From Phenomenography Study to Planning Teaching

Errol Thompson
School of Computer Science,
University of Birmingham,
Birmingham, UK
+44 7951 213782
kiwiet@acm.org

ABSTRACT
A phenomenographic study uncovers variations in the way that the subjects are aware of a phenomenon. In the categories of description that represent the variations in awareness there are features that, through their variation, define the characteristics of the categories. Teaching seeks to foster a change in the way that the learner is aware of a phenomenon through opening up a space of learning. This paper outlines the way that the outcome spaces from a phenomenographic study can be used to plan a teaching programme that utilises variations in the features. It discusses a strategy for teaching programming based on a phenomenographic study of practitioner conceptions of an object-oriented program. The strategy covers features related to the nature of an object-oriented program.

Categories and Subject Descriptors
K.3.2 [Computer and Information Science Education]: Computer Science Education

General Terms
Design

Keywords
Phenomenography, teaching strategy, object-oriented programming.

1. INTRODUCTION
The “most important form of learning” involves a change in the way that the learner sees something in the world [2]. This perspective of learning involves a change in the learner’s “conceptions of aspects of reality” [3]. Since the perception of the task is an influence on the approach to the task, and consequently the learning outcomes [4], endeavouring to foster appropriate perceptions should be part of the focus in teaching.

Different perceptions have been identified in relation to student perceptions of “object” and “class” [5], and Lister et al. [6] have examined the variations of understanding present in the objects-early debate.

If learning involves a change in perception, then it is desirable to understand the perceptions that learners may have of a particular learning task and to attempt to foster change in those perceptions.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright 2010 ACM 978-1-60558-820-9/10/06...$10.00.

In order to know the desired perception of the task, it is necessary to know the perceptions that those who solve such problems have of the task [1]. It is also desirable to understand the techniques that will help a learner change their perceptions so that they can complete the desired task.

This paper presents the results of a phenomenographic study that uncovered practitioner perceptions of the nature of an object-oriented program, and uses the resulting categories of description to plan a strategy for teaching that opens up the space of learning in relation to object-oriented programming.

2. PHENOMENOGRAPHIC STUDIES
Phenomenographic studies focus on uncovering categories of description that are based on variations in the ways that observers are aware of a phenomenon. These categories are defined in terms of dimensions of variation (i.e. variations in features) used to describe the phenomenon. The outcome space can be depicted as a table that describes each category of description in terms of its referential aspect (meaning) and its structural aspect (features and relationships) [7]. Through identifying the features, and their variations in relation to both the categories of description and to each other, it is possible to identify hierarchies in understanding of the phenomenon under study.

Most phenomenographic studies have focused on a learning context in an attempt to determine either the way learners perceive a phenomenon or teachers perceive the teaching context [5, 8-10]. Thompson’s [1] study used practitioners who were involved in using the object-oriented programming paradigm to determine the different ways in which the practitioners expressed their understanding of an object-oriented program and object-oriented design. The underlying hypothesis for the study was that through understanding how practitioners were aware of a phenomenon, it would be possible to identify the desired object of learning that should be targeted in teaching object-oriented programming.

3. DEVELOPING A TEACHING STRATEGY
One approach to studying a learning environment is to explore the variations around features that are presented in order to determine the enacted object of learning and the space of learning [11]. The teacher presents a range of variations or challenges the learners to explore the phenomenon from different perspectives in order to achieve a desired lived object of learning – what the learner takes away from the learning situation.

Thompson [1] contends that if the teacher is aware of the desired level of understanding as represented by a category of description and the variations in the features (critical aspects) that help form that category, then the teacher can plan variations around those features thus creating the desired enacted object of learning (see
Further, through monitoring the teaching it should be possible to assess the actual enacted object of learning, and through monitoring what the learners have learnt it should be possible to assess the lived object of learning.

4. OBJECT-ORIENTED PROGRAMMING

In Thompson's study [1], he documented a set of categories related to the nature of an object-oriented program. These categories were developed through interviewing practitioners with a range of years of experience. The features described by the practitioners were flow of control, object usage, and problem solution. From the variations identified for these features, five categories were uncovered (see Table 1). The referential and structural descriptions of these categories summarise the movement in understanding that is occurring. The descriptions of the critical aspects provide the variations in the features used to describe the categories.

