**Dave:** You mean what there is in one slot, for instance?

**Interviewer:** Yeah.

**Dave:** Well, it holds this elevator object, meaning that it contains all the data that the elevator class contains. It has to have the data showing which floor it’s on, and the top floor. Those it has to have at least. So it has two such... ‘bits of information’ in this case. Such integers that it stores for each element.

Brad describes the movement of an elevator as something that changes the state of the elevator object. (The elevator moves via its method move, which receives a destination floor as an integer parameter and assigns a new value to floor accordingly.)

**Interviewer:** So how do you understand what happens when the elevator is moved?

**Brad:** Well, it has... again, stored in some memory slot the... number where it is at. And in practice, I guess this assignment statement within the method move means to the computer that a value is assigned again to the memory slot that floor refers to. A new value is assigned to the same memory block that used to hold zero, replacing the old one.

Brad’s understanding includes a notion of changeable object state, that is, a discernment of variation in how program behaviour is dependent on and affects the stored properties of objects.

---

1Interviewee names changed.

2Over the course of the interview, Greg expressed multiple different, sometimes contradictory, views about storing objects and program control flow. He seemed to fluctuate between understandings, perhaps indicative of a liminal state (Meyer & Land 2005, Eckerdal et al. 2007).
4.3 Category: \textbf{VARNAME}

\textit{Storing an object means storing the object’s state as the values of the object’s instance variables, and the name of the variable that the object was most recently assigned to.}

This category, which further extends \textsc{INSTANCEVARS} described above, represents an understanding that focuses on the assignment of objects to variables. Specifically, the name of the variable that an object is assigned to is considered a property of the object, and is stored with the rest of the object’s state. A dimension of variation is perceived along which different relationships between objects and variables constitute different values. An object is seen as being different from another if it has been assigned to a differently named variable. As Ian describes a scenario where an elevator object has been created and assigned to a variable named testElevator, he first alludes to the elevator being “named”:

\begin{quote}
\textbf{Interviewer}: If you think specifically about this scenario with the elevators, what about the elevator is in memory here?
\textbf{Ian}: Well, there’s precisely that there exists an elevator named testElevator. It has been given the value six. I mean, it has been given the value of the top floor which is five and then this.floor which is zero at the moment.
\end{quote}

Elsewhere in the interview, Ian describes elevators that are stored in an array, which he likens to an Excel worksheet.

\begin{quote}
\textbf{Interviewer}: So in each ‘cell’ of the ‘worksheet’ you have…?
\textbf{Ian}: The data for one object.
\textbf{Interviewer}: So if it’s an elevator, that means…?
\textbf{Ian}: Well… it would have the this.floor of the elevator and the top floor… and then the name, I guess… Yeah.
\end{quote}

In the example program elevators are repeatedly added to such an array using the following lines:

\begin{verbatim}
Elevator newElevator = new Elevator(this.height - 1);
this.elevators[this.numberOfElevators] = newElevator;
this.numberOfElevators++;
\end{verbatim}

Ian’s description of this code illustrates that ‘object names’ are not necessarily unique identifiers of objects.

\begin{quote}
\textbf{Interviewer}: You mentioned that a new elevator is created, with the name newElevator?
\textbf{Ian}: Yeah. So the [two elevator objects so far created and placed in the array] have the same name as a matter of fact.
\textbf{Interviewer}: There are several elevators named newElevator in the array, or?
\textbf{Ian}: Yeah, there are.
\end{quote}

Greg is another interviewee who considered variable names to be part of an object’s data. For Greg, repeated assignment of an object value involves changing the object’s name. Below, Greg examines the statement Elevator test1 = office.orderElevator(2); He has established that the method call finds the closest elevator in a building, moves it to the given floor and returns it. He then explains what happens to the elevator:

\begin{quote}
\textbf{Greg}: I suppose it changes its name to be test1, then…
\textbf{Interviewer}: Changes the name of what to be test1?
\textbf{Greg}: The closest elevator which it fetches with this [method call].
\textbf{Interviewer}: It renames the memory slot to be test1?
\textbf{Greg}: Renames the elevator that’s in the slot. Since it had been assigned the name newElevator, it changes it to test1.
\textbf{Interviewer}: All right.
\textbf{Greg}: And I suppose its floor also changes as it’s ordered.
\end{quote}

For Ian, reassignment means making a copy of the elevator, only with a different name.

\begin{quote}
\textbf{Ian}: The elevator that has gone there [to the floor that it was ordered to]… It decides that its name is now test1.
\textbf{Interviewer}: So the name of the elevator changes?
\textbf{Ian}: Yeah it changes to test1… Mmm, or I mean no… I mean now an elevator named test1 exists in addition to the original still being there in the array.
\end{quote}

4.4 Category: \textbf{CODE}

\textit{Storing an object means storing a copy of the program code of the class that defines the object’s type.}

This category places a focus on the program code that defines the object’s type, its instance variables and methods. This code is perceived as being included in the properties that define an object. Eric has this view:

\begin{quote}
\textbf{Interviewer}: So what is there after it’s created it in memory? How do you think about it?
\textbf{Eric}: Well, there in memory are at least all these instructions in the code, all these characters and spaces and newlines.

Whenever an object is created, the computer is understood to take the class definition and store a copy of it. A dimension of variation is perceived where distinct copies of code are characteristic of distinct objects. For instance, Greg pointed out that for each element in an elevator array, all the code in the elevator apart from the main methods needs to be stored.
\end{quote}

4.5 Category: \textbf{PARAMETERS}

\textit{Storing an object means storing the constructor parameters given upon object creation. These parameter values unambiguously define the object’s properties.}

This category represents an understanding that focuses on constructor parameters as the defining properties of objects. Not only do an object’s creation parameters have an effect on initializing the object, but the object is thought to be stored in memory in terms of those parameters. A dimension of variation is discerned where the objects created with different parameter values constitute examples of different objects in computer memory. Fred shows an example of this kind of understanding:
Fred: It remembers what value that particular elevator object has received for the parameter.

Interviewer: Which parameter?

Fred: (Points at new Elevator(5) in the given code.) Like, here where an elevator was created and received five as a parameter.

Fred’s understanding of object properties is limited to the PARAMETERS category. He does not have a notion of a persistent object state. In keeping with the idea of storing the constructor parameters for each object, he considers an object to “start with a clean slate every time [any of its methods] is called”. Whenever an object’s method is called, Fred consistently considers its execution to start with the object in whatever state the class’s constructor sets for it. For him, method calls do not do have a lasting impact on state and even assignments to instance variables last only as long as the current method call.

4.6 Relationships between Categories

In the category PROPERTIES, the focus is on objects as ‘chunks’ of data in the computer. Variation is discerned in how these chunks differ in terms of some type-specific properties. Variation in what different kinds of properties objects have is linked to variation in the objects’ types. However, while variation in object properties is discerned, the relationship of those properties to the program that determines the objects is not necessarily discerned at all. Variation in object properties is vaguely if at all linked to program structure (e.g. instance variables). This kind of understanding is viable and not incorrect, but by itself not enough for understanding how the computer handles objects in object-oriented programs. This category serves as a basis for other categories of description, which extend this rudimentary understanding of storing objects and link object properties to specific features of program structure.

The category INSTANCEVARS places a focus on the instance variables of an object. They are seen as key features that define what a computer stores about an object. Variation in objects’ properties is discerned as being linked to variation in the values of objects’ instance variables. This is a viable, more concrete and useful extension of the category PROPERTIES.

An understanding of the VARNAME variety is characterized by mistaking a critical feature (name) of one concept (variables) for a critical feature of another concept (objects). This is an incorrect ‘over-extension’ of INSTANCEVARS that focuses on assignment operations and links variation in assignments to variation in objects’ stored properties. A name that an object has been assigned to is seen as part of the object’s state when in fact assignments are irrelevant to object state. This kind of understanding is a symptom, cause, or both, of not being able to distinguish between object-valued variables and object representations in memory. In other words, the category is related to the often-reported inability of students to grasp the idea of references, which are known to be a difficult topic in learning programming (Adcock et al. 2007). Thinking of a variable name as a part of an object’s state is likely to result in a non-viable mental model of references. For instance, Ian has a mental model where dot notation means that the computer looks through the contents of memory to find an object with a matching name. This misunderstanding is also related to the ‘identity/attribute confusion’ reported by Holland et al. (1997) and Ragonis & Ben-Ari (2005).