For category N1, the feature that distinguishes an object-oriented program from an imperative or procedural program is that an object-oriented program makes use of objects from the provided framework. In category N2, there is an acknowledgement that the programmer uses objects as building blocks but only because they are a requirement of the programming language. With category N3, objects are seen as a way of extending the data types used in the language. The objects in this context form a data model that reflects the real world or problem domain represented in the application. With categories N1, N2, and N3, the primary understanding of flow of control is seen as being sequential in nature although this is given less emphasis in categories N2 and N3.

### Table 1: Categories of the nature of an object-oriented program [1]

<table>
<thead>
<tr>
<th>Cat</th>
<th>Referential</th>
<th>Structural</th>
<th>Flow of Control</th>
<th>Critical Aspects</th>
<th>Problem Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>N1</td>
<td>Sequence of instructions</td>
<td>Recognizes that all programs have a flow of control that defines the order in which program instructions are executed. The order of execution is seen primarily as sequential in nature</td>
<td>Flow of control is seen as sequential</td>
<td>Objects are used from provided framework</td>
<td>Provides required functionality</td>
</tr>
<tr>
<td>N2</td>
<td>Written using an object-oriented framework</td>
<td>Sees a program as object-oriented if it uses an object-oriented framework or using an object-oriented environment</td>
<td>Flow of control is still seen as sequential but is not a major focus</td>
<td>Programmer objects are defined and used but primarily as a requirement of the language or framework</td>
<td>Provides required functionality</td>
</tr>
<tr>
<td>N3</td>
<td>Based on data types</td>
<td>Sees an object-oriented program as being based on defining new data types or on data analysis or the development of a data-driven model</td>
<td>Flow of control is not explicitly discussed or considered an issue</td>
<td>Objects are defined based on data requirements</td>
<td>A data model of the problem space (real world model)</td>
</tr>
<tr>
<td>N4</td>
<td>Based on interacting entities</td>
<td>Sees an object-oriented program as a set of behavioural entities that interact to achieve the required objective</td>
<td>Flow of control is defined by the interactions between objects</td>
<td>Objects are defined based on behavioural requirements</td>
<td>A behaviour-driven model reflecting objects in the problem space</td>
</tr>
<tr>
<td>N5</td>
<td>Artificial construct</td>
<td>Sees a program as an artificial construct imposed by operating systems and not applicable to the discussion of object-oriented entities</td>
<td>Flow of control is defined by the interactions between objects</td>
<td>Objects are liberated from the confines of program boundaries</td>
<td>A behaviour-driven model using useful abstractions</td>
</tr>
</tbody>
</table>
N3. With category N4, there is a clear shift away from a sequential flow of control to a focus on interactions between behavioural oriented entities represented by objects. The model is now behavioural driven rather than data-driven but it still endeavours to reflect entities in the real world. With category N5, there is recognition that the objects used may not reflect real world entities nor be confined to a specific program or context. However, the shift to this level of understanding is less dramatic than the change between levels N3 and N4 where there is both a change in the understanding of the flow of control and in the understanding of object usage.

5. PLANNING A TEACHING STRATEGY
From studies into the use of variations in teaching of concepts, Marton, et al. [12] define four patterns of variation used in defining the space of learning. These are contrast, generalization, separation, and fusion. The use of contrasting variations gives the learner a point of comparison for the phenomenon. It enables them to distinguish the phenomenon from other related phenomenon. The use of generalization provides the learner with variations in features of the phenomenon thus providing a broader understanding. Separation focuses on helping the learner become aware of the features or aspects of the phenomenon by examining the phenomenon from different angles. One feature is varied while the other features remain constant. Fusion is where several features are considered together thus fostering simultaneous awareness of those features. A teaching strategy based on the use of variations should include all four of these patterns of variation.

Learning theory [13] also places emphasis on the importance of connecting with background knowledge of the learner. By using phenomenon familiar to the learner, it is possible to draw on existing knowledge and to extend this into the new domain. In the context of programming, the notion of a programme is used in a wide range of contexts. Using programme examples familiar to the learner provides an initial framework for drawing out the features that distinguish a programme from other phenomenon or the features that provide a distinction between programming paradigms.

6. A TEACHING STRATEGY
The starting point for the teaching strategy is to foster an appropriate perception of the nature of an object-oriented program. The level of understanding sought is that represented by category N4. The learner should recognise that the flow of control is based on interacting entities and that objects are based on behavioural abstractions that primarily represent entities in the real world. The use of useful abstractions is seen as a desirable learning outcome but objects liberated from the confines of program boundaries is seen as beyond the scope of the teaching strategy.