The category CODE includes the entire code of a class within the focus of awareness. This category extends PROPERTIES: it encompasses the notion that a computer needs to know each object’s type at runtime, and adds a dimension of variation where a separate copy of the same code is attributed to each object of a particular type. According to this kind of understanding, it is storing the code that is key to defining what an object is. While the idea of storing each object’s runtime type is correct, the notion that each object has a copy of the class’s code is not. Still, this misunderstanding – which has some intuitive appeal – may be a viable way to think about the matter in what comes to introductory programming, and may be relatively harmless in terms of CS1 studies. However, having this understanding may also be indicative of difficulties in distinguishing between classes and objects, an often-reported beginner problem.

The category PARAMETERS extends and concretizes PROPERTIES in yet a different way. The focus in this category is – incorrectly – placed on the values of constructor parameters. A relationship is discerned between variation of construction parameters and variation in what is stored about an object, so that the former fully defines the latter. A person with this kind of understanding is unlikely to form an understanding about object state, and will probably find it relatively hard to see the competing, even contradictory (but correct) idea of storing objects’ instance variables. This was certainly the case with Fred, quoted in the previous section.

Table 2 summarizes the five categories of description. The relationships between the categories are illustrated in Figure 1.

Figure 1: Relationships between categories of description. Each line indicates that the category below encompasses and extends the category above.

typically, in a phenomenographic outcome space that takes the shape of a tree, there are multiple branches that represent partial but correct understandings, which are extended by the ‘root’ of the tree, which represents a more complete way of understanding the phenomenon. The reader may note that Figure 1, while tree-shaped, is not typical in this sense. Instead, only INSTANCEVARS represents a correct, more complete extension understanding of PROPERTIES. It encompasses all the correct understandings represented in the diagram, and in this sense corresponds to the rich ‘root’ understanding in a more typical outcome space. The other three categories ‘branching out’ from the two correct categories represent partially incorrect understandings. In Figure 1, there is no one category which would encompass and extend all these branches (since there is no category of understanding which would encompass the various partially incorrect understandings).

5 Discussion

5.1 On Incorrect Understandings in Phenomenographic Results

Of the five categories of description that emerged from the data, PROPERTIES and INSTANCEVARS represent viable understandings that we would like our students to have. These understandings are ‘true’ in the sense that they correspond to the actual technical reality of the computer. To learn object-oriented programming, it is crucial to discern that object types and instance variables are key to how objects exist within the computer. Not all students discern
Storing an object means storing a chunk consisting of object properties of some kind, as defined by the object’s type. Storing an object means storing the object’s state as the values of the object’s instance variables. Storing an object means storing the object’s state as the values of the object’s instance variables, and the name of the variable that the object was most recently assigned to. Storing an object means storing a copy of the program code of the class that defines the object’s type. Storing an object means storing the constructor parameters given upon object creation. These parameter values unambiguously define the object’s properties.

In Figure 1, each of the three categories that ‘deviates’ towards the right extends a correct understanding in some incorrect way. That is, these understandings – PROPERTIES, INSTANCEVARS, and PARAMETER – are partially incorrect. They represent understandings that do not correspond to the reality of the computer and that are, to varying degrees, harmful for learning programming. A learner with one of these partially incorrect understandings is focusing their awareness on something that is not helpful for reaching the intended learning outcomes. They mistakenly link to the phenomenon of storing objects a dimension of variation that is unwanted from the pedagogical point of view. The problems that a learner faces if they associate incorrect dimensions of variation with a phenomenon may be further compounded by the inability to discern the correct dimensions of variation.

In phenomenography, the researcher, drawing on the data, decides what falls within the scope of a phenomenon. This is to be done in such a way that the categories in the outcome space can be said to describe different understandings of one phenomenon. We find that in our case, it is not only acceptable but in fact important to include both correct and partially incorrect understandings of the phenomenon. Since students’ understandings of storing objects can clearly be incorrect in pedagogically interesting ways, it behoves us to include these incorrect understandings when enumerating qualitatively different ways of understanding the phenomenon. Linking the incorrect understandings to the correct ones seems like a promising way to make sense of them systematically. As discussed in Section 2, this research is not typical phenomenography as it is applied to CS phenomena, and the inclusion of partially incorrect understandings in the outcome space raises some pertinent questions.