6.1 FLOW OF CONTROL
This teaching strategy sees the opening teaching theme focus on building the appropriate flow of control perception using contrasts and separation. Contrasts are established by trying to reveal the different forms of flow of control while avoiding the discussion of how the data is dealt with.

By utilising the learner’s background knowledge, common real world items that might be seen or confused with programmes are used as the starting point. Sequential flow of control is represented by recipes, TV programmes, and instructions for building something [14-15]. When contrasted with shopping lists and tables of contents, the distinction between no flow of control and sequential flow can be discussed and brought into focal awareness. Contrasting sequential flow of control programmes with asking a friend for information without really knowing where the friend obtained that information or asking a friend to do something without knowing how the friend may have performed the task brings in the notion of interactions and object use. Care needs to be taken to ensure in the first instance that only one feature is focused on (separation). The learner simply knows the friend is capable of providing the information or doing what is requested. Other interaction examples are the interactions that occur between people in the work place where people are assigned different tasks that are performed based on the arrival of requests or information (the message), or a football team where each player takes on their responsibility as the ball (the message) is passed from player to player. These metaphors of a programme have an element of multitasking or responding to alternative messages from other sources (i.e. the footballer observing and responding to state of the game) that isn’t there by default in most object-oriented programs.

A key principle of object-oriented programs is that an object encapsulates the behaviour and hides the nature of the internal implementation. The calling object or code has no idea of the programming paradigm used within the object or whether multitasking is involved. The calling object or code has an expectation of the behaviour of the called object but no knowledge of how that expectation will be satisfied.

In utilizing real world examples, the objective is to draw out the feature of interacting entities. Examining the difference between the sequential flow of control and interacting entities provides a point of contrast. Using multiple examples from each category provides for generalisation of the variations in flow of control. This provides a base for the initial experiencing of the desired form of flow of control.

6.2 OBJECT USAGE
The second theme in the teaching strategy focuses on object usage. Providing illustrations of the different forms of object usage is difficult. Revisiting the previous examples and focusing on the way that data is handled and behaviour is hidden provides an initial framework for discussing object usage. The contrast in data usage from a recipe (ingredients as data) with the encapsulation of behaviour and data in a person in the interacting examples provides an initial basis for discussion of objects that encapsulate both data and behaviour.

Utilising pre-existing objects is represented through examples of utilising tools or machinery. The tools encapsulate properties or behaviours that are used to perform desired tasks. However, these examples don’t make the distinction of using objects because they are a requirement of using the language. A static tool doesn’t foster the notion of interaction. The use of a machine does have

1 UK English spelling has been used deliberately to distinguish between programmes that are not computer programs and computer programs.
Finding examples that illustrate the notion of an abstract data type is more difficult. If the students have prior knowledge of data types possibly from computer architecture courses then these can be used to discuss the nature of a data type and to draw out the key ideas in defining an abstract data type and how this contrasts with behavioural objects. An example is the representation of the IEEE floating point arithmetic standard. There are issues of how a number is represented and there are rules for defining the way that operations behave on the numbers.

The primary focus of this portion of the teaching strategy is to draw out the conceptual understanding of using an object to encapsulate behaviour. Although it is desirable to illustrate the alternative variations in object usage, it isn’t essential.

### 6.3 PROGRAM CODE EXAMPLES

With a conceptual foundation laid with respect to the nature of an object-oriented program, the teaching strategy in its next phase focuses on program code.

#### 6.3.1 Reading before writing

In line with research that identifies a link between an ability to read and trace code, and the ability to write code [16], the teaching strategy introduces code examples for the learner to read, trace, and discuss prior to endeavouring to teach the writing of code. Each code example introduces a variation in usage of or a new feature of the language or programming paradigm. The code examples also utilise the desired design characteristics identified in the study [1].

#### 6.3.2 Simple interactions

The first code examples focus on interactions by examining the use of function calls. To illustrate function concepts, a function call is compared with asking a friend to perform a simple calculation such as calculating the area of a square when given the length of a side. The code example has a single parameter and returns the area.

Code examples are provided to the learner so that they become familiar with the syntax for the function and the operations for performing simple calculations. The emphasis in these examples is on code reading and identifying the semantics of the code rather than writing.