There is a limitless number of understandings – and misunderstandings – of each phenomenon. Phenomenography posits that there are not very many qualitatively different understandings. Does this change if we include partially incorrect understandings in the scope of our studies? Are we researching multiple phenomena while pretending to research just one? Are we faced with incomprehensible outcome spaces with countless categories of description? We argue that the inclusion of partially incorrect understandings is a valid application of the phenomenographic research approach. All the understandings reported in this paper share a correct core in that they include an understanding of an object stored in the computer as a collection of properties that depends on the type of the object. This correct core links all the understandings reported above and serves to delimit the phenomenon. Care must be taken to determine that the incorrect understandings pertain to the actual phenomenon under investigation and are not completely unrelated figments of people’s imaginations. This is a challenging task, but not one that is new to phenomenography: the phenomenographer always has the responsibility delimiting the phenomenon of interest, be it ‘love’, ‘the decision-making process in our company’, or ‘storing objects in the computer’. We argue that the correct part of an outcome space, which represents correct understandings of a phenomenon (e.g. PROPERTIES and INSTANCEVARS), can be used, as above, to help distinguish between partially correct understandings and wholly incorrect, unrelated understandings. This helps delimit the phenomenon and the size of the outcome space even when incorrect understandings are included in a study.

Another issue worth noting is coverage. It seems to be possible, even in a small-scale study, to construct an outcome space with a good coverage of the correct understandings that one can expect to find in a similar group of people. If partially incorrect understandings are included in the study, the size of the task increases, and it is likely that a lesser coverage will be achieved. Nevertheless, such a study can discover valuable qualitative information, contributing towards a body of knowledge that other studies – qualitative and quantitative – can complement. It is also worth noting that not all incorrect understandings are equally common nor pedagogically relevant; the more common ones are likelier to be discovered first.

A future challenge to phenomenography could be to investigate people’s understandings and misunderstandings of the relationships between phenomena. Many beginners have trouble distinguishing phenomena (e.g. object/class, object/variable) from one another, and their understandings of one phenomenon may confute with their understandings of another. As we explore understandings of one phenomenon, we end up ‘charting the borders’ that the phenomenon has with other phenomena and exploring overlapping aspects of multiple phenomena. This poses challenges to phenomenography, but we feel that further work on the research approach could help produce interesting insights into people’s understandings and misunderstandings.

5.2 Pedagogical Implications of the Results

In light of constructivist theory and the results presented above, it seems clear that people construct a number of different interpretations of how a computer keeps track of object data. Especially when little explicit attention has been paid to the notional machine in teaching, as was the case on the CS1 course we investigated, students are liable to come up with their own models of it. Sometimes they get it right, sometimes they don’t. From our data, it is obvious that some students had constructed incorrect models of storing objects in the computer.

The results presented in this paper highlight two kinds of learning difficulties. On one hand, some students are not able to focus on all the important aspects of storing objects and do not link the phenomenon to the variation in the values of instance variables as we would like them.
to. As programming instructors, we need to draw students’ attention to the important aspects and variation in those aspects, and underline their importance where possible. Visualizations in course materials or visualization tools — e.g. Jeliot (Moreno et al. 2004) or the BlueJ object bench (Kölling et al. 2003) — can help here by making explicit and visible the idea that objects’ states are defined by those objects’ instance variables.

On the other hand, students may also mistakenly focus on irrelevant variation and mistake it for critical variation. Here, one task is to draw the students’ awareness away from the irrelevant focus. For example, Fred from Subsection 4.5 would have benefited from teaching that draws his attention to the variation in the values of instance variables and how this links to the behaviour of objects, while at the same time underlining the lack of variation with respect to constructor parameters. It seems a good idea to show him an example with multiple objects of the same type that end up being different despite having been constructed with the exact same constructor parameter values. Using variation of a critical aspect in combination with the intentional lack of variation in another should be a useful tool in dispelling incorrect understandings. Fred might also learn better from seeing and interacting with visualizations of changing object states. (See the work of Yosef et al. 2005 for one example of using a visualization tool to prevent the rise of misconceptions.) Similarly, careful misconception-aware instruction and visualizations could also help dispel the problems represented by the category VARNAME by clarifying object-variable relationships to students.