The variations in code examples shown reinforce the principle of interaction. The initial examples are simple calculations but these quickly move to examples of functions calling other functions in order to perform more complex calculations. The focus is on applying the variation pattern of generalization to build a broader understanding of interactions and function usage.

#### 6.3.3 The Object Puzzle

A stepping stone from reading and tracing to writing is the use of object puzzles. The object puzzle provides the learner with a set of objects (i.e. building blocks) that they have to connect to implement the appropriate object interactions in order to achieve the desired functionality. Building an object puzzle is like making a jigsaw puzzle. The learner is looking for the interlocking behaviours that bind the objects together. The use of these puzzles leads to the use of objects from provided frameworks and to the discussion of the concept of type and interfaces. In these exercises, the focus is not on the internals of the objects but the interdependencies or interfaces of the objects.

With the focus on interaction, the use of these puzzles applies the variation patterns of separation and generalisation. As the learner becomes more familiar with using object interactions, the focus moves to a discussion of object interfaces as defining type and object behaviour through using the variation pattern of fusion.

#### 6.3.4 Testing code

Having introduced the concept of type and object behaviours through object puzzles, the next exercises are related to testing of object behaviour. In the teaching, emphasis is placed on object interactions as the foundation for being able to write tests. Initially, the learner is provided with example unit test cases. These are analysed in terms of how they cover the range of behaviours expected of the objects. The focus at this stage is still on exercising simple methods of objects.

The first step into code writing occurs at this point with the learner developing a test strategy for method behaviour and then implementing that strategy in a unit test. Here the learner is introduced to the issues of designing test code based on requirements. This lays the foundation for a test-driven or test-first approach to code writing and design [17-20].

#### 6.3.5 Introducing writing

Using a framework for writing tests for code now enables the writing of simple methods based on the translation of requirements to executable tests and then the simple function code to satisfy those tests. From this point on in the course, new concepts or constructs are introduced through reading exercises, object puzzles, and test code that leads to the writing of code. Following the pattern of Barnes and Kölling [21], this ensures that the complexity of code writing task is based on small incremental steps and variations, some of the initial code writing will involve the modification of existing code.

#### 6.3.6 Introducing object states

Objects that simply provide a set of methods but have no state are not that useful in writing programs but the learner now has a set of tools for exploring more complex objects and code. At this point in the teaching strategy, the learner is introduced to objects that have state and functions that require some data to be stored and shared between them.

The issue of object state is initially introduced through the use of simple getters and setters, and calculations based on stored values. The concept of a constructor and value object is then introduced. The variations in complexity are always taken in small steps reinforcing current usage (generalisation and fusion) and increasing complexity through the introduction of single features or concepts (contrast and separation). Care is taken in the design of the exercises to ensure that the point of variation is emphasizing the new feature or concept initially through contrast with prior solutions and then through alternative examples of its use (generalisation). The focus at this point is on encapsulating behaviour in objects and the issue of an object retaining state.

The use of object state is initially introduced as a reading exercise using simple interactions followed by the use of object puzzles with objects with state, and the use of test code to ensure that the
6.3.7 Introducing logic and design concepts

Once the foundation of interaction is well established, logic constructs are introduced. Again each construct is introduced as a variant on existing implementations.

With an emphasis on introducing design concepts, polymorphism and inheritance are introduced through examples such as the drawing of shapes [21] and the comparing of conditionals based on the use of an 'if' statement with a solution to the same problem using polymorphism [24]. This use of contrasting examples helps reinforce the desired understanding of interacting entities and object usage.

As with the introducing of objects with state, each concept is introduced through reading exercises, interactions, object puzzles, and testing code that leads to the writing of code. The repeated learning pattern reinforces an approach to learning programming concepts that will provide a tool for learning beyond the current course. This pattern also reinforces the patterns of variation with respect the concepts providing opportunities to utilise contrast, generalisation, separation, and fusion as each concept is introduced and developed.

7. CONCLUSION

The teaching strategy outlined in this paper draws on the research results of the phenomenographic study conducted into object-oriented programming [1]. These results are used to provide the overriding structure and direction as well as the variations to be used in teaching. It emphasises the use of variation theory as a key component in opening up concepts in the space of learning. The teaching strategy builds on sound educational practice drawn from other areas of educational [13] and computing education research [16,19-20,23], and also draws on practitioner experience and patterns [17-18,22].

8. REFERENCES