As noted above, an understanding described by the category CODE may — despite being untrue — be relatively harmless for a beginner. Even so, showing in a concrete way how classes exist as separate entities in the machine and how each object can be stored without having to store a copy of the code should help towards understanding the difference between classes and objects.

In summary, we think that the use of variation of the correct critical aspects, as suggested by variation theory, is an excellent pedagogical tool. However, the impact of any pedagogical method will vary depending on each student’s prior mental model, which is why it is very valuable for us teachers to know about the specific pitfalls (i.e., incorrect understandings) that students may be in. To help our students avoid those pitfalls, to recognize when they fall in them, and to help them out, we should be aware of at least the most common incorrect understandings related to the phenomena we teach. By recognizing what critical aspects students most often incorrectly focus on, we can make more efficient use of the pedagogical advice provided by variation theory and be better prepared to help our students construct viable models.

6 Conclusions and Future Work

This paper has reported a phenomenographic study of how introductory students view objects that have been created and stored by an object-oriented program. The results suggest that students come up with many kinds of understandings of this phenomenon, not all of which are correct. While the qualitative results presented in this paper do not give an idea of how widespread particular misunderstandings are, they do paint a somewhat worrying picture by illustrating a number of ways in which students misunderstand some of the most fundamental object-oriented concepts. This is unsurprising: these results complement earlier findings, which suggest that learning programming in general is very troublesome and difficult to many students (McCracken et al. 2001, Lister, Seppälä, Simon, Thévenaz, Adams, Fitzgerald, Fone, Hamer, Lindholm, McCarty, Moström & Sanders 2004). It is also not unprecedented that students struggle with and misunderstand the basic concepts of a field of study, as shown by physics students’ difficulties with understanding the concept of force, for instance (Hestenes et al. 1992). Teachers of programming should be aware of and look out for such misunderstandings in order to facilitate the construction of correct, viable models.

This paper contributes to computer science education research in two different ways. First, it has presented detailed results of an in-depth study of a students’ understandings of a particular phenomenon, storing objects. These results are concretized and illustrated with quotes, and highlight some pedagogically relevant misunderstandings. Second, the paper serves as an example of methodologically exploratory research, which brings together two previously disjointed branches of computer science education research: the study of misconceptions and the phenomenographic research approach. The phenomenographic approach used in this study extends traditional phenomenography by including partially incorrect understandings in a phenomenographic outcome space, and explicitly treating them as such. This approach offers a new way of studying misconceptions and linking them to dimensions of variation and to correct understandings of the phenomenon.

There are many paths for future work that can follow from the present study. The effects of visualizing the notions’ machine on students’ understanding of storing objects could be assessed. Other aspects of storing objects could be explored: for instance, this study has examined what students think objects are, but has largely ignored the issues of where or how the computer stores objects. Still other aspects of the notional machine could be explored, e.g. the structure of memory, the management of control flow in programs, and so forth. The possible relationships between students’ learning goals and field of study and the resulting understandings could be charted out.

Last but not least, the prospect of investigating misunderstandings using phenomenography seems to have potential, but needs further theoretical work. A vocabulary for discussing partially incorrect understandings in phenomenographic outcome spaces is needed, and there are many open questions. What is the best way to deal with a slew of different incorrect understandings? Exactly what kind of criteria are used to delimit a phenomenon? What exactly is the nature of the links between correct understandings and partially incorrect ones? How should incorrect understandings be reported so that they can be of use? What is the stage of a phenomenography-based research project at which it is appropriate to make judgements about the correctness of people’s understandings? Can we use phenomenography to explore people’s understandings and misunderstandings of relationships between two or more phenomena? Hopefully, this paper can serve as a basis for discussing some of these interesting issues.

Acknowledgements

My thanks to Anders Berglund and Lauri Malmi for many fruitful discussions about the nature of phenomenography and for their comments on this paper. Thanks are also due to the staff of the CS1 course for their help in organizing the programming background and attitudes questionnaire and for letting me interview their students.

References